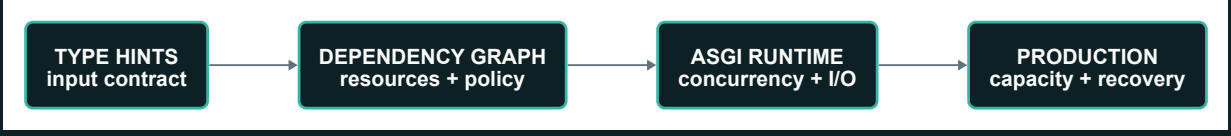
ΑΠΟ ΤΟ ΠΡΩΤΟ TYPED ENDPOINT ΣΕ PRODUCTION SYSTEM

FastAPI

μέχρι Production

Python types, Pydantic, ASGI, concurrency και correctness χωρίς framework magic.



30

Από function σε ολόκληρο σύστημα

ΚΕΦΑΛΑΙΑ · FASTAPI 0.141.1 · PYDANTIC 2.13.5

ΠΡΙΝ ΓΡΑΨΟΥΜΕ ΤΟ ΠΡΩΤΟ ROUTE

Τα types περιγράφουν το contract. Δεν εκτελούν το σύστημα για σένα.

Το FastAPI κάνει το πρώτο σωστό endpoint εντυπωσιακά μικρό. Διαβάζει type hints, επικυρώνει input, παράγει OpenAPI και μετατρέπει το αποτέλεσμα σε response. Αυτή η άνεση είναι χρήσιμη, αλλά μπορεί να κρύψει τα πραγματικά όρια: πότε τρέχει blocking code, ποιος ανοίγει μια transaction, πότε κλείνει ένα dependency με `yield`, ποιος κρατά connection pool και τι έχει ήδη σταλεί όταν αποτυγχάνει ένα stream.

Το βιβλίο ξεκινά χωρίς να θεωρεί ότι ξέρεις FastAPI. Χρειάζεται μόνο βασική Python: functions, classes, exceptions και type hints. Στα πρώτα κεφάλαια χτίζουμε request και response models. Μετά ανοίγουμε το dependency graph, το Pydantic pipeline και το ASGI boundary. Στο τελευταίο τρίτο περνάμε σε transactions, concurrency, queues, WebSockets, backpressure, security και production releases.

Πώς διαβάζεται

Τα κεφάλαια 1 έως 10 φτιάχνουν τα θεμέλια. Τα 11 έως 20 χωρίζουν transport, application και data concerns. Τα 21 έως 30 εξετάζουν τι αλλάζει όταν το ίδιο σύστημα έχει παράλληλα requests, πολλούς workers, αργά dependencies και rolling deployments.

Κάθε κεφάλαιο έχει `SOURCE` `TRACE` και `PROBE`. Το πρώτο σε οδηγεί στην πραγματική υλοποίηση της έκδοσης αναφοράς. Το δεύτερο είναι μικρό πείραμα για να δεις observable behavior. Τα internals δεν είναι public API. Είναι ο τρόπος να καταλαβαίνεις το framework και να ξαναβρίσκεις την αλήθεια σε ένα upgrade.

Έκδοση αναφοράς

Η έκδοση 1.0 γράφτηκε τον Σεπτέμβριο του 2026 με βάση το FastAPI 0.141.1 και τις τότε stable εκδόσεις Pydantic 2.13.5, Starlette 1.6.0, Uvicorn 0.52.4, HTTPX 0.28.1 και SQLAlchemy 2.0.52. Το FastAPI 0.141.1 απαιτεί Python 3.10 ή νεότερη. Τα snippets ελέγχθηκαν συντακτικά με Python 3.14.7. Δεν αποτελούν από μόνα τους integration test matrix για κάθε συνδυασμό dependencies.

Δεν χτίζουμε abstractions επειδή «έτσι γίνεται σε enterprise». Προσθέτουμε boundary μόνο όταν έχει owner, invariant ή failure mode που πρέπει να φαίνεται.

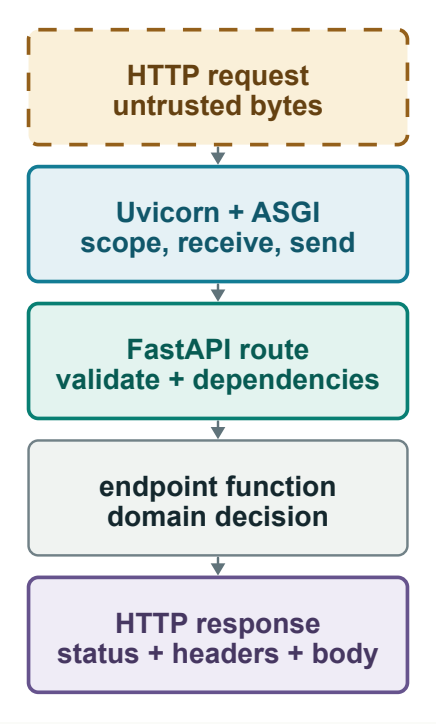
Ο ΧΑΡΤΗΣ ΤΟΥ ΒΙΒΛΙΟΥ

Περιεχόμενα

ΚΕΦΑΛΑΙΟ 1

Το πρώτο endpoint και ο χάρτης του request

Ένα FastAPI endpoint είναι Python function μέσα σε ένα ASGI σύστημα. Τα type hints περιγράφουν το HTTP contract, αλλά το request συνεχίζει να είναι bytes, I/O και failure modes.



Σχήμα 1.1 · Από τα bytes του client μέχρι το public response.

Ξεκινάμε με το μικρότερο χρήσιμο API. Το αρχείο `main.py` δημιουργεί μια εφαρμογή και καταχωρίζει μία path operation. Η function δεν καλείται όταν γίνεται import το module. Ο decorator διαβάζει τη δήλωση και προσθέτει route. Η function θα κληθεί αργότερα, για κάθε request που ταιριάζει.

```python
from fastapi import FastAPI
from pydantic import BaseModel

app = FastAPI(title="Catalogue API")


class Health(BaseModel):
    service: str
    status: str


@app.get("/health", response_model=Health)
async def health() -> Health:
    return Health(service="catalogue", status="ok")
```

Με εγκατεστημένο το standard extra, το development command είναι `fastapi dev main.py`. Για production υπάρχει το `fastapi run`, ενώ μπορείς να τρέξεις και ASGI server απευθείας, για παράδειγμα `uvicorn main:app --reload` μόνο σε development. Το object μετά την άνω τελεία είναι η ASGI εφαρμογή που φορτώνει ο server.

Το `/docs` είναι interactive documentation και το `/openapi.json` είναι το machine-readable schema. Δεν είναι ανεξάρτητη περιγραφή που γράφτηκε στο χέρι. Παράγεται από route metadata, types και Pydantic models. Αυτό μειώνει drift, αλλά δεν αποδεικνύει ότι οι business κανόνες είναι σωστοί.

Τι κάνει το type hint

Το `response_model=Health` δηλώνει public output. Το FastAPI επικυρώνει το return value και κρατά μόνο fields που ανήκουν στο model. Αν η function επιστρέψει λάθος shape, αυτό είναι server bug και όχι client validation error. Το return annotation βοηθά editor και type checker. Το `response_model` είναι η ρητή HTTP δήλωση και μπορεί να χρησιμοποιηθεί ακόμη κι όταν ο εσωτερικός return type είναι διαφορετικός.

Σε input arguments η πηγή προκύπτει από τον συνδυασμό path, annotation και FastAPI metadata. Ένα scalar όνομα που υπάρχει στο path είναι path parameter. Ένα άλλο scalar γίνεται συνήθως query parameter. Ένα `BaseModel` γίνεται body. Αργότερα θα κάνουμε κάθε αμφίβολη πηγή explicit με `Path`, `Query`, `Header`, `Cookie` και `Body`.

Ο ASGI πυρήνας

Ο Uvicorn μεταφράζει HTTP σε τρία ASGI στοιχεία: `scope`, `receive` και `send`. Το scope περιέχει metadata όπως method, path και headers. Με το `receive` η εφαρμογή παίρνει body events ή disconnect. Με το `send` εκπέμπει response start και body events. Το FastAPI κληρονομεί το transport layer από το Starlette και προσθέτει routing με validation, dependency injection και OpenAPI.

Αυτό εξηγεί δύο σημαντικά πράγματα. Πρώτον, το `async def` δεν δημιουργεί νέο thread για κάθε request. Η coroutine μοιράζεται event loop και παραχωρεί τον έλεγχο μόνο σε `await` που πραγματικά περιμένει async I/O. Δεύτερον, όταν επιστρέψεις ένα Starlette `Response`, αυτό είναι το ASGI callable που τελικά θα στείλει status, headers και body.

`def` ή `async def`;

Χρησιμοποίησε `async def` όταν οι βιβλιοθήκες που καλείς έχουν async API και κάνεις `await`. Χρησιμοποίησε απλό `def` όταν το endpoint καλεί blocking library. Το FastAPI εκτελεί sync endpoint σε thread pool ώστε να μη μπλοκάρει άμεσα το event loop. Αυτό δεν κάνει το blocking work φθηνό. Τα threads και οι downstream connections παραμένουν πεπερασμένοι πόροι.

Μην γράφεις `async def` γύρω από synchronous database driver και μετά θεωρείς ότι έγινε non-blocking. Το blocking call μέσα σε coroutine κρατά ολόκληρο το event loop. Το σωστό μοντέλο είναι end-to-end async ή καθαρό sync boundary που πηγαίνει στο thread pool.

Το πρώτο σωστό mental model

Η route function είναι transport adapter. Παραλαμβάνει ήδη validated values, καλεί application code και μεταφράζει outcome σε HTTP. Δεν πρέπει να γίνει ταυτόχρονα ORM model, authorization engine, transaction manager και client τρίτου provider. Στην αρχή αυτό φαίνεται υπερβολικό. Μόλις εμφανιστούν retries, δύο workers ή ένα partial failure γίνεται η διαφορά ανάμεσα σε demo και σύστημα.

SOURCE TRACE
Στο tag `0.141.1`, η `FastAPI` βρίσκεται στο `fastapi/applications.py` και κληρονομεί από `Starlette`. Η `APIRoute.get_route_handler` στο `fastapi/routing.py` καταλήγει σε `get_request_handler`, ενώ το `request_response` μετατρέπει τον handler σε ASGI callable με `scope`, `receive` και `send`.

ΠΑΓΙΔΑ
Το ότι το `/docs` δείχνει πράσινο response δεν ελέγχει transaction ownership, authorization ή συμπεριφορά σε timeout. Είναι explorer του contract, όχι production verification.

ΜΗΧΑΝΙΣΜΟΣ
Types και metadata δηλώνουν το boundary. Ο ASGI server κινεί το I/O. Το endpoint πρέπει να μείνει μικρός adapter ανάμεσα στο HTTP και στο application workflow.

PROBE
Βάλε ένα `print` αμέσως πριν από τον decorator, ένα μέσα στο endpoint και ένα μέσα σε dependency. Ξεκίνα τον server και στείλε δύο requests. Κατέγραψε τι τρέχει μία φορά στο import και τι τρέχει ανά request. Μετά άλλαξε το endpoint από `async def` σε `def` και τύπωσε το thread name για να δεις το boundary.

ΚΕΦΑΛΑΙΟ 2

Περιβάλλον, application factory και lifespan

Boot είναι η στιγμή που συνθέτεις το process. Lifespan είναι η στιγμή που ανοίγεις και κλείνεις shared resources με σαφή σειρά.

Σχήμα 2.1 · Το process περνά από import, lifespan enter, serving και cleanup.

Ένα virtual environment απομονώνει τις Python packages του project. Το version control πρέπει να κρατά declarations και lock data, όχι το directory του environment. Δήλωσε άμεσα dependencies όπως FastAPI, Uvicorn και Pydantic Settings με συγκεκριμένη πολιτική versioning. Μην αφήνεις ένα production image να λύνει ξανά απεριόριστες εκδόσεις σε κάθε build.

Μια μικρή δομή αρκεί:

OUTPUT

```
app/
  __init__.py
  main.py
  api.py
  settings.py
  domain/
  adapters/
tests/
pyproject.toml
```

Το `main.py` είναι composition root. Εκεί ενώνονται settings, clients, repositories και routers. Δεν είναι χώρος για business logic. Μια application factory επιτρέπει σε test ή διαφορετικό process να δημιουργήσει app με άλλη configuration χωρίς να αλλάξει globals.

PYTHON

```
from collections.abc import AsyncIterator
from contextlib import asynccontextmanager

from fastapi import FastAPI, Request


class Catalogue:
    async def close(self) -> None:
        pass
```

```
def build_catalogue() -> Catalogue:
    return Catalogue()


def create_app() -> FastAPI:
    @asynccontextmanager
    async def lifespan(app: FastAPI) -> AsyncIterator[None]:
        app.state.catalogue = build_catalogue()
        try:
            yield
        finally:
            await app.state.catalogue.close()

    api = FastAPI(title="Catalogue API", lifespan=lifespan)

    @api.get("/health")
    async def health(request: Request) -> dict[str, str]:
        assert request.app.state.catalogue is not None
        return {"status": "ok"}

    return api


app = create_app()
```

Το object `app` παραμένει συμβατικό import target για τον server. Η factory όμως απομονώνει τη σύνθεση. Αν αργότερα χρειαστείς CLI worker, μπορεί να χρησιμοποιήσει το ίδιο settings loader και adapters χωρίς να κάνει import το HTTP app και να καταχωρίζει routes χωρίς λόγο.

Import time δεν είναι startup

Το module import μπορεί να συμβεί σε test discovery, documentation build, worker preload ή command line script. Network calls, migrations και μεγάλα model loads στο top level κάνουν κάθε import επικίνδυνο. Επίσης, αν ο process forkάρει μετά το import, ένα ήδη ανοιχτό socket ή connection pool μπορεί να μοιραστεί με λάθος τρόπο ανάμεσα σε workers.

Κράτα στο import μόνο φθηνές, deterministic δηλώσεις. Άνοιξε network clients, database engines και consumers στο lifespan enter. Κλείσε τα στο αντίστροφο μονοπάτι μετά το `yield`. Αν ένα resource αποτύχει να ανοίξει, το process δεν πρέπει να δηλωθεί ready.

Ένα owner για κάθε resource

Το `app.state` είναι process-local storage. Δεν συγχρονίζεται ανάμεσα σε workers και δεν είναι database. Είναι κατάλληλο για objects όπως connection pools ή immutable registries που ανήκουν στη ζωή ενός worker. Απόφυγε να γεμίσεις route functions με untyped lookups στο state. Βάλε μικρό dependency που επιστρέφει συγκεκριμένο protocol ή object. Έτσι τα endpoints δηλώνουν τι χρειάζονται και τα tests μπορούν να κάνουν override.

Resource που ανοίγει στο lifespan πρέπει να έχει σαφή shutdown deadline. Πρώτα σταματάς να δέχεσαι νέα εργασία, μετά περιμένεις bounded χρόνο για in-flight work, μετά κλείνεις clients και pools. Αν η εφαρμογή ξεκινά consumer task, κράτα handle, στείλε cancellation και περίμενε την ολοκλήρωσή του. Ένα ορφανό `create_task` δεν έχει lifecycle owner.

Readiness και liveness

Liveness σημαίνει ότι ο process δεν έχει κολλήσει τελείως. Readiness σημαίνει ότι μπορεί να εξυπηρετήσει τώρα. Μην κάνεις database query σε κάθε liveness probe και προκαλείς restart storm όταν πέσει η database. Η readiness μπορεί να αντανακλά critical startup state, αλλά χρειάζεται μικρό timeout και φθηνή πολιτική. Ένα health endpoint που επιστρέφει πάντα 200 χωρίς να ξέρει αν τελείωσε το lifespan δεν είναι readiness.

Οι migrations δεν πρέπει να τρέχουν ταυτόχρονα από κάθε web worker. Είναι ξεχωριστό release step με έναν owner. Το lifespan επαληθεύει συμβατότητα και ανοίγει runtime resources. Δεν αλλάζει ανεξέλεγκτα shared schema.

SOURCE TRACE

Το `FastAPI` περνά το `lifespan` στον router στο `fastapi/applications.py`. Στο `fastapi/routing.py`, το `_merge_lifespan_context` συνθέτει lifespan contexts όταν περιλαμβάνονται routers. Η εκτέλεση των ASGI lifespan events παραμένει ευθύνη του server και του Starlette-compatible routing layer.

ΠΑΓΙΔΑ

Το `app.state` αντιγράφεται ως ιδέα σε κάθε worker, όχι ως κοινή μνήμη. Counter, cache ή lock εκεί δεν προστατεύει cluster-wide invariant.

ΜΗΧΑΝΙΣΜΟΣ

Import δηλώνει το πρόγραμμα. Lifespan κατέχει τα long-lived resources. Request dependencies δίνουν typed πρόσβαση χωρίς να αλλάζουν το lifecycle τους.

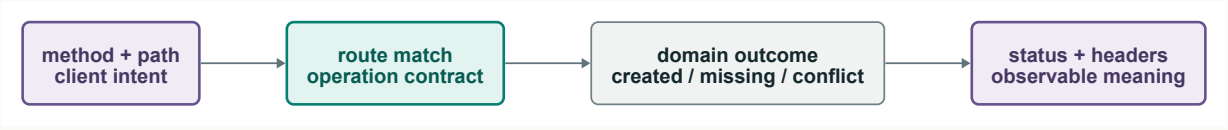
PROBE

Φτιάξε fake resource που καταγράφει `open`, `use`, `close`. Τρέξε δύο requests μέσα σε `TestClient` context και μετά ένα test χωρίς context manager. Assert μία έναρξη, δύο χρήσεις και ένα κλείσιμο. Πρόσθεσε exception στο startup και επιβεβαίωσε ότι κανένα request δεν εξυπηρετείται ως ready.

ΚΕΦΑΛΑΙΟ 3

Path operations και HTTP semantics

To route δεν είναι απλώς URL που καλεί function. Method, status, headers και response body περιγράφουν τι συνέβη και τι επιτρέπεται να κάνει ο client μετά.



Σχήμα 3.1 · Το domain outcome μεταφράζεται σε observable HTTP meaning.

Οι decorators `get`, `post`, `put`, `patch` και `delete` καταχωρίζουν path operations. Το όνομα της function είναι Python detail. Το public contract είναι ο συνδυασμός method, path, accepted input, outcomes και response schema. Ένας client δεν πρέπει να μαντεύει το αποτέλεσμα από human message.

Το GET είναι για ανάγνωση και πρέπει να είναι safe ως προς business mutation. Το PUT αντικαθιστά resource σε γνωστό URI και είναι σχεδιασμένο ώστε η ίδια πρόθεση να μπορεί να επαναληφθεί. Το PATCH εφαρμόζει μερική αλλαγή. Το POST δημιουργεί subordinate resource ή εκτελεί command και δεν είναι εγγενώς idempotent. Το DELETE μπορεί να σχεδιαστεί ώστε η επανάληψη να διατηρεί το ίδιο τελικό state, παρότι το δεύτερο response ίσως είναι διαφορετικό.

PYTHON

```python
from uuid import UUID, uuid4

from fastapi import FastAPI, Header, HTTPException, Response, status
from pydantic import BaseModel

app = FastAPI()


class WidgetCreate(BaseModel):
    name: str


class WidgetView(BaseModel):
    id: UUID
    name: str


@app.post(
    "/widgets",
    response_model=WidgetView,
    status_code=status.HTTP_201_CREATED,
)
async def create_widget(data: WidgetCreate, response: Response) -> WidgetView:
    widget = WidgetView(id=uuid4(), name=data.name)
    response.headers["Location"] = f"/widgets/{widget.id}"
    return widget


@app.get("/widgets/{widget_id}", response_model=WidgetView)
```

```
async def get_widget(widget_id: UUID) -> WidgetView:
    raise HTTPException(status_code=404, detail="widget_not_found")
```

Το 201 δηλώνει δημιουργία και το `Location` δείχνει το νέο resource. Το 202 σημαίνει ότι η εργασία έγινε accepted, όχι ότι ολοκληρώθηκε. Πρέπει να δώσεις status resource ή άλλο protocol παρακολούθησης. Το 204 δεν έχει response body. Το 409 περιγράφει conflict με τρέχον state. Το 412 ταιριάζει σε αποτυχημένο precondition όπως `If-Match`. Το 422 χρησιμοποιείται από το FastAPI για request validation failures.

Στατικές και δυναμικές διαδρομές

Τα routes αξιολογούνται με σειρά. Αν δηλώσεις `/users/{user_id}` πριν από `/users/me`, το string `me` μπορεί να αντιμετωπιστεί σαν dynamic value και να αποτύχει validation ή να πάει σε λάθος handler. Δήλωσε τις συγκεκριμένες διαδρομές πριν από τις γενικές. Path converters και annotations δεν είναι υποκατάστατο για καθαρό URL design.

Το trailing slash επίσης είναι observable. Redirect behavior μπορεί να επηρεάσει non-GET clients, signatures ή proxies. Διάλεξε canonical paths και δοκίμασε τι γίνεται όταν ο client στείλει άλλη μορφή. Μην θεωρείς ότι κάθε HTTP client επαναλαμβάνει με ασφάλεια body μετά από redirect.

Domain outcome πριν από HTTP mapping

Μην πετάς `HTTPException` βαθιά μέσα στο domain επειδή είναι βολικό. Ένα use case μπορεί να επιστρέψει outcome όπως `Created`, `NotFound`, `Conflict` ή να σηκώσει application exception που δεν γνωρίζει status codes. Ο route adapter κάνει τη μετάφραση. Έτσι το ίδιο workflow μπορεί να κληθεί από queue ή CLI.

Το `HTTPException` είναι σωστό στο transport boundary και σε dependencies που εκφράζουν HTTP policy, όπως missing credentials. Το `detail` είναι public data. Μην βάζεις stack trace, SQL message ή token εκεί.

Όταν επιστρέφεις `Response`

Αν επιστρέψεις dict ή model, το FastAPI περνά από response serialization και validation. Αν επιστρέψεις απευθείας `Response`, `JSONResponse`, file ή stream, αναλαμβάνεις status, headers, media type και body. Αυτό είναι χρήσιμο, αλλά το runtime αποτέλεσμα δεν φιλτράρεται από `response_model`. Κράτα documentation και πραγματικό response ευθυγραμμισμένα.

Headers όπως `ETag`, `Cache-Control`, `Location`, `Retry-After` και `Content-Disposition` είναι μέρος του contract. Μην τα αντιμετωπίζεις σαν διακοσμητικό metadata. Ένα σωστό status με λάθος caching header μπορεί να παράγει σοβαρότερο bug από ένα λάθος JSON field.

> **SOURCE TRACE**
> Οι convenience decorators καταλήγουν σε `FastAPI.add_api_route` και `APIRouter.add_api_route`. Στο `fastapi/routing.py`, η route state κρατά methods, status code, response class και response fields. Το actual matching βασίζεται στο Starlette routing contract και διατηρεί τη σειρά των routes.

> **ΠΑΓΙΔΑ**
> Το `200 {"ok": true}` για κάθε αποτέλεσμα αφαιρεί πληροφορία από proxies, clients και monitoring. Τα HTTP semantics είναι μέρος του API, όχι περιτύλιγμα γύρω από ένα generic envelope.

ΜΗΧΑΝΙΣΜΟΣ

Πρώτα μοντελοποίησε outcomes. Μετά αντιστοίχισε καθένα σε status, headers και body. Η αντιστοίχιση πρέπει να είναι σταθερή και να δοκιμάζεται ως contract.

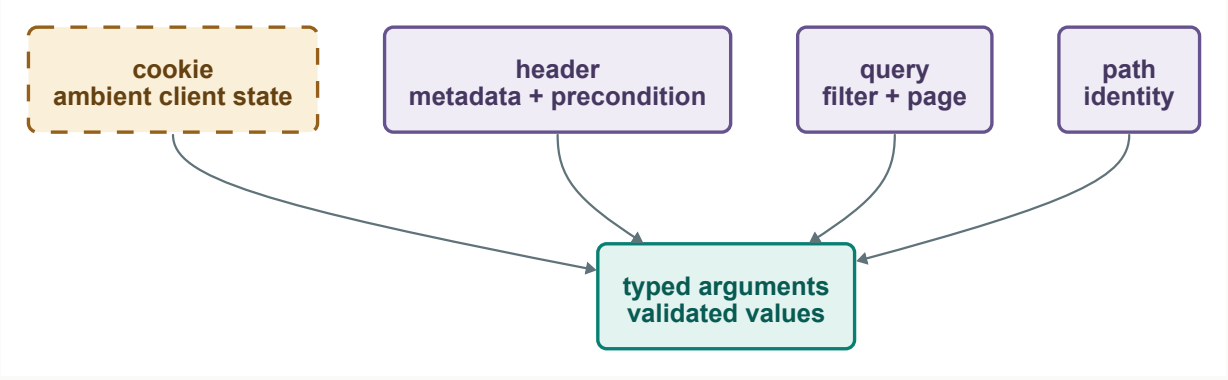
PROBE

Γράψε tests για create, duplicate create, missing resource και stale update. Assert ακριβώς status, `Location`, body και absence body στο 204. Πρόσθεσε `/users/me` μετά από `/users/{user_id}` και παρατήρησε το αποτέλεσμα. Μετά αντιστροφή της σειράς και κλείδωσέ το με regression test.

ΚΕΦΑΛΑΙΟ 4

Path, query, header και cookie parameters

Η ίδια string τιμή έχει διαφορετικό νόημα ανάλογα με το σημείο του HTTP request από όπου προήλθε. Κάνε source, constraints και absence explicit.



Σχήμα 4.1 · Τέσσερις πηγές input καταλήγουν σε typed arguments.

Το path προσδιορίζει resource identity. Το query εκφράζει filtering, sorting, pagination ή optional projection. Τα headers μεταφέρουν protocol metadata, credentials και preconditions. Τα cookies είναι ambient state που ο browser στέλνει σύμφωνα με domain και cookie policy. Αν τα ανακατέψεις σε ένα generic `params` dictionary, χάνεις validation και νόημα.

Χρησιμοποίησε `Annotated` ώστε το Python type να μένει καθαρό και το FastAPI metadata να βρίσκεται δίπλα του. Constraints στο boundary κόβουν άκυρο input πριν φτάσει στο use case, αλλά δεν αντικαθιστούν domain invariants.

PYTHON

```
from typing import Annotated
from uuid import UUID

from fastapi import Cookie, FastAPI, Header, Path, Query

app = FastAPI()


@app.get("/accounts/{account_id}/orders")
async def list_orders(
    account_id: Annotated[UUID, Path(description="Account identity")],
    limit: Annotated[int, Query(ge=1, le=100)] = 20,
    cursor: Annotated[str | None, Query(max_length=200)] = None,
    tags: Annotated[list[str] | None, Query()] = None,
    if_none_match: Annotated[
```

```
        str | None, Header(alias="If-None-Match")
    ] = None,
    session_id: Annotated[
        str | None, Cookie(alias="session")
    ] = None,
) -> dict[str, object]:
    return {
        "account_id": account_id,
        "limit": limit,
        "cursor": cursor,
        "tags": tags or [],
        "conditional": if_none_match is not None,
        "has_session": session_id is not None,
    }
```

Το `UUID` απορρίπτει malformed identity πριν το repository. Το `limit` οριοθετεί κόστος. Το cursor έχει length bound επειδή αργότερα μπορεί να αποκωδικοποιηθεί ή να μπει σε log. Επαναλαμβανόμενο query όπως `?tags=paid&tags=priority` γίνεται list. Το absence δηλώνεται με `None`, όχι με κενό string που αποκτά δέκα διαφορετικές ερμηνείες.

Naming και aliases

Python identifiers γράφονται συνήθως με underscore, ενώ public APIs μπορεί να χρησιμοποιούν άλλο όνομα. Το `alias` κρατά explicit mapping. Στα headers το FastAPI μετατρέπει underscore σε hyphen από default. Για protocol headers προτίμησε explicit alias ώστε ο αναγνώστης να βλέπει ακριβώς το wire name.

Μην αφήνεις public names να αλλάζουν επειδή έγινε rename μια local μεταβλητή. Path και query names είναι deployed contract. Αν πρέπει να μετακινηθεί field, σχεδίασε migration window και παρακολούθησε χρήση του παλιού alias.

Pagination που αντέχει αλλαγές

Το `offset` είναι απλό, αλλά σε μεγάλη και μεταβαλλόμενη συλλογή έχει αστάθεια και αυξανόμενο database cost. Cursor pagination κωδικοποιεί stable ordering key και tie-breaker. Ο cursor είναι opaque για τον client, signed αν δεν πρέπει να αλλοιώνεται και versioned ώστε να μπορεί να εξελιχθεί.

Το order πρέπει να είναι deterministic. `created_at` μόνο του δεν αρκεί αν δύο rows έχουν ίδιο timestamp. Πρόσθεσε unique identity ως δεύτερο key. Η εφαρμογή ορίζει maximum page size ακόμη κι αν ο client ζητήσει εκατομμύριο. Το response δίνει next cursor και όχι εσωτερικό SQL offset.

Headers και trust

Τα απλά request headers είναι input του client. Forwarded host, scheme και client IP μπορεί να έχουν ξαναγραφτεί από proxy. Μην εμπιστεύεσαι `X-Forwarded-For` επειδή υπάρχει. Ο ASGI server πρέπει να εμπιστεύεται μόνο γνωστές proxy addresses και η edge topology να καθορίζει ποιο hop γράφει τι.

Τα conditional headers χρειάζονται parser. Το `If-Match` μπορεί να περιέχει λίστα entity tags και το `If-None-Match` wildcard. Μην κάνεις απλή string σύγκριση αν υποστηρίζεις πλήρη HTTP semantics. Αν το δικό σου API ορίζει πιο στενό grammar, τεκμηρίωσέ το και απέρριψε τα υπόλοιπα καθαρά.

Cookies δεν είναι απλό parameter

Cookie authentication φέρνει browser concerns όπως `Secure`, `HttpOnly`, `SameSite`, CSRF και domain scope. Το ότι το FastAPI μπορεί να διαβάσει cookie δεν σημαίνει ότι η πολιτική είναι ασφαλής. Διάβασε την τιμή στο boundary, επαλήθευσε signature ή session lookup και μετέτρεψέ τη σε principal. Μην περνάς raw cookie βαθιά στο domain.

SOURCE TRACE

Οι declarations `Path`, `Query`, `Header` και `Cookie` είναι factories στο `fastapi/param_functions.py` και δημιουργούν metadata από `fastapi/params.py`. Το `get_dependant` στο `fastapi/dependencies/utils.py` αναλύει τη signature. Το `solve_dependencies` συλλέγει path, query, header και cookie values πριν καλέσει το endpoint.

ΠΑΓΙΔΑ

Το validation ενός `limit <= 100` προστατεύει μόνο αυτό το field. Δεν προστατεύει query που κάνει expensive join, regex search ή N+1 provider calls. Κόστος και authorization χρειάζονται ξεχωριστή πολιτική.

ΜΗΧΑΝΙΣΜΟΣ

Η θέση του input είναι μέρος του νοήματος. Μετέτρεψε untrusted strings σε bounded typed values στο HTTP boundary και πέρασε στο use case μόνο ό,τι χρειάζεται.

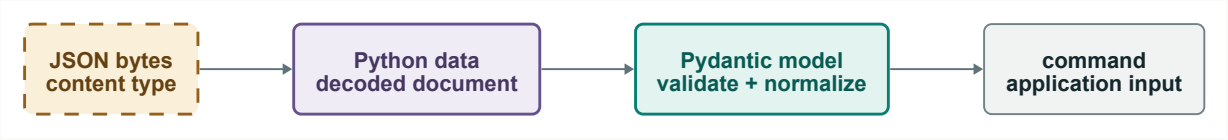
PROBE

Στείλε duplicate query keys, κενή τιμή, invalid UUID, limit 0 και limit 101. Κατέγραψε το error `loc` για κάθε περίπτωση. Δοκίμασε header με διαφορετικό case και cookie absence. Μετά γράψε property test για cursor decoder με τυχαία και υπερβολικά μεγάλα strings.

ΚΕΦΑΛΑΙΟ 5

Request bodies και Pydantic models

To request model είναι anti-corruption layer. Μετατρέπει ένα εξωτερικό JSON document σε γνωστή δομή πριν η εφαρμογή πάρει business απόφαση.



Σχήμα 5.1 · Από JSON bytes σε validated command.

Ένα JSON body δεν είναι Python object μέχρι να αποκωδικοποιηθεί. Μετά την αποκωδικοποίηση παραμένει untrusted data: fields λείπουν, έχουν λάθος type, είναι υπερβολικά μεγάλα ή συνδυάζονται με τρόπο που δεν επιτρέπεται. Το Pydantic model περιγράφει τη shape και εκτελεί validation. Το domain model μπορεί να είναι διαφορετικό object με αυστηρότερα invariants.

PYTHON

```
from decimal import Decimal
from typing import Self

from fastapi import FastAPI
from pydantic import BaseModel, ConfigDict, Field, field_validator

app = FastAPI()


class LineInput(BaseModel):
    model_config = ConfigDict(extra="forbid")

    sku: str = Field(min_length=1, max_length=40)
    quantity: int = Field(ge=1, le=100)
    unit_price: Decimal = Field(gt=0, max_digits=10, decimal_places=2)

    @field_validator("sku")
    @classmethod
    def normalize_sku(cls, value: str) -> str:
        normalized = value.strip().upper()
        if not normalized:
            raise ValueError("sku cannot be blank")
        return normalized


class OrderCreate(BaseModel):
    model_config = ConfigDict(extra="forbid")

    customer_reference: str = Field(min_length=1, max_length=80)
    lines: list[LineInput] = Field(min_length=1, max_length=50)


@app.post("/orders")
async def create_order(data: OrderCreate) -> dict[str, int]:
    return {"accepted_lines": len(data.lines)}
```

Το `extra="forbid"` απορρίπτει άγνωστα fields. Αυτό είναι καλό για command API όπου typo δεν πρέπει να αγνοηθεί σιωπηλά. Σε ingestion boundary με versioned events ίσως επιλέξεις forward-compatible αγνόηση. Η επιλογή είναι protocol decision και πρέπει να δοκιμάζεται.

Το Pydantic κάνει coercion σε αρκετές περιπτώσεις. Αυτό είναι βολικό για query strings, αλλά μπορεί να κρύψει αμφίβολα bodies. Το strict mode μπορεί να ρυθμιστεί ανά field ή model. Μη γυρίζεις κάθε model σε strict από συνήθεια. Αποφάσισε τι αποδέχεται το wire format και γράψε tests για ακριβώς αυτές τις μετατροπές.

Optional, nullable και missing

`str | None` σημαίνει ότι η τιμή μπορεί να είναι `null`, όχι αυτομάτως ότι το field μπορεί να λείπει. Default `None` το κάνει μη απαιτούμενο. Σε PATCH API χρειάζεται συχνά να ξεχωρίσεις missing από explicit null: το πρώτο σημαίνει «μην αλλάξεις», το δεύτερο «καθάρισε την τιμή». Το `model_fields_set` δείχνει ποια fields δόθηκαν και το `model_dump(exclude_unset=True)` παράγει μόνο αυτά.

Μην εφαρμόζεις τυφλά το dump πάνω σε ORM row. Πρώτα όρισε allowed patch fields, μετά έλεγξε domain rules και τέλος κάνε explicit update. Mass assignment από public model σε persistence object μπορεί να ανοίξει authorization bug όταν προστεθεί νέο field.

Validators στο σωστό επίπεδο

Field validator είναι σωστός για normalization και κανόνα ενός field. Model validator ταιριάζει σε σχέση μεταξύ fields, όπως `starts_at < ends_at`. Business invariant που απαιτεί database ή current user δεν ανήκει στο Pydantic model. Το validation πρέπει να είναι deterministic και χωρίς network I/O.

Χρησιμοποίησε `Decimal` για χρηματικές τιμές όταν το protocol δέχεται decimal number ή string με σαφή convention. Το binary float δεν αναπαριστά ακριβώς πολλά δεκαδικά. Παρ' όλα αυτά, ποσό και currency πρέπει να γίνουν domain value object. Το `gt=0` μόνο του δεν ελέγχει currency scale, pricing authority ή αν ο client επιτρέπεται να στείλει τιμή.

Content type και μέγεθος

Στην έκδοση αναφοράς το strict content-type checking είναι ενεργό από default για routes με body. JSON πρέπει να δηλώνεται με συμβατό `application/json` ή `application/*+json` media type. Αυτό εμποδίζει ambiguous parsing. Το body size limit όμως συνήθως πρέπει να επιβληθεί στην edge ή ASGI server layer πριν φορτωθούν τεράστια bytes στη μνήμη.

Nested models κάνουν το OpenAPI ακριβές, αλλά βάθος και πλήθος στοιχείων είναι επίσης attack surface. Βάλε bounds σε lists και strings. Για πραγματικά μεγάλα datasets χρησιμοποίησε streaming upload ή asynchronous ingestion protocol, όχι JSON document εκατοντάδων megabytes.

Από DTO σε command

Το request model ανήκει στο API version. Μετέτρεψέ το σε application command με values που το domain καταλαβαίνει. Αυτό εμποδίζει Pydantic configuration, aliases και optional transport fields να διαχυθούν παντού. Επίσης επιτρέπει σε job ή CLI να καλέσει το ίδιο use case χωρίς να κατασκευάζει fake HTTP model.

SOURCE TRACE

Το body parsing γίνεται μέσα στο `get_request_handler` του `fastapi/routing.py`. Εκεί ελέγχεται το content type και παράγεται decoded body. Το `request_body_to_args` στο `fastapi/dependencies/utils.py` αντιστοιχίζει το body στα model fields. Η Pydantic v2 compatibility βρίσκεται κυρίως κάτω από `fastapi/_compat/v2.py`.

ΠΑΓΙΔΑ

Το Pydantic validation αποδεικνύει shape και τοπικούς κανόνες. Δεν αποδεικνύει ότι το SKU υπάρχει, ότι η τιμή είναι έγκυρη ή ότι ο caller έχει δικαίωμα να δημιουργήσει την παραγγελία.

ΜΗΧΑΝΙΣΜΟΣ

Request DTO, application command και domain object μπορούν να έχουν παρόμοια fields αλλά διαφορετικό ownership. Η μετατροπή μεταξύ τους είναι χρήσιμο boundary, όχι άχρηστο duplication.

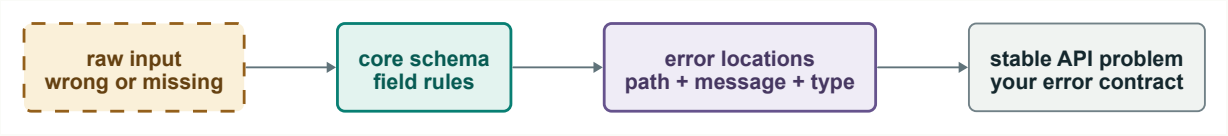
PROBE

Στείλε unknown field, κενό SKU, 51 lines, decimal με τρία ψηφία και JSON με λάθος `Content-Type`. Για PATCH model δοκίμασε `{}`, `{"name": null}` και `{"name": "Ada"}`. Κατέγραψε `model_fields_set` και το ακριβές HTTP error ώστε να δεις τη διαφορά missing και null.

ΚΕΦΑΛΑΙΟ 6

Validation, strictness και error contract

To validation error είναι public API response. Η εσωτερική λίστα λαθών του Pydantic είναι πολύτιμη, αλλά δεν χρειάζεται να γίνει αυτούσια μόνιμο contract.



Σχήμα 6.1 · Το internal validation result μεταφράζεται σε stable problem document.

Όταν path, query, header, cookie ή body δεν μπορεί να επικυρωθεί, το FastAPI συγκεντρώνει errors και σηκώνει `RequestValidationError`. Ο default handler επιστρέφει 422 με `detail`. Κάθε item περιέχει location, error type και message. Αυτό είναι πολύ καλό για ανάπτυξη. Σε public API όμως πρέπει να αποφασίσεις αν θέλεις να δεσμευτείς ακριβώς σε αυτή τη shape και στα messages της εκάστοτε Pydantic έκδοσης.

Ένα application-wide handler μπορεί να κρατήσει σταθερό envelope και να μετατρέψει τα internals σε δικούς σου codes. Μη χάνεις το location. Είναι ο μόνος τρόπος να ξεχωρίσει ο client το `body.lines.2.quantity` από query field με ίδιο όνομα.

PYTHON

```
from typing import Any

from fastapi import FastAPI, Request
from fastapi.exceptions import RequestValidationError
from fastapi.responses import JSONResponse

app = FastAPI()


def public_issue(error: dict[str, Any]) -> dict[str, Any]:
    location = [str(part) for part in error.get("loc", ())]
    return {
        "path": location,
        "code": str(error.get("type", "invalid")),
        "message": str(error.get("msg", "Invalid value")),
    }


@app.exception_handler(RequestValidationError)
async def validation_handler(
    request: Request,
    exc: RequestValidationError,
) -> JSONResponse:
    return JSONResponse(
        status_code=422,
        content={
            "type": "https://api.example.test/problems/invalid-request",
            "title": "Invalid request",
```

```
            "status": 422,
            "issues": [public_issue(error) for error in exc.errors()],
        },
    )
```

Το παράδειγμα μοιάζει με problem details, αλλά το κρίσιμο είναι η δική σου σταθερή σημασιολογία. Το `type` είναι identifier της κατηγορίας. Το `status` συμφωνεί με το HTTP status. Τα field issues είναι structured. Αν προσθέσεις request id, κράτησέ το ως correlation value και όχι σαν υποκατάστατο του machine-readable error code.

Request error και domain rejection

Malformed UUID, missing field και λάθος integer είναι validation failures. «Το κουπόνι έληξε», «δεν υπάρχει διαθέσιμο απόθεμα» και «η έκδοση είναι stale» είναι domain outcomes. Μην τα βάζεις όλα σε 422 επειδή ο client πρέπει να αλλάξει κάτι. Χρειάζονται διαφορετικούς codes, status και retry semantics.

Το ίδιο ισχύει για authentication. Missing ή invalid credentials είναι 401 με κατάλληλο `WWW-Authenticate`. Authenticated principal χωρίς permission είναι συνήθως 403. Resource που κρύβεις για λόγους privacy μπορεί να χαρτογραφεί σε 404. Αυτές είναι policy αποφάσεις, όχι Pydantic rules.

Coercion ως protocol decision

Το URL query είναι text, άρα η μετατροπή `"20"` σε integer είναι φυσική. Σε JSON body, η αποδοχή `"20"` ως number μπορεί να κρύψει bug άλλου client. Το Pydantic υποστηρίζει strict types και model configuration. Εφάρμοσέ τα εκεί όπου το wire contract το απαιτεί και γράψε compatibility tests πριν αλλάξεις υπάρχον endpoint.

Normalization και validation είναι διαφορετικές πράξεις. Το trim ενός human-entered display name ίσως είναι χρήσιμο. Το trim ενός signed token ή idempotency key αλλάζει identity και μπορεί να είναι λάθος. Μη μετασχηματίζεις security-sensitive input για να το κάνεις να «περάσει».

Μην επιστρέφεις όλο το input

Validation exceptions μπορούν να κρατούν body ή values για debugging. Εκεί μπορεί να υπάρχουν passwords, access tokens, προσωπικά δεδομένα ή μεγάλα documents. Ο public handler δεν πρέπει να κάνει echo raw input. Τα logs επίσης χρειάζονται allowlist και redaction. Κατέγραψε category, safe field path, route template και request id. Όχι authorization header ή ολόκληρο body.

Messages για humans μπορεί να αλλάξουν ή να μεταφραστούν. Clients πρέπει να βασίζονται σε stable code και path. Αν αλλάξεις error taxonomy, αντιμετώπισέ το σαν API migration. Snapshot του OpenAPI δεν αρκεί επειδή πολλά error shapes δηλώνονται generic. Χρειάζονται πραγματικά response contract tests.

Όρια κόστους

Validation συμβαίνει αφού το request έχει φτάσει αρκετά βαθιά. Edge limits για body bytes, header count και request line length κόβουν abusive input νωρίτερα. Model limits σε list length, string length και nesting προστατεύουν το application layer. Database statement timeout και bounded external calls

προστατεύουν το υπόλοιπο workflow. Καμία μοναδική validation ρύθμιση δεν καλύπτει όλο το cost surface.

SOURCE TRACE

Το `RequestValidationError` ορίζεται στο `fastapi/exceptions.py`. Ο default `request_validation_exception_handler` στο `fastapi/exception_handlers.py` χρησιμοποιεί `jsonable_encoder(exc.errors())`. Στο `get_request_handler` του `fastapi/routing.py`, τα errors από body και dependencies συγκεντρώνονται πριν σηκωθεί η exception.

ΠΑΓΙΔΑ

Αν οι clients κάνουν branch πάνω στο αγγλικό `msg`, ένα μικρό Pydantic upgrade μπορεί να γίνει breaking change. Δώσε δικό σου stable code.

ΜΗΧΑΝΙΣΜΟΣ

Validation errors περιγράφουν invalid transport input. Domain outcomes περιγράφουν έγκυρο command που δεν μπορεί να εκτελεστεί. Κράτα διαφορετικό taxonomy και διαφορετικά tests.

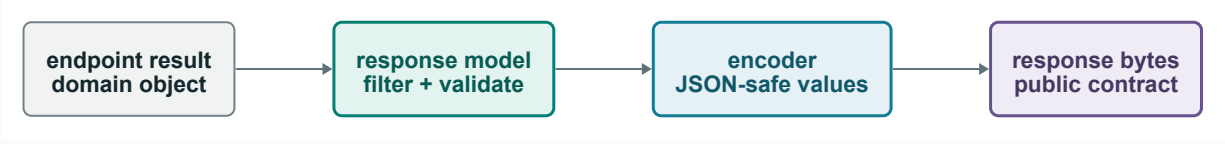
PROBE

Πάρε ένα nested body και δημιούργησε πέντε failures σε διαφορετικά επίπεδα. Snapshot μόνο τα stable fields του custom envelope. Μετά αναβάθμισε προσωρινά το error message στο test fixture και βεβαιώσου ότι ο client simulator κάνει branch σε `code` και `path`, όχι σε human text.

ΚΕΦΑΛΑΙΟ 7

Response models και serialization boundary

To response model δεν είναι documentation decoration. Είναι allowlist του public output και runtime έλεγχος ότι ο server τηρεί το contract του.



Σχήμα 7.1 · Το internal result φιλτράρεται και γίνεται public JSON.

Input και output models σπάνια πρέπει να είναι ίδια. Ένα persistence object μπορεί να περιέχει password hash, internal notes, provider ids και flags που δεν ανήκουν στον client. Αν επιστρέψεις τυχαίο `dict`, κάθε μελλοντικό field μπορεί να διαρρεύσει. Το `response_model` δηλώνει ακριβώς τι επιτρέπεται να περάσει το boundary.

PYTHON

```
from datetime import datetime, timezone
from uuid import UUID

from fastapi import FastAPI
from pydantic import BaseModel, ConfigDict

app = FastAPI()


class UserRecord:
    def __init__(self, user_id: UUID, email: str, password_hash: str) -> None:
        self.id = user_id
        self.email = email
        self.password_hash = password_hash
        self.created_at = datetime.now(timezone.utc)


class UserPublic(BaseModel):
    model_config = ConfigDict(from_attributes=True)

    id: UUID
    email: str
    created_at: datetime


@app.get("/users/{user_id}", response_model=UserPublic)
async def read_user(user_id: UUID) -> UserRecord:
    return UserRecord(user_id, "ada@example.test", "never-expose-this")
```

Το `from_attributes=True` επιτρέπει ανάγνωση attributes αντί μόνο mappings. Δεν σημαίνει ότι πρέπει να επιστρέφεις live ORM entities παντού. Lazy-loaded relationship κατά το serialization μπορεί να κάνει κρυφό query, να συναντήσει κλειστή session ή να παράγει N+1. Προτίμησε explicit projection που φορτώνει ό,τι χρειάζεται το response μέσα στο transaction boundary.

Τι συμβαίνει στο return

Για declarative response, το FastAPI επικυρώνει το result απέναντι στο response field και το serializes σύμφωνα με include, exclude, aliases και unset policy. Αν δεν υπάρχει response field, χρησιμοποιεί `jsonable_encoder` ώστε values όπως datetime, UUID και Pydantic models να γίνουν JSON-compatible Python data. Η response class, συνήθως `JSONResponse`, μετατρέπει αυτά τα values σε bytes.

Response validation failure είναι bug του server. Ο client έστειλε έγκυρο request και δεν φταίει επειδή η εφαρμογή ξέχασε field. Μην επιστρέφεις την εσωτερική λεπτομέρεια ως 422. Κατέγραψε ασφαλή diagnostics με endpoint context, επέστρεψε generic 500 problem και διόρθωσε το producer.

Include, exclude και ξεχωριστά models

Τα options `response_model_exclude_none`, `exclude_unset` και include/exclude είναι χρήσιμα, αλλά δεν πρέπει να γίνουν μόνιμη αρχιτεκτονική για δέκα public views ενός τεράστιου model. Ξεχωριστά output models κάνουν το contract ορατό, δίνουν σωστό OpenAPI και αποτρέπουν accidental exposure όταν προστεθεί field.

Aliases είναι επίσης public contract. Με `response_model_by_alias=True`, που είναι το default, το output χρησιμοποιεί alias. Αν αλλάξεις alias ή casing, κάνεις schema migration. Μην αφήνεις serializer preference να αλλάξει χωρίς contract diff.

Pydantic field serializers είναι κατάλληλοι για deterministic representation, όπως canonical timestamp. Δεν είναι χώρος για database lookup ή provider call. Serialization μπορεί να συμβεί αφού έχει τελειώσει μέρος του dependency lifecycle και πρέπει να παραμένει bounded.

Direct Response σημαίνει ανάληψη ελέγχου

Όταν επιστρέφεις `Response` subclass, το FastAPI δεν μετατρέπει το content με το response model. Αυτό χρειάζεται για redirect, file, stream, custom media type ή ήδη encoded bytes. Τότε εσύ κατέχεις το πραγματικό content type, status, headers και encoding. Η OpenAPI δήλωση μπορεί να παραμένει χωριστή, αλλά δεν επιβάλλει runtime validation.

Για 204 επέστρεψε empty `Response(status_code=204)`. Μην αφήνεις JSON `null` σε status που δεν επιτρέπεται να έχει body. Για downloads όρισε safe filename και `Content-Disposition`. Για streams θυμήσου ότι failure μετά το response start δεν μπορεί να μετατραπεί καθαρά σε νέο JSON error.

Σταθερή αναπαράσταση

JSON objects δεν έχουν business ordering. Arrays έχουν. Timestamps χρειάζονται timezone convention. Decimal και μεγάλοι integers χρειάζονται συμφωνία με clients που ίσως χρησιμοποιούν JavaScript numbers. Enum values πρέπει να είναι σταθερά wire values και όχι τυχαία display labels.

Αν response γίνεται cache key, signature ή webhook payload, canonicalization πρέπει να είναι explicit. Το συνηθισμένο API response δεν χρειάζεται canonical JSON, αλλά χρειάζεται semantic stability. Δοκίμασε values και schema, όχι whitespace του encoder εκτός αν τα bytes είναι όντως contract.

SOURCE TRACE

Η `serialize_response` στο `fastapi/routing.py` επικυρώνει και serializes μέσω του response field ή καλεί `jsonable_encoder`. Η ίδια διαδρομή σηκώνει `ResponseValidationError` με endpoint context. Το `fastapi/encoders.py` περιέχει το generic `jsonable_encoder`.

ΠΑΓΙΔΑ

Το output filtering είναι defense, όχι άδεια να φορτώνεις secrets στο ίδιο object χωρίς λόγο. Ένα direct `JSONResponse` ή άλλο endpoint μπορεί αργότερα να παρακάμψει το φίλτρο.

ΜΗΧΑΝΙΣΜΟΣ

Σχεδίασε output model από την οπτική του consumer. Φόρτωσε explicit projection, επικύρωσέ το και μετά serialize. Το persistence shape δεν είναι API shape.

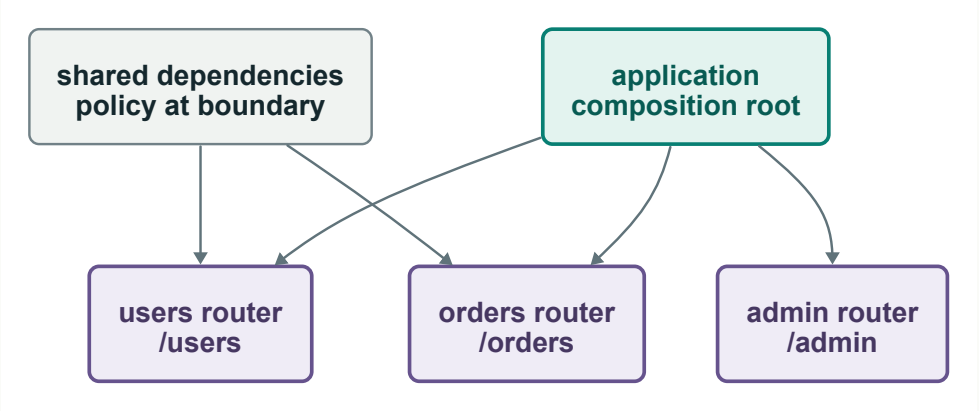
PROBE

Επέστρεψε object με ένα extra secret field και επιβεβαίωσε ότι λείπει. Μετά αφαίρεσε required public field και assert 500, χωρίς λεπτομέρειες στον client. Πρόσθεσε query-count assertion σε response με relationships ώστε να εντοπίσεις lazy serialization και N+1.

ΚΕΦΑΛΑΙΟ 8

Routers και modular composition

Ο router οργανώνει HTTP contracts. Δεν είναι application layer και δεν πρέπει να γίνει κρυφό service container.



Σχήμα 8.1 · Το app συνθέτει routers και boundary policies.

Ένα `APIRouter` συγκεντρώνει routes με κοινό prefix, tags, dependencies και responses. Το module μπορεί να δηλώσει HTTP behavior χωρίς να γνωρίζει το τελικό `FastAPI` object. Το composition root αποφασίζει ποιοι routers μπαίνουν, με ποια version και κάτω από ποια policy.

PYTHON

```
from typing import Annotated
from uuid import UUID

from fastapi import APIRouter, Depends, FastAPI
from pydantic import BaseModel


class UserView(BaseModel):
    id: UUID
    name: str


def require_api_key() -> None:
    pass


users = APIRouter(prefix="/users", tags=["users"])


@users.get("/{user_id}", response_model=UserView)
async def show_user(user_id: UUID) -> UserView:
    return UserView(id=user_id, name="Ada")


def create_app() -> FastAPI:
    app = FastAPI(title="Accounts API")
    app.include_router(
```

```
        users,
        prefix="/v1",
        dependencies=[Depends(require_api_key)],
    )
    return app
```

Η parameterless dependency εκτελεί policy χωρίς να περάσει value στο endpoint. Είναι χρήσιμη για κοινό authentication gate ή audit setup. Αν όμως χρειάζεται ο principal μέσα στο route, δήλωσέ τον ως typed argument αντί να κρύψεις την τιμή σε global context.

Δομή ανά capability

Folders μόνο ανά τεχνικό είδος, όπως `routers/`, `services/`, `schemas/` και `models/`, διασκορπίζουν μία αλλαγή σε όλο το repository. Για μεγαλύτερο σύστημα προτίμησε capability slice: `orders/api.py`, `orders/contracts.py`, `orders/use_cases.py`, `orders/adapters.py`. Shared primitives μένουν μικρά και πραγματικά shared.

Το router module μπορεί να εισάγει use case interface ή factory dependency. Δεν πρέπει να κάνει import το κεντρικό `app`, επειδή δημιουργεί circular composition και δυσκολεύει τα tests. Η ροή dependencies πηγαίνει προς τα μέσα: transport γνωρίζει application contracts, το domain δεν γνωρίζει FastAPI.

Prefixes, tags και versioning

Το prefix είναι μέρος του deployed URI. Μην φτιάχνεις versioning απλώς αντιγράφοντας όλο τον router και αφήνοντας δύο implementations να αποκλίνουν. Κράτα κοινό use case όταν το business behavior είναι ίδιο και ξεχωριστούς HTTP adapters ή DTOs όπου αλλάζει το contract.

Τα tags οργανώνουν documentation, όχι authorization. Το ότι route βρίσκεται σε `admin` router δεν αποδεικνύει ότι είναι protected. Βάλε dependency policy στο σωστό include boundary και γράψε test που επιθεωρεί κάθε protected route.

Router-level default responses μπορούν να τεκμηριώνουν κοινά errors. Πρόσεξε να μη δηλώνεις 401 ή 403 σε routes που πραγματικά επιστρέφουν άλλο envelope. OpenAPI metadata χωρίς matching handler δημιουργεί ψεύτικη βεβαιότητα.

Composition και route identity

Η ένταξη router συνθέτει metadata από parent και child. Route name και operation id πρέπει να είναι σταθερά αν παράγεις SDKs. Δύο functions με ίδιο όνομα σε διαφορετικά modules μπορεί να δημιουργήσουν ασταθή ή συγκρουόμενα operation ids ανάλογα με generator. Όρισε deterministic convention και κάνε snapshot το OpenAPI artifact.

Μπορείς να περιλάβεις router περισσότερες από μία φορές, για παράδειγμα με άλλο prefix. Αυτό δημιουργεί ξεχωριστές exposed operations, δεν κάνει dynamic mount ανά request. Κάθε include πρέπει να είναι συνειδητό. Έλεγξε duplicate method-path pairs κατά το build.

Μην κάνεις το router business object

Ένα route πρέπει να κάνει parse, να πάρει dependencies, να καλέσει use case και να παρουσιάσει outcome. Αν οι routes μοιράζονται helpers που κάνουν database writes, retries και HTTP calls, το

shared module είναι κρυφό application layer. Δώσε του όνομα, interface και owner. Οι decorators δεν είναι architecture.

Στην άλλη πλευρά, μη δημιουργήσεις class για κάθε function χωρίς λόγο. Plain functions και μικρά immutable commands είναι αρκετά μέχρι να υπάρξει state ή πολλαπλή implementation. Το ζητούμενο είναι visible boundary, όχι pattern count.

SOURCE TRACE

Η `FastAPI.include_router` στο `fastapi/applications.py` αναθέτει στον κεντρικό router. Το `APIRouter.include_router` και τα include contexts βρίσκονται στο `fastapi/routing.py`. Στο 0.141.1 η υλοποίηση συνθέτει route metadata και lifespan contexts διατηρώντας τη router hierarchy, αλλά αυτά τα internal objects δεν είναι public extension API.

ΠΑΓΙΔΑ

Ένα global dependency σε μεγάλο router μπορεί να εκτελεί ακριβό I/O και σε health ή public endpoint που συμπεριλήφθηκε κατά λάθος. Το scope της policy πρέπει να είναι ορατό και ελεγμένο.

ΜΗΧΑΝΙΣΜΟΣ

Router είναι composition unit για HTTP operations. Το application behavior μένει σε use cases που μπορούν να κληθούν χωρίς ASGI request.

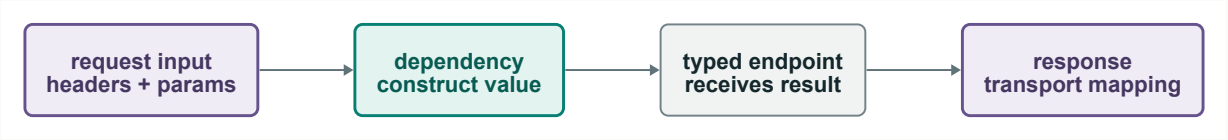
PROBE

Παρήγαγε λίστα `(method, path, name, operation_id)` από `app.routes`. Assert ότι δεν υπάρχουν duplicates, ότι όλα τα `/admin` έχουν το αναμενόμενο dependency και ότι το `/health` δεν το έχει. Κάνε include τον ίδιο router σε δεύτερο prefix και δες ακριβώς τι αλλάζει στο OpenAPI.

ΚΕΦΑΛΑΙΟ 9

Dependency injection χωρίς magic

Το dependency system λύνει ένα graph από callables για κάθε request. Δεν είναι global container και δεν αποφασίζει μόνο του το lifecycle των υποδομών σου.



Σχήμα 9.1 · Ένα dependency παράγει typed value για το endpoint.

Οποιοδήποτε callable με inspectable signature μπορεί να γίνει dependency. Το FastAPI αναλύει τα arguments του όπως αναλύει endpoint: μπορούν να είναι request parameters ή άλλα dependencies. Το return value περνά στη function που το δήλωσε. Με `Annotated` μπορείς να δώσεις επαναχρησιμοποιήσιμο typed alias.

PYTHON

```
from dataclasses import dataclass
from typing import Annotated

from fastapi import Depends, FastAPI, Header, HTTPException

app = FastAPI()


@dataclass(frozen=True)
class Principal:
    user_id: str


async def current_principal(
    authorization: Annotated[str | None, Header()] = None,
) -> Principal:
    if authorization != "Bearer demo-token":
        raise HTTPException(
            status_code=401,
            detail="invalid_credentials",
            headers={"WWW-Authenticate": "Bearer"},
        )
    return Principal(user_id="user-123")


CurrentPrincipal = Annotated[Principal, Depends(current_principal)]


@app.get("/me")
async def me(principal: CurrentPrincipal) -> dict[str, str]:
    return {"id": principal.user_id}
```

Το demo token δεν είναι production authentication. Το παράδειγμα δείχνει τη ροή: header, verifier, principal, endpoint. Η υπόλοιπη εφαρμογή δεν χρειάζεται raw Authorization string. Ένα αληθινό

verifier θα ελέγξει scheme, signature, issuer, audience, expiry και revocation policy με bounded κόστος.

Dependency ή απλή function;

Χρησιμοποίησε dependency όταν η τιμή ανήκει στο request boundary ή χρειάζεται FastAPI composition: principal, database session, locale, request-scoped service ή parsed precondition. Μια pure function που υπολογίζει tax από amount δεν κερδίζει κάτι αν γίνει `Depends`. Κάλεσέ τη κανονικά από το use case.

Η υπερβολική χρήση dependencies κρύβει call flow. Αν κάθε μικρή business function είναι node του graph, ο αναγνώστης πρέπει να προσομοιώσει το framework για να καταλάβει το workflow. Κράτα το graph κοντά στο transport και resource composition. Μετά κάνε explicit Python calls.

Sub-dependencies

Ένα `current_principal` μπορεί να εξαρτάται από settings, token decoder και user repository. Το endpoint βλέπει μόνο principal, αλλά ο graph διατηρεί όλα τα requirements. Το FastAPI λύνει πρώτα τα child nodes. Αν υπάρξουν validation errors, το endpoint δεν καλείται.

Dependencies μπορούν να είναι sync ή async. Sync callable εκτελείται στο thread pool. Async callable εκτελείται στο event loop. Η ίδια προειδοποίηση με τα endpoints ισχύει: blocking library μέσα σε async dependency μπλοκάρει το loop. Το framework δεν εξετάζει το σώμα της function για να το διορθώσει.

Cache ανά request

Από default, η ίδια dependency με το ίδιο effective cache key υπολογίζεται μία φορά ανά request και το value επαναχρησιμοποιείται. Αυτό αποφεύγει δύο session objects ή δύο token verifications όταν διαφορετικοί parents ζητούν το ίδιο node. Δεν είναι cache ανά process και εξαφανίζεται στο τέλος του request.

Το `Depends(..., use_cache=False)` ζητά νέα εκτέλεση. Χρειάζεται σπάνια, για παράδειγμα σε value που πρέπει να διαβαστεί εκ νέου μέσα στο ίδιο request. Αν το χρησιμοποιήσεις σε resource dependency, μπορεί να ανοίξεις πολλαπλά resources και να αλλάξεις cleanup order. Κάνε τη διαφορά explicit σε test.

Policy και data

Dependency που επιστρέφει principal λύνει authentication. Δεν αρκεί για authorization πάνω σε συγκεκριμένο order. Το use case πρέπει να ελέγξει σχέση principal, action και resource μέσα στο ίδιο consistent read. Ένα generic `require_admin` είναι boundary policy. Ένα `can_edit_order` εξαρτάται από τρέχον resource state και συνήθως ανήκει πιο κοντά στο application workflow.

Επίσης, dependency δεν είναι εγγύηση ότι εκτελέστηκε σε κάθε route που έπρεπε. Router composition test πρέπει να ελέγχει policy coverage. Security που βασίζεται σε developer memory δεν είναι security boundary.

SOURCE TRACE

Το `Depends` στο `fastapi/param_functions.py` δημιουργεί `fastapi.params.Depends`. Το `get_dependant` στο `fastapi/dependencies/utils.py` κατασκευάζει το graph από signatures. Το `solve_dependencies` το διασχίζει recursively, συγκεντρώνει errors και περνά το dictionary των values στο endpoint.

ΠΑΓΙΔΑ

Dependency override αντιστοιχεί στο αρχικό callable object. Αν τυλίγεις dependencies δυναμικά ή τα δημιουργείς μέσα σε factory σε κάθε import, τα tests μπορεί να κάνουν override άλλο identity από αυτό που έχει το route.

ΜΗΧΑΝΙΣΜΟΣ

Το dependency graph κατασκευάζει request values και resources. Μετά το HTTP boundary, κράτα το application workflow ως κανονικό, ευανάγνωστο Python call graph.

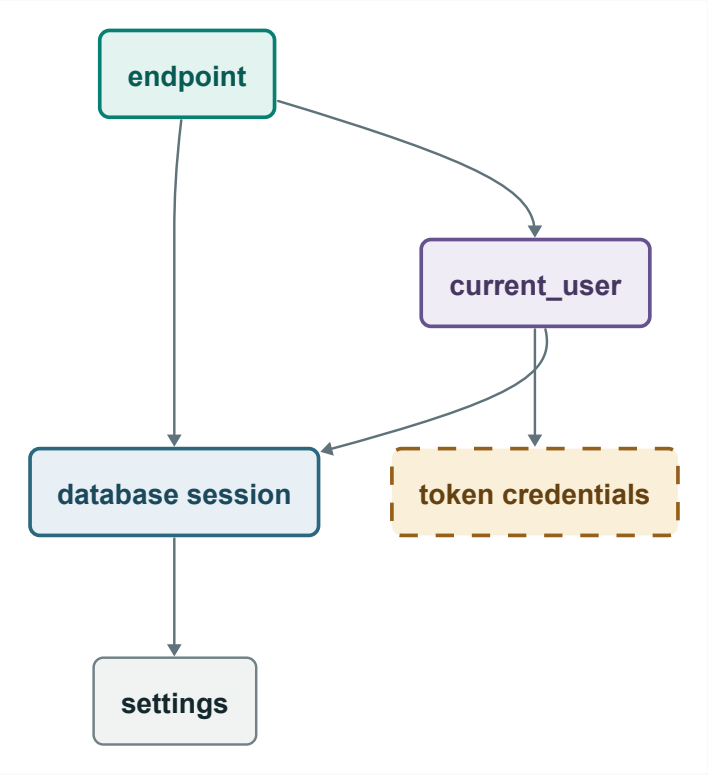
PROBE

Δήλωσε dependency counter που ζητείται απευθείας από endpoint και μέσω δύο sub-dependencies. Επιβεβαίωσε μία εκτέλεση. Πρόσθεσε `use_cache=False` σε ένα edge και κατέγραψε τη νέα σειρά. Κάνε override το leaf σε test και assert ότι κανένα πραγματικό external call δεν έγινε.

ΚΕΦΑΛΑΙΟ 10

Dependency graph, cache και `yield`

Το `yield` δίνει resource και ορίζει cleanup. Το κρίσιμο δεν είναι μόνο αν θα κλείσει, αλλά αν θα κλείσει πριν ή μετά την αποστολή του response.



Σχήμα 10.1 · Το endpoint βρίσκεται στην κορυφή ενός resource και policy graph.

Generator dependency λειτουργεί σαν context manager. Ο κώδικας πριν από το `yield` αποκτά resource. Η yielded τιμή περνά στον consumer. Ο κώδικας στο `finally` απελευθερώνει resource ακόμη και όταν το endpoint αποτύχει. Αυτό ταιριάζει σε database session, lock handle ή scoped client operation.

PYTHON

```
from collections.abc import AsyncIterator
from typing import Annotated

from fastapi import Depends, FastAPI

app = FastAPI()
```

```
class Session:
    async def rollback(self) -> None:
        pass

    async def close(self) -> None:
        pass


async def open_session() -> Session:
    return Session()


async def database_session() -> AsyncIterator[Session]:
    session = await open_session()
    try:
        yield session
    except Exception:
        await session.rollback()
        raise
    finally:
        await session.close()


DbSession = Annotated[
    Session,
    Depends(database_session, scope="function"),
]


@app.get("/orders/{order_id}")
async def show_order(order_id: str, session: DbSession) -> dict[str, str]:
    return {"id": order_id}
```

Το bare `except` που καταπίνει exception μέσα σε yield dependency είναι ιδιαίτερα επικίνδυνο. Μπορεί να εμποδίσει τη σωστή propagation και να αφήσει το response path σε παράλογη κατάσταση. Αν κάνεις cleanup και δεν μπορείς να χειριστείς πραγματικά το failure, ξανασήκωσέ το.

Function scope και request scope

Στο 0.141.1, `Depends(scope="function")` κλείνει το dependency αφού τελειώσει η path operation function και πριν σταλεί το response. Το `Depends(scope="request")` κλείνει αφού σταλεί το response. Για yield dependency χωρίς explicit scope, το request scope είναι default.

Function scope είναι καλό όταν το resource χρειάζεται μόνο για να φτιαχτεί το result. Database transaction μπορεί να κάνει commit ή rollback και να κλείσει πριν ξεκινήσει αργό network send. Request scope χρειάζεται όταν το ίδιο το response χρησιμοποιεί resource αργότερα, όπως lazy stream. Τότε όμως κρατάς connection για όλη τη διάρκεια του client download, κάτι που μπορεί να εξαντλήσει pool.

Η συνηθέστερη καλύτερη λύση είναι να μην εξαρτάται το stream από request transaction. Φόρτωσε bounded metadata νωρίς ή άνοιξε resource μέσα στον generator με δικό του explicit lifecycle. Αν streamάρεις database rows, βάλε admission limit, timeout, cancellation cleanup και capacity budget.

Cleanup order

Ο graph έχει nested lifetimes. Parent dependency μπορεί να χρησιμοποιεί child στο cleanup, άρα το child πρέπει να παραμένει ζωντανό μέχρι να τελειώσει ο parent. Το FastAPI χρησιμοποιεί

`AsyncExitStack` και κλείνει σε αντίστροφη σειρά από την είσοδο. Δεν πρέπει να βασίζεσαι τυχαία στη σειρά declarations. Δήλωσε την πραγματική εξάρτηση ως sub-dependency.

Request-scoped dependency δεν μπορεί να εξαρτάται από function-scoped child που θα έκλεινε νωρίτερα ενώ ο parent το χρειάζεται στο δικό του late cleanup. Οι scope constraints εκφράζουν ακριβώς αυτή τη χρονική σχέση. Αν η σύνθεση δεν επιτρέπεται, το πρόβλημα είναι στο lifecycle design και όχι σε ενοχλητικό framework restriction.

Transaction στο `yield`;

Μπορείς να κάνεις automatic commit μετά το `yield`, αλλά τότε ο endpoint έχει ήδη επιστρέψει result και το commit μπορεί να αποτύχει αργότερα. Σε function scope θα αποτύχει πριν σταλεί το response, όμως το control flow παραμένει κρυφό. Για σημαντικά writes προτίμησε το use case να κάνει explicit commit και να επιστρέφει outcome μόνο μετά την επιτυχία. Το dependency κατέχει rollback σε exception και close, όχι business transaction policy.

Επίσης μην βάζεις external provider call στο cleanup. Το cleanup πρέπει να είναι bounded και αξιόπιστο. Durable work πηγαίνει σε transaction, outbox ή queue. Το τέλος ενός HTTP request δεν είναι job runner.

Cache key και mutable values

Η per-request cache επαναχρησιμοποιεί το ίδιο object. Αν δύο consumers τροποποιούν mutable dependency value, μοιράζονται state. Αυτό είναι επιθυμητό για μία session αλλά επικίνδυνο για generic dict. Επιστρέφε immutable principal και settings. Για mutable unit of work, δώσε έναν ξεκάθαρο owner.

Overrides ξαναχτίζουν dependent graph για το replacement callable. Το cleanup του fake πρέπει να μιμείται όσα lifecycle properties ελέγχει το test. Ένα override που επιστρέφει απλό object δεν αποδεικνύει ότι production pool κλείνει.

SOURCE TRACE
Στο `fastapi/dependencies/utils.py`, το `solve_dependencies` λύνει child nodes, χρησιμοποιεί dependency cache και περνά generator callables στο κατάλληλο `AsyncExitStack`. Το `request_response` στο `fastapi/routing.py` έχει χωριστό function stack πριν από την αποστολή και request stack γύρω από ολόκληρο το response cycle.

ΠΑΓΙΔΑ
Request-scoped database session πίσω από αργό stream μπορεί να δεσμεύσει μία pool connection για λεπτά. Δέκα αργοί clients μπορούν να εξαντλήσουν το pool χωρίς μεγάλο CPU load.

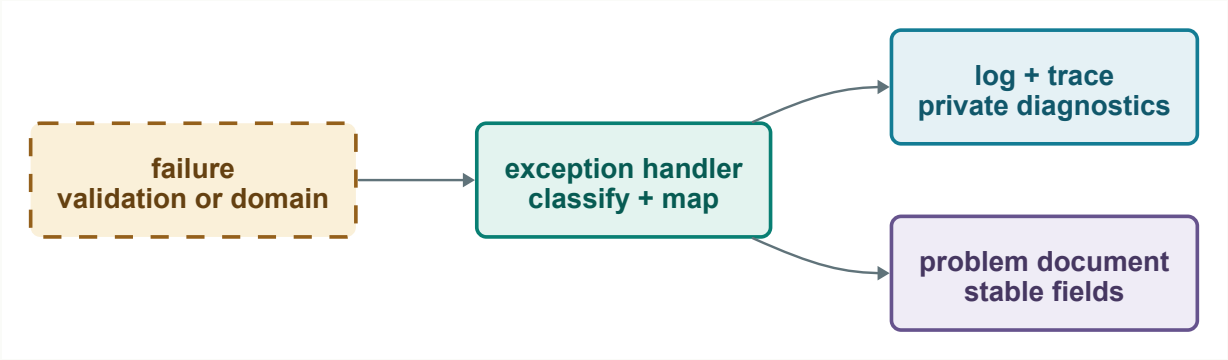
ΜΗΧΑΝΙΣΜΟΣ
Το scope είναι χρονικό contract. Διάλεξέ το από το τελευταίο σημείο όπου χρειάζεται το resource, όχι από συνήθεια. Κράτα commit policy explicit.

PROBE
Φτιάξε δύο nested yield dependencies που καταγράφουν enter και exit. Δοκίμασε normal response, endpoint exception και streaming response με μικρή καθυστέρηση ανά chunk. Σύγκρινε `scope="function"` και `scope="request"` και assert την ακριβή σειρά σε κάθε περίπτωση.

ΚΕΦΑΛΑΙΟ 11

Exceptions και σταθερό error contract

Η exception είναι control flow μέσα στον server. Το error response είναι public protocol. Ανάμεσά τους χρειάζεται ελεγχόμενη μετάφραση.



Σχήμα 11.1 · Κάθε failure ταξινομείται πριν γίνει public problem.

Το `HTTPException` σταματά το endpoint και μεταφέρει status, public detail και optional headers. Είναι σωστό εργαλείο στο HTTP boundary. Δεν είναι καλή exception για domain package που μπορεί να χρησιμοποιηθεί από worker ή CLI. Εκεί όρισε outcomes ή application exceptions χωρίς γνώση του HTTP.

PYTHON

```
from dataclasses import dataclass
from uuid import UUID

from fastapi import FastAPI, Request
from fastapi.responses import JSONResponse

app = FastAPI()


@dataclass(frozen=True)
class OrderConflict(Exception):
    order_id: UUID
    reason: str


@app.exception_handler(OrderConflict)
async def order_conflict_handler(
    request: Request,
    exc: OrderConflict,
) -> JSONResponse:
    request_id = getattr(request.state, "request_id", None)
    return JSONResponse(
        status_code=409,
        content={
            "type": "https://api.example.test/problems/order-conflict",
```

```
            "title": "Order conflict",
            "status": 409,
            "code": exc.reason,
            "request_id": request_id,
        },
    )
```

Το handler γνωρίζει το public taxonomy. Η exception κρατά μόνο ασφαλές reason code και identity που μπορεί να χρησιμοποιηθεί σε private log. Αν το id είναι ευαίσθητο, δεν χρειάζεται να επιστραφεί. Το request id βοηθά support και operator να συνδέσουν response, log και trace.

Μια μικρή error matrix

Γράψε πριν από τον κώδικα ποια outcomes υπάρχουν. Για `POST /orders` μπορεί να είναι 201 created, 409 duplicate business reference, 422 malformed input, 401 invalid credentials, 403 forbidden, 503 dependency unavailable και 500 unexpected bug. Για κάθε γραμμή όρισε retry semantics και αν η operation μπορεί να είχε ήδη εκτελεστεί.

Το 503 με `Retry-After` είναι χρήσιμο όταν η υπηρεσία ξέρει ότι προσωρινά δεν μπορεί να δεχτεί εργασία. Δεν είναι σωστή χαρτογράφηση για κάθε provider timeout. Αν το request είχε side effect και το outcome είναι άγνωστο, generic retry μπορεί να διπλασιάσει την πράξη. Χρειάζεται idempotency protocol ή status lookup.

Expected και unexpected failures

Expected rejection, όπως conflict, καταγράφεται συνήθως χωρίς stack trace και με metric counter. Unexpected exception καταγράφεται μία φορά στο boundary με stack trace, route template και correlation context. Μην την καταγράψεις σε repository, use case, middleware και server ξανά. Τα τέσσερα ίδια stack traces δεν δίνουν τέσσερις φορές περισσότερη πληροφορία.

Μη μετατρέπεις κάθε `Exception` σε 400. Κρύβεις server bugs ως client errors και οι success/error metrics ψεύδονται. Άφησε unknown failures να γίνουν 500 με generic body. Το πραγματικό diagnostic μένει σε ασφαλή logs και traces.

Cancellation και process shutdown είναι επίσης ιδιαίτερα control flows. Μην πιάνεις `BaseException`. Σε broad cleanup handler, κάνε μόνο bounded cleanup και ξανασήκωσε. Το σύστημα πρέπει να μπορεί να ακυρώσει request ή task χωρίς η εφαρμογή να συνεχίζει κρυφά expensive work.

Exception timing

Πριν σταλεί response start, handler μπορεί να αλλάξει status και body. Μετά το response start, όπως σε streaming response, δεν υπάρχει καθαρός τρόπος να αντικαταστήσεις το 200 με 500. Μπορείς να κλείσεις connection, να γράψεις protocol-specific error event ή να καταγράψεις failure. Γι’ αυτό η εργασία που μπορεί να αποτύχει πρέπει, όπου γίνεται, να ολοκληρώνεται πριν ξεκινήσει το stream.

Background task exception επίσης συμβαίνει αφού το response έχει φύγει. Δεν μπορεί να ενημερώσει τον client. Χρειάζεται monitoring και, αν το effect είναι σημαντικό, durable queue με retry και dead-letter policy.

Headers και content negotiation

Authentication failure χρειάζεται `WWW-Authenticate`. Rate limit μπορεί να χρειάζεται `Retry-After` και quota headers σύμφωνα με το δικό σου contract. Method mismatch και content-type failures έχουν δικά τους protocol semantics. Κράτα ένα κοινό problem media type μόνο αν clients το υποστηρίζουν. Το να αλλάξεις error content type είναι compatibility change.

Exception handlers είναι hierarchical ανά exception class. Όρισε specific handlers πριν σκεφτείς generic catch-all. Ένα catch-all δεν πρέπει να επιστρέφει exception string. Επίσης χρειάζεται να συνεργάζεται με middleware που προσθέτει CORS ή request id, αλλιώς τα errors διαφέρουν από τα success responses.

SOURCE TRACE

Τα FastAPI exceptions βρίσκονται στο `fastapi/exceptions.py` και οι default handlers στο `fastapi/exception_handlers.py`. Το request handler στο `fastapi/routing.py` αφήνει endpoint exceptions να κινηθούν προς το Starlette exception stack. Το `request_response` χρησιμοποιεί `wrap_app_handling_exceptions` γύρω από το ASGI app.

ΠΑΓΙΔΑ

Provider timeout δεν σημαίνει απαραίτητα ότι ο provider δεν έκανε το effect. Μην επιστρέφεις απλώς «retry» χωρίς idempotency key ή lookup protocol.

ΜΗΧΑΝΙΣΜΟΣ

Ταξινόμησε failures σε stable application outcomes. Ο HTTP adapter τα μετατρέπει σε status, headers και problem body. Τα unknown bugs παραμένουν 500.

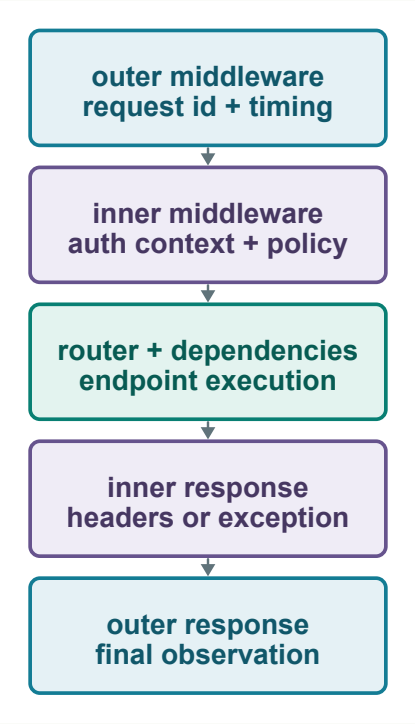
PROBE

Κάνε fault injection πριν το write, μετά το commit, πριν το response start και στο δεύτερο chunk ενός stream. Για κάθε σημείο κατέγραψε τι βλέπει ο client, τι γράφτηκε durable και ποιο signal βλέπει ο operator. Το διάγραμμα αυτό είναι η πραγματική error policy σου.

ΚΕΦΑΛΑΙΟ 12

Middleware και request context

Middleware τυλίγει ολόκληρο ASGI app. Είναι σωστό για transport-wide policy, όχι για business αποφάσεις που ανήκουν σε συγκεκριμένο resource.



Σχήμα 12.1 · Το request περνά προς τα μέσα και το response επιστρέφει προς τα έξω.

Middleware βλέπει request πριν το routing και response αφού επιστρέψει ο εσωτερικός handler. Ταιριάζει σε request ids, timing, CORS, trusted hosts, compression και ορισμένα security headers. Δεν ταιριάζει σε «ο user μπορεί να ακυρώσει αυτή την παραγγελία», επειδή αυτή η απόφαση χρειάζεται domain state.

PYTHON

```
from time import perf_counter
from uuid import uuid4

from fastapi import FastAPI, Request, Response

app = FastAPI()


@app.middleware("http")
```

```
async def request_context(
    request: Request,
    call_next,
) -> Response:
    incoming = request.headers.get("X-Request-ID")
    request_id = incoming if incoming and len(incoming) <= 80 else str(uuid4())
    request.state.request_id = request_id
    started = perf_counter()
    try:
        response = await call_next(request)
    finally:
        elapsed_ms = (perf_counter() - started) * 1000
        request.state.elapsed_ms = elapsed_ms
    response.headers["X-Request-ID"] = request_id
    return response
```

Σε πραγματικό σύστημα, μην εμπιστεύεσαι οποιοδήποτε incoming id. Βάλε grammar και length bound ή δημιούργησε δικό σου id και κράτα το upstream value ξεχωριστά. Αλλιώς attacker μπορεί να προκαλέσει log injection ή να ενώσει άσχετα traces.

Το παράδειγμα αποθηκεύει context στο `request.state`, άρα η τιμή ανήκει στο request object. Για libraries που δεν λαμβάνουν request, ένα `ContextVar` μπορεί να μεταφέρει correlation context μέσα στο ίδιο async task. Χρειάζεται πάντα reset token στο `finally`, επειδή worker threads και copied contexts έχουν διαφορετική συμπεριφορά.

Η σειρά είναι behavior

Αν middleware A τυλίγει B, τότε η request πλευρά τρέχει A και μετά B, ενώ η response πλευρά επιστρέφει B και μετά A. Ένα exception μπορεί να handled σε εσωτερικό ή εξωτερικό layer. CORS headers, access log status και security headers πρέπει να υπάρχουν και σε error responses. Μην απομνημονεύεις μόνο τη σειρά των `add_middleware` calls. Γράψε probe που καταγράφει πραγματική είσοδο και έξοδο μετά από κάθε framework upgrade.

Η εφαρμογή έχει επίσης built-in server error και exception handling layers. Όταν χρησιμοποιείς `app.add_middleware`, η θέση του user middleware καθορίζεται από το Starlette stack. Αν τυλίξεις ολόκληρο το FastAPI object με εξωτερικό ASGI middleware, αυτό γίνεται πραγματικά outermost αλλά μπορεί να αλλάξει το import target. Τεκμηρίωσε το τελικό graph.

BaseHTTPMiddleware και pure ASGI

Το `@app.middleware("http")` είναι βολικό request-response interface. Για λεπτό έλεγχο streaming, ASGI messages ή context propagation, ένα pure ASGI middleware είναι πιο ακριβές. Δέχεται `scope`, `receive`, `send`, ελέγχει `scope["type"]` και τυλίγει το `send` για να παρατηρήσει response start.

Μην διαβάζεις ολόκληρο request body σε generic logging middleware. Καταναλώνει memory, μπορεί να επηρεάσει streaming και καταγράφει secrets. Αν χρειάζεται audit payload για συγκεκριμένο command, κάνε safe projection μετά το validation στο application boundary.

Το ίδιο για response body: compression ή body rewrite πρέπει να χειρίζεται πολλαπλά ASGI body events και `more_body`. Ένα implementation που υποθέτει ένα μοναδικό chunk θα σπάσει streams. Προτίμησε δοκιμασμένο Starlette middleware για standard behavior.

Timing που έχει νόημα

Το middleware timing μετρά συνήθως μέχρι να επιστρέψει response object ή μέχρι να ολοκληρωθεί η `call_next`, ανά implementation. Για streaming, το συνολικό send duration μπορεί να είναι πολύ μεγαλύτερο. Χρειάζεσαι ξεχωριστές μετρήσεις: time to response start, total stream duration και bytes sent. Μη συγκρίνεις ανόμοια metrics σαν ένα latency histogram.

Χρησιμοποίησε route template ως low-cardinality label, όχι raw path με UUID. Το route μπορεί να μην έχει γίνει match όταν τρέχει outer middleware. Συλλογή metadata μετά το inner app ή ASGI scope inspection πρέπει να χειρίζεται 404.

SOURCE TRACE

Το decorator `middleware("http")` στο `fastapi/applications.py` προσθέτει Starlette `BaseHTTPMiddleware`. Το ίδιο αρχείο χτίζει το stack γύρω από τον router και τους exception handlers. Τα request και response objects που χρησιμοποιεί το FastAPI εξάγονται από το Starlette.

ΠΑΓΙΔΑ

Generic body logging middleware μπορεί να αντιγράψει passwords, tokens και προσωπικά δεδομένα σε πολύ λιγότερο προστατευμένο storage από την κύρια βάση.

ΜΗΧΑΝΙΣΜΟΣ

Middleware κατέχει cross-cutting transport behavior. Request state είναι request-local. Domain authorization και workflows μένουν έξω από το middleware.

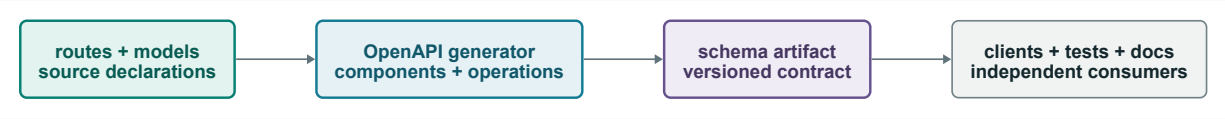
PROBE

Φτιάξε τρία middleware που γράφουν `enter` και `exit`. Δοκίμασε success, handled HTTP exception, unexpected exception και streaming response. Assert σειρά, headers και timing points. Επανέλαβε με pure ASGI outer wrapper ώστε να δεις ποια failures παρατηρεί το κάθε layer.

ΚΕΦΑΛΑΙΟ 13

OpenAPI ως deployed contract

Το OpenAPI είναι προϊόν του build που καταναλώνουν άνθρωποι και εργαλεία. Αν δεν το κάνεις diff, μπορεί να αλλάξει χωρίς κανείς να το προσέξει.



Σχήμα 13.1 · Routes και models γίνονται versioned schema artifact.

Το FastAPI παράγει OpenAPI 3.1.0 από default στην έκδοση αναφοράς. Συνδυάζει routes, parameters, request bodies, response models, security schemes και metadata. Το `/docs` και `/redoc` είναι views πάνω σε αυτό το schema. SDK generators, contract tests και API gateways μπορεί επίσης να το καταναλώνουν.

Δώσε στο app πραγματικό title, API version, summary και description. Δώσε σε κάθε operation συγκεκριμένο summary, tags και responses. Το Python docstring μπορεί να τροφοδοτήσει description, αλλά μην αφήνεις εσωτερικές σημειώσεις να γίνουν public documentation.

PYTHON

```
from fastapi import FastAPI
from pydantic import BaseModel


class OrderView(BaseModel):
    id: str
    state: str


def operation_id(route) -> str:
    method = sorted(route.methods or {"get"})[0].lower()
    return f"{route.name}_{method}"


app = FastAPI(
    title="Orders API",
    version="2026-09",
    generate_unique_id_function=operation_id,
)


@app.get(
    "/orders/{order_id}",
    response_model=OrderView,
    summary="Read one order",
    responses={404: {"description": "Order not found"}},
)
async def get_order(order_id: str) -> OrderView:
    return OrderView(id=order_id, state="pending")
```

Operation id είναι συχνά method name σε generated client. Αν αλλάξει επειδή μετονομάστηκε local function, οι consumers βλέπουν source-level breaking change ακόμη κι αν το HTTP path έμεινε ίδιο. Χρησιμοποίησε deterministic και unique convention. Το απλό example θέλει επιπλέον duplicate check αν δύο routes έχουν ίδιο name και method.

Schema generation και cache

Το `app.openapi()` δημιουργεί schema lazily και το αποθηκεύει στο `app.openapi_schema`. Στο 0.141.1 η cache συνδέεται και με router version ώστε να ανανεώνεται όταν αλλάζουν routes. Παρ' όλα αυτά, dynamic route registration μετά το startup είναι δύσκολο operational model. Κράτα composition deterministic και παρήγαγε artifact στο build.

Για custom schema, φύλαξε το original generator ή κάλεσε `get_openapi`, πρόσθεσε deterministic extensions και cache το αποτέλεσμα. Μην κάνεις network I/O μέσα στο `/openapi.json`. Documentation endpoint πρέπει να είναι φθηνό και να μην εξαρτάται από provider.

Contract diff, όχι απλό JSON diff

Τα JSON object keys μπορεί να αλλάξουν σειρά χωρίς σημασία. Ένα semantic diff πρέπει να αναγνωρίζει αφαίρεση path, νέο required field, στενότερο enum, αλλαγμένο type ή status response ως πιθανό breaking change. Προσθήκη optional field είναι συνήθως backward-compatible για tolerant clients, αλλά generated clients και strict decoders μπορεί να έχουν άλλη πολιτική.

Κράτα canonical `openapi.json` στο release artifact. Στο CI σύγκρινέ το με την τελευταία deployed έκδοση. Το review πρέπει να δείχνει ποια consumer behavior αλλάζει, όχι απλώς χιλιάδες regenerated γραμμές.

Documentation και πραγματικότητα

Το `responses={...}` τεκμηριώνει πιθανό response. Δεν εγκαθιστά handler και δεν εγγυάται body shape. Αν όλες οι errors περνούν από κοινό problem model, δήλωσέ το. Μετά γράψε integration tests που προκαλούν κάθε error. Το ίδιο ισχύει για security scopes: το OpenAPI lock icon δεν αποδεικνύει authorization.

Direct `Response`, custom serialization και middleware μπορούν να αλλάξουν το wire αποτέλεσμα χωρίς το schema να το ξέρει. Για critical operations κράτα golden HTTP examples ή consumer-driven contracts μαζί με το OpenAPI.

API version και application version

Το `app.version` είναι metadata, όχι routing strategy. Semantic package version, deployment revision και public API version είναι διαφορετικά πράγματα. Μπορείς να κάνεις δέκα deploys χωρίς να αλλάξεις public contract. Μπορείς επίσης να τρέχεις `/v1` και `/v2` στο ίδιο binary με διαφορετικά DTOs.

Versioning δεν θεραπεύει κακό migration design. Προτίμησε additive αλλαγές, deprecation signals και measured consumer usage. Νέα major version χρειάζεται owner, sunset policy και συγκεκριμένο σχέδιο για παλιούς clients.

SOURCE TRACE

Η `FastAPI.openapi` στο `fastapi/applications.py` καλεί `get_openapi` από `fastapi/openapi/utils.py` και αποθηκεύει schema μαζί με router version. Τα OpenAPI Pydantic models βρίσκονται στο `fastapi/openapi/models.py`. Η default OpenAPI version του app είναι `3.1.0`.

ΠΑΓΙΔΑ

Το schema μπορεί να είναι σωστό και ο runtime handler να επιστρέφει άλλο body, ειδικά όταν χρησιμοποιεί direct `Response`. Χρειάζονται executable contract tests.

ΜΗΧΑΝΙΣΜΟΣ

Παρήγαγε, αποθήκευσε και κάνε semantic diff το OpenAPI σε κάθε release. Κράτα stable operation ids και σύνδεσε δηλωμένα responses με πραγματικά tests.

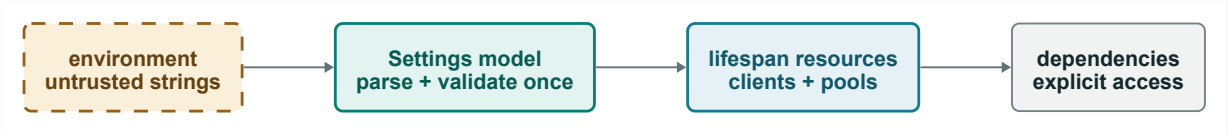
PROBE

Αποθήκευσε το `app.openapi()` ως canonical JSON. Κάνε μία αλλαγή κάθε φορά: πρόσθεσε optional field, κάνε field required, αφαίρεσε enum value και άλλαξε operation name. Δες τι πιάνει το απλό diff και γράψε policy που ταξινομεί τη συμβατότητα κάθε αλλαγής.

ΚΕΦΑΛΑΙΟ 14

Settings και secrets boundary

Configuration είναι untrusted input του process. Διάβασέ την μία φορά, επικύρωσέ την και πέρασέ την ρητά στα resources που τη χρειάζονται.



Σχήμα 14.1 · Environment strings γίνονται validated settings και runtime resources.

Environment variables είναι strings με ασαφές provenance. Ένα typo σε database URL, timeout ή environment name πρέπει να αποτύχει στο startup, όχι στο πρώτο request τρεις ώρες αργότερα. Το `pydantic-settings` διαχωρίζεται από το core Pydantic package και δίνει typed `BaseSettings`.

PYTHON

```
from functools import lru_cache
from typing import Literal

from pydantic import Field, SecretStr
from pydantic_settings import BaseSettings, SettingsConfigDict


class Settings(BaseSettings):
    model_config = SettingsConfigDict(
        env_prefix="ORDERS_",
        env_file=None,
        extra="ignore",
    )

    environment: Literal["development", "test", "production"]
    database_url: SecretStr
    provider_timeout_seconds: float = Field(gt=0, le=30)
    max_page_size: int = Field(ge=1, le=500)


@lru_cache
def load_settings() -> Settings:
    return Settings()
```

Το `SecretStr` μειώνει accidental display στη representation. Δεν κρυπτογραφεί τη μνήμη και όποιος καλέσει `get_secret_value()` παίρνει το secret. Κάνε unwrap μόνο όταν δημιουργείς το client που το χρειάζεται. Μην περάσεις ολόκληρο Settings object σε κάθε component, γιατί τότε κάθε component αποκτά πρόσβαση σε όλα τα secrets.

Cache και test isolation

Το `lru_cache` είναι κοινό pattern ώστε τα settings να διαβαστούν μία φορά ανά process. Σε tests πρέπει να καθαρίζεται ή, καλύτερα, η application factory να δέχεται έτοιμο `Settings`. Αλλαγή environment variable μετά την πρώτη κλήση δεν αλλάζει cached object. Αυτό είναι feature στο production και παγίδα σε test suite χωρίς isolation.

Settings object καλό είναι να αντιμετωπίζεται ως immutable snapshot. Runtime feature flags που αλλάζουν χωρίς deploy δεν είναι environment settings. Είναι external state με caching, failure policy και audit. Μην τα αναμειγνύεις.

`.env` και production

Το `.env` είναι άνετο σε local development, αλλά δεν πρέπει να γίνει κρυφή πηγή production truth ή να μπει στο repository με credentials. Δήλωσε `env_file` μόνο στην τοπική composition policy αν το χρειάζεσαι. Σε production χρησιμοποίησε το secret injection mechanism της πλατφόρμας και περιορισμένα permissions.

Μη βάζεις insecure production defaults για critical values. Αν λείπει signing key ή database URL, το startup πρέπει να αποτύχει. Defaults είναι κατάλληλα για μη ευαίσθητες bounded επιλογές όπως page size. Ακόμη κι εκεί, το default πρέπει να είναι ασφαλές για production load.

Derived configuration

Μην επαναϋπολογίζεις connection limits και timeout budgets σε κάθε request. Μετέτρεψε configuration σε συγκεκριμένα client options κατά το lifespan. Κράτα validation για σχέσεις: provider timeout μικρότερο από request deadline, database pool όχι μεγαλύτερο από το συνολικό budget του deployment και allowed origins χωρίς wildcard όταν χρησιμοποιούνται credentials.

Μερικοί κανόνες χρειάζονται model validator. Άλλοι εξαρτώνται από deployment topology και ελέγχονται στο release pipeline. Το Pydantic δεν γνωρίζει πόσους workers θα ξεκινήσει το orchestrator.

Rotation χωρίς έκθεση

Secret rotation μπορεί να απαιτεί restart ή client refresh. Δήλωσε ποιο από τα δύο υποστηρίζεται. Για signing keys, συχνά χρειάζεται set από active key ids ώστε παλιά tokens να επαληθεύονται για μικρό overlap. Για database password, rotation πρέπει να συντονιστεί με connections που παραμένουν ανοιχτές.

Σε logs γράψε configuration fingerprint ή safe fields, ποτέ raw secrets. Redaction μόνο πάνω σε key names δεν αρκεί, επειδή token μπορεί να βρεθεί σε generic `value` ή URL. Η ασφαλέστερη επιλογή είναι allowlist των fields που επιτρέπεται να καταγραφούν.

SOURCE TRACE

Το FastAPI δεν κατέχει configuration loader. Η επίσημη κατεύθυνση χρησιμοποιεί το ξεχωριστό `pydantic-settings`. Το dependency system μπορεί να προσφέρει settings, ενώ το `lifespan` του `FastAPI` είναι το κατάλληλο composition point για resources που κατασκευάζονται από αυτά.

ΠΑΓΙΔΑ

Το `SecretStr` προστατεύει κυρίως από accidental repr. Δεν κάνει ασφαλές ένα dump, ένα explicit unwrap ή ένα request log που περιέχει το credential.

ΜΗΧΑΝΙΣΜΟΣ

Parse configuration μία φορά, fail fast και δώσε σε κάθε adapter μόνο τις ρυθμίσεις που χρειάζεται. Secrets δεν είναι γενικό application context.

PROBE

Ξεκίνα app με missing secret, invalid timeout και άγνωστο environment. Assert startup failure πριν από readiness. Σε test άλλαξε env μετά το πρώτο load και παρατήρησε την cache. Τέλος, σάρωσε captured logs και exception strings για γνωστή canary secret value.

ΚΕΦΑΛΑΙΟ 15

Authentication, passwords και tokens

Authentication μετατρέπει credential σε verified principal. Το FastAPI μπορεί να εξαγάγει το token και να τεκμηριώσει το scheme. Η πραγματική επαλήθευση παραμένει δική σου security boundary.

Σχήμα 15.1 · Credential, verification και principal είναι ξεχωριστά βήματα.

Το `OAuth2PasswordBearer` διαβάζει Bearer token από το Authorization header, επιστρέφει 401 όταν λείπει και προσθέτει OAuth2 security scheme στο OpenAPI. Δεν επαληθεύει JWT signature, issuer, audience ή expiry. Το όνομά του δεν σημαίνει ότι ολοκλήρωσες OAuth2 server.

PYTHON

```
from dataclasses import dataclass
from typing import Annotated

from fastapi import Depends, FastAPI, HTTPException, status
from fastapi.security import OAuth2PasswordBearer

app = FastAPI()
bearer = OAuth2PasswordBearer(tokenUrl="/auth/token")


@dataclass(frozen=True)
class Principal:
    subject: str
    roles: frozenset[str]


def decode_and_verify(token: str) -> Principal:
    # Delegate cryptography to a reviewed library and fixed algorithm policy.
    if token != "verified-demo-token":
        raise ValueError("invalid token")
    return Principal(subject="user-123", roles=frozenset({"member"}))


async def current_principal(
    token: Annotated[str, Depends(bearer)],
) -> Principal:
    try:
        return decode_and_verify(token)
    except ValueError:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="invalid_credentials",
            headers={"WWW-Authenticate": "Bearer"},
        ) from None
```

Σε πραγματική υλοποίηση, ο decoder επιτρέπει συγκεκριμένους algorithms και ελέγχει signature, `iss`, `aud`, `exp`, `nbf` και όποια claims απαιτεί το protocol. Μην επιλέγεις algorithm από μη επαληθευμένο header χωρίς allowlist. Μη χρησιμοποιείς το token payload πριν ολοκληρωθεί η signature verification.

JWT είναι signed container, όχι session strategy

Ένα συνηθισμένο JWT είναι readable από όποιον το έχει. Μην βάζεις secrets ή ευαίσθητα profile data. Κράτα μικρό lifetime, stable subject και ελάχιστες claims. Μεγάλο token στέλνεται σε κάθε request και μπορεί να καταλήξει σε proxy limits ή logs.

Stateless verification δεν σημαίνει μηδενικό state. Χρειάζεσαι key rotation, issuer configuration, clock policy και απόφαση ανάκλησης. Αν πρέπει να κόψεις άμεσα πρόσβαση, short-lived access token μαζί με server-side account state ή revocation version είναι συνηθισμένη λύση. Κάθε online lookup αυξάνει latency και availability coupling, άρα χρειάζεται cache και fail policy.

Key id βοηθά επιλογή verification key. Unknown id δεν πρέπει να προκαλεί unbounded refresh από remote key endpoint σε κάθε malicious request. Κάνε bounded caching, refresh coalescing και stale-key policy.

Password login

Password δεν αποθηκεύεται ποτέ plaintext ή με γρήγορο general-purpose hash. Χρησιμοποίησε σύγχρονη password hashing library με memory-hard algorithm και ρυθμίσεις που μετρήθηκαν στο δικό σου hardware. Αποθήκευσε encoded hash που περιλαμβάνει algorithm και parameters ώστε να μπορείς να κάνεις rehash μετά από επιτυχημένο login.

Το login error πρέπει να είναι generic για unknown user και wrong password. Διαφορετικά κάνεις account enumeration. Παρ' όλα αυτά, timing και rate behavior πρέπει επίσης να μην αποκαλύπτουν εύκολα την ύπαρξη account. Βάλε rate limits ανά πολλαπλές διαστάσεις, monitoring και ασφαλές recovery flow. Μην κλειδώνεις μόνιμα account επειδή attacker μπορεί να προκαλέσει denial of service.

Bearer token ή cookie

Authorization header ταιριάζει σε API clients. Browser εφαρμογή μπορεί να χρησιμοποιεί secure HttpOnly cookie ώστε JavaScript να μην διαβάζει token, αλλά τότε χρειάζεται CSRF design. `SameSite`, origin checks και anti-CSRF token εξαρτώνται από τη ροή και τα cross-site requirements.

Token σε `localStorage` είναι προσβάσιμο σε injected JavaScript. Cookie δεν θεραπεύει XSS, αλλά περιορίζει direct token theft όταν είναι HttpOnly. Διάλεξε με threat model και όχι από framework tutorial.

Principal ως εσωτερικό contract

Μετά την επαλήθευση, πέρασε immutable principal με subject, tenant και trusted claims. Μην περνάς raw token σε repositories. Μην θεωρείς display email μόνιμη identity. Το domain χρησιμοποιεί stable internal id και κάνει authorization πάνω σε τρέχον resource state.

Authentication failure καταγράφεται χωρίς token. Χρήσιμα safe signals είναι reason category, issuer class, route template και request id. Μην βάζεις user controlled subject σε metric label με απεριόριστη cardinality.

SOURCE TRACE

Τα security dependencies βρίσκονται στο `fastapi/security/`. Η `OAuth2PasswordBearer.__call__` στο `fastapi/security/oauth2.py` εξάγει το Bearer token από το request και χειρίζεται absence. Τα OpenAPI security models περιγράφονται στο ίδιο module και στο `fastapi/openapi/models.py`. Η cryptographic verification δεν υλοποιείται από το extractor.

ΠΑΓΙΔΑ

Η αποκωδικοποίηση JWT χωρίς signature verification παράγει attacker-controlled claims. «Μπορώ να διαβάσω το payload» δεν σημαίνει «ξέρω ποιος το υπέγραψε».

ΜΗΧΑΝΙΣΜΟΣ

Credential extraction, cryptographic verification, principal construction και authorization είναι τέσσερα διαφορετικά βήματα. Κάνε καθένα explicit.

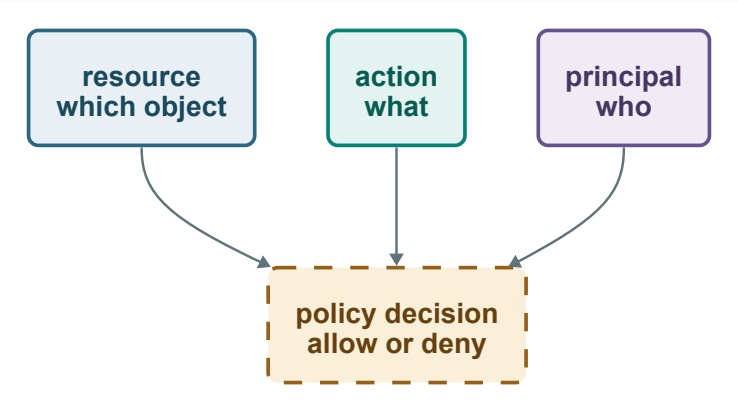
PROBE

Δοκίμασε missing header, λάθος scheme, bad signature, expired token, wrong issuer, wrong audience και unknown key id. Assert ίδιο safe 401 envelope όπου δεν πρέπει να διαρρεύσει λεπτομέρεια, σωστό `WWW-Authenticate` και μηδενική καταγραφή του raw token.

ΚΕΦΑΛΑΙΟ 16

Authorization και object ownership

Authentication απαντά ποιος είναι ο caller. Authorization αποφασίζει αν αυτός ο principal μπορεί να κάνει αυτή την πράξη πάνω σε αυτό το συγκεκριμένο state.



Σχήμα 16.1 · Principal, action και resource οδηγούν σε policy decision.

Ένα role στο token είναι input της authorization απόφασης, όχι η απόφαση η ίδια. Το `admin` μπορεί να είναι stale, να ισχύει μόνο σε tenant ή να επιτρέπει ανάγνωση αλλά όχι refund. Καθάρισε τη γλώσσα σου: principal, action, resource, context και decision.

Για object-level authorization, φόρτωσε resource και έλεγξε ownership ή policy στο ίδιο application workflow. Αν route dependency ελέγχει μόνο generic scope και το repository αργότερα φορτώνει οποιοδήποτε id, έχεις κλασικό insecure direct object reference.

PYTHON

```
from dataclasses import dataclass
from typing import Protocol


@dataclass(frozen=True)
class Principal:
    user_id: str
    tenant_id: str
    permissions: frozenset[str]


@dataclass(frozen=True)
class Order:
    id: str
    tenant_id: str
    owner_id: str
    state: str
```

```
class Orders(Protocol):
    async def get(self, order_id: str) -> Order | None: ...


class Forbidden(Exception):
    pass


async def cancel_order(
    principal: Principal,
    order_id: str,
    orders: Orders,
) -> Order:
    order = await orders.get(order_id)
    if order is None or order.tenant_id != principal.tenant_id:
        raise Forbidden()
    owns_order = order.owner_id == principal.user_id
    if not owns_order and "orders:cancel:any" not in principal.permissions:
        raise Forbidden()
    if order.state != "pending":
        raise ValueError("order_not_cancellable")
    return order
```

Το example ξεχωρίζει permission από domain transition. Ο principal πρέπει να έχει πρόσβαση στο order και το order πρέπει να είναι σε cancellable state. Το ένα δεν αντικαθιστά το άλλο. Σε πραγματικό write, ο έλεγχος state και το update πρέπει να γίνουν atomically ή με optimistic version.

403 ή 404;

Το 403 δηλώνει ότι η ταυτότητα έγινε κατανοητή αλλά η πράξη απαγορεύεται. Σε multi-tenant ή private resources, η ίδια η ύπαρξη μπορεί να είναι sensitive. Τότε unknown id και foreign-tenant id συχνά επιστρέφουν ίδιο 404 envelope και παρόμοιο timing. Η πολιτική πρέπει να είναι συνεπής, όχι διαφορετική ανά route.

Μην εφαρμόζεις αυτή την απόκρυψη μόνο στο response body. Διαφορές σε timing, cache headers ή error detail μπορούν να αποκαλύψουν existence. Δεν χρειάζεται τέλεια constant-time database συμπεριφορά, αλλά απόφυγε προφανές branch όπου το foreign resource κάνει ακριβή δεύτερη query και το missing όχι.

Scopes, roles και permissions

OAuth scopes είναι strings που περιγράφουν delegated capability και εμφανίζονται στο OpenAPI. Το `Security` dependency μπορεί να περάσει required scopes σε verifier μέσω `SecurityScopes`. Αυτό είναι χρήσιμο στο transport boundary. Πάλι, scope `orders:write` δεν αποδεικνύει ownership συγκεκριμένου order.

Roles είναι bundles που αλλάζουν ευκολότερα server-side. Permissions είναι πιο λεπτές capabilities. Attributes όπως tenant, region, account state και resource owner οδηγούν σε attribute-based policy. Μην βάζεις ολόκληρη policy σε token που ζει μία ώρα αν πρέπει να αλλάζει άμεσα.

Κράτα authorization API θετικό και μικρό: `can_cancel(principal, order)` ή policy object με explicit action. Generic expression engine αξίζει μόνο όταν υπάρχει πραγματική ανάγκη, tooling και audit. Διαφορετικά κάνει απλές αποφάσεις αόρατες.

Lists και bulk operations

Σε list endpoint δεν φορτώνεις όλα τα rows και μετά φιλτράρεις σε Python. Το repository query πρέπει να ενσωματώνει tenant και visibility constraints, ώστε unauthorized data να μη βγει ποτέ από τη database boundary. Count, pagination και search πρέπει να χρησιμοποιούν το ίδιο filter, αλλιώς metadata διαρρέει κρυφά resources.

Bulk command χρειάζεται authorization ανά item ή query που περιορίζει atomic update στα allowed rows. Αν δέχεσαι partial success, το response protocol πρέπει να δείχνει outcome ανά item χωρίς να αποκαλύπτει forbidden identities. Αν απαιτείς all-or-nothing, έλεγξε όλα μέσα σε transaction πριν από mutation.

TOCTOU και consistent decision

Αν ελέγξεις permission, κλείσεις transaction και μετά γράψεις με νέα session, το resource ή membership μπορεί να αλλάξει ενδιάμεσα. Βάλε authorization read και protected transition στο ίδιο isolation boundary ή χρησιμοποίησε expected version. Database constraints μπορούν να προστατεύσουν tenant foreign keys και state transitions που εκφράζονται δομικά.

Κατέγραψε security-relevant decisions με actor, action, target type, outcome και policy version, όχι raw token. Audit log πρέπει να έχει access controls και retention. Δεν είναι debug dump.

SOURCE TRACE

Το `Security` στο `fastapi/param_functions.py` επεκτείνει το dependency metadata με scopes. Η `SecurityScopes` στο `fastapi/security/oauth2.py` δίνει τα required scopes στο verifier. Το FastAPI κατασκευάζει και τεκμηριώνει αυτόν τον graph, αλλά η object-level policy ανήκει στην εφαρμογή.

ΠΑΓΙΔΑ

Filter μετά το query μπορεί να έχει ήδη διαρρεύσει counts, timing ή data σε logs και caches. Το tenant boundary πρέπει να μπει στην ίδια την data access operation.

ΜΗΧΑΝΙΣΜΟΣ

Η πλήρης απόφαση είναι `principal + action + current resource + context`. Generic scope στο route είναι μόνο ένα από τα inputs.

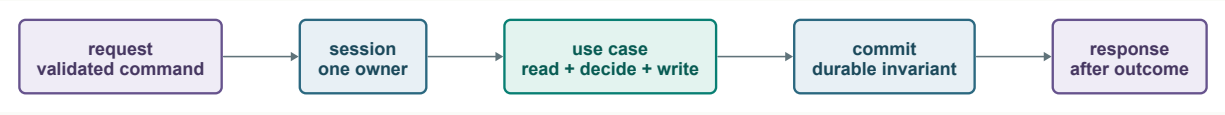
PROBE

Για κάθε protected operation δοκίμασε owner, άλλον user στον ίδιο tenant, ίδιο user σε άλλο tenant, privileged role, suspended account και resource που αλλάζει version ανάμεσα σε read και write. Assert response, durable state και audit event χωρίς existence leak.

ΚΕΦΑΛΑΙΟ 17

SQLAlchemy sessions και transactions

Session είναι unit of work, όχι global database handle. Κάθε business command χρειάζεται έναν owner που αποφασίζει πότε γίνεται commit και τι είναι atomic.



Σχήμα 17.1 · Validated command, session, use case και commit σχηματίζουν ένα boundary.

Το FastAPI δεν επιβάλλει ORM. Με SQLAlchemy 2.0, ένα `Engine` και το connection pool είναι long-lived process resources. Μία `Session` ή `AsyncSession` είναι βραχύβια μονάδα εργασίας και δεν μοιράζεται ανάμεσα σε concurrent requests ή tasks. Το dependency μπορεί να κατέχει creation και close. Το use case κατέχει την transaction policy.

PYTHON

```
from collections.abc import AsyncIterator
from typing import Annotated

from fastapi import Depends
from sqlalchemy.ext.asyncio import (
    AsyncSession,
    async_sessionmaker,
    create_async_engine,
)

engine = create_async_engine(
    "postgresql+asyncpg://app@db/orders",
    pool_size=10,
    max_overflow=0,
    pool_pre_ping=True,
)
SessionFactory = async_sessionmaker(engine, expire_on_commit=False)


async def session_dependency() -> AsyncIterator[AsyncSession]:
    async with SessionFactory() as session:
        yield session


DbSession = Annotated[
    AsyncSession,
    Depends(session_dependency, scope="function"),
]
```

Το URL είναι placeholder, όχι secret-management recommendation. Σε application factory το engine ανοίγει στο lifespan από validated settings και κάνει `await engine.dispose()` στο shutdown. Η module-level μορφή δείχνει μόνο τις σχέσεις των objects.

Explicit transaction

Για write use case, κάνε transaction boundary ορατό:

```python
from sqlalchemy.ext.asyncio import AsyncSession


async def place_order(command, session: AsyncSession):
    async with session.begin():
        customer = await load_customer(session, command.customer_id)
        order = build_order(customer, command.lines)
        session.add(order)
        session.add(outbox_event_for(order))
    return order.to_view()
```

Το context κάνει commit στην επιτυχή έξοδο και rollback σε exception. Business row και outbox event γράφονται atomically. Το returned view φτιάχνεται όσο τα απαραίτητα attributes είναι διαθέσιμα. Αν το commit μπορεί να παράγει generated values, κάνε flush ή refresh στο κατάλληλο σημείο πριν το projection.

Automatic commit στον κώδικα μετά το dependency `yield` κρύβει μια κρίσιμη failure. Η endpoint function μπορεί να έχει «επιστρέψει» success result πριν αποτύχει constraint στο commit. Function scope επιτρέπει ακόμη να μετατραπεί σε error πριν το send, αλλά το workflow παραμένει δύσκολο να διαβαστεί. Το explicit commit δείχνει πότε υπάρχει durable success.

Constraints είναι τελευταίος arbiter

Το `if not exists: insert` έχει race. Δύο requests μπορούν να διαβάσουν absence και να γράψουν μαζί. Unique constraint προστατεύει το invariant. Η εφαρμογή μεταφράζει τη συγκεκριμένη integrity violation σε conflict. Μην μεταφράζεις κάθε `IntegrityError` σε duplicate, επειδή μπορεί να είναι foreign key bug ή λάθος schema.

Για optimistic concurrency βάλε version column και atomic update με `WHERE id = :id AND version = :expected`. Zero affected rows σημαίνει missing ή stale και ίσως χρειάζεται δεύτερο bounded read για σωστή χαρτογράφηση. Row lock είναι εργαλείο όταν η transaction κρατιέται σύντομη. Ποτέ μην κρατάς lock ενώ περιμένεις external HTTP provider.

Async δεν σημαίνει parallel session

Η `AsyncSession` δεν είναι ασφαλής για ταυτόχρονη χρήση από πολλά tasks. Αν ξεκινήσεις task group και δώσεις την ίδια session σε όλα, οι operations και το transaction state συγκρούονται. Κράτα sequential unit of work ή δώσε ανεξάρτητη session και transaction σε κάθε πραγματικά ανεξάρτητη εργασία.

Async driver επιτρέπει στο event loop να εξυπηρετεί άλλα requests όσο περιμένει network I/O. Η database όμως έχει finite connections, locks και CPU. Pool size ανά worker πολλαπλασιάζεται με worker count. Τέσσερις workers επί δέκα connections είναι σαράντα πιθανές connections, πριν από jobs και migrations.

Query shape και serialization

Μην αφήνεις response serialization να ανακαλύψει relationships. Δήλωσε eager loading ή projection για συγκεκριμένο endpoint. Μετρά query count σε tests. Pagination query, count query και returned ordering πρέπει να χρησιμοποιούν ίδιο visibility filter.

Statement timeout προστατεύει από runaway query. Request timeout μόνο του μπορεί να ακυρώσει coroutine ενώ η database συνεχίζει δουλειά ανά driver. Ρύθμισε timeouts σε κάθε layer και επαλήθευσε cancellation behavior.

Migrations είναι release concern. Μην καλείς `create_all` στο web startup σε production. Με πολλούς workers υπάρχει race και με rolling release παλιά και νέα έκδοση πρέπει να συνυπάρχουν με το ίδιο schema.

SOURCE TRACE
Το FastAPI παρέχει lifecycle μέσω yield dependencies και δεν γνωρίζει SQLAlchemy transaction semantics. Στο `fastapi/routing.py` τα function-scoped resources κλείνουν πριν σταλεί το response. Engine, session factory, isolation και commit behavior ανήκουν στη SQLAlchemy 2.0 API και στην εφαρμογή.

ΠΑΓΙΔΑ
Μία global `AsyncSession` μπορεί να διαρρεύσει identity map και transaction state ανάμεσα σε users. Shared είναι το engine και pool, όχι η session.

ΜΗΧΑΝΙΣΜΟΣ
Ένα command έχει μία explicit transaction. Η database constraint προστατεύει το invariant όταν δύο requests αγωνιστούν. External effects βγαίνουν μέσω outbox, όχι μέσα σε ανοιχτό lock.

PROBE
Στείλε δύο concurrent creates με ίδιο business key και assert μία row και ένα 409. Προκάλεσε exception μετά το πρώτο insert και assert rollback. Μέτρα total connections με πολλούς workers και επιβεβαίωσε ότι το deployment budget δεν ξεπερνά το database limit.

ΚΕΦΑΛΑΙΟ 18

Use cases, repositories και adapters

Το FastAPI είναι ένας adapter εισόδου. Το application core πρέπει να μπορεί να εκτελέσει το ίδιο workflow από HTTP, job ή test χωρίς fake Request.

Σχήμα 18.1 · HTTP adapter, use case, domain, ports και infrastructure adapters.

Thin route δεν σημαίνει ότι όλη η λογική μεταφέρθηκε σε `service.py` με 2.000 γραμμές. Σημαίνει ότι κάθε layer έχει απόφαση που κατέχει. Το route κατέχει HTTP mapping. Το use case κατέχει workflow και transaction. Το domain κατέχει invariants. Το repository κατέχει persistence semantics. Ο provider adapter κατέχει external protocol, timeouts και response parsing.

Τα Python `Protocol` δίνουν μικρά ports χωρίς framework. Δεν χρειάζεται abstract base class για κάθε dependency.

PYTHON

```
from dataclasses import dataclass
from datetime import datetime
from typing import Protocol


@dataclass(frozen=True)
class CreateOrder:
    customer_id: str
    reference: str


@dataclass(frozen=True)
class CreatedOrder:
    order_id: str
    created_at: datetime


class OrderRepository(Protocol):
    async def create_once(
        self,
        command: CreateOrder,
        now: datetime,
    ) -> CreatedOrder: ...


class Clock(Protocol):
    def now(self) -> datetime: ...


async def create_order(
    command: CreateOrder,
    orders: OrderRepository,
```

```
    clock: Clock,
) -> CreatedOrder:
    return await orders.create_once(command, clock.now())
```

Το `create_once` είναι purpose-specific port. Ένα generic repository με `find`, `save`, `delete` συχνά διαρρέει ORM semantics και αναγκάζει το use case να υλοποιήσει race-prone read-then-write. Το συγκεκριμένο method μπορεί να υπόσχεται unique constraint, transaction και σαφή duplicate outcome.

Commands και results

Το API request model μετατρέπεται σε immutable command. Ο principal περνά ξεχωριστά αν η identity δεν πρέπει να πλαστογραφείται από body. Το result είναι application outcome ή view data, όχι Starlette Response. Ο route adapter το μετατρέπει σε output model και HTTP status.

Μην χρησιμοποιείς exceptions για κάθε φυσιολογικό branch αν ένα sealed-like result union είναι πιο καθαρό. Από την άλλη, μην επιστρέφεις generic `Result[dict, str]` που κρύβει όλες τις περιπτώσεις. Συγκεκριμένα types όπως `Created`, `AlreadyExists`, `Forbidden` και `DependencyUnavailable` κάνουν την exhaustive mapping εφικτή.

Πού μπαίνει η transaction

Το use case γνωρίζει ποια persistence operations αποτελούν μία atomic αλλαγή. Μπορεί να πάρει Unit of Work port με repositories και `commit`, ή request session από adapter. Το domain entity δεν πρέπει να κάνει commit. Ο route handler επίσης δεν πρέπει να κάνει τρία ανεξάρτητα commits γύρω από ένα business command.

External HTTP call δεν μπορεί να συμμετέχει στην ίδια database transaction. Αν χρειάζεται reliable effect, γράψε intent σε outbox και άφησε worker να το εκτελέσει idempotently. Αν το provider πρέπει να απαντήσει synchronously, μοντελοποίησε unknown outcome και reconciliation. Μην προσποιείσαι atomicity.

Adapters με πλήρες contract

Ένας provider adapter δεν είναι απλό `client.post`. Κατέχει base URL, authentication, request mapping, connect/read/write/pool timeouts, retry classification, idempotency key, response validation και redaction. Μετατρέπει provider-specific failures σε μικρό application taxonomy.

Retry library δεν αποφασίζει τι είναι safe. Adapter και use case γνωρίζουν αν η operation είναι idempotent και αν timeout συνέβη πριν ή μετά από πιθανό send. Budget όλων των attempts πρέπει να χωρά στο caller deadline.

Dependency composition

FastAPI dependencies κατασκευάζουν ports ή use case object από request session, settings και principal. Μετά η κλήση είναι κανονική Python. Αυτό κρατά test override στο boundary. Τα use case tests περνούν fakes απευθείας χωρίς να ξεκινούν ASGI app.

Μην κάνεις import concrete SQLAlchemy repository μέσα στο domain. Το composition root διαλέγει implementation. Μην κάνεις επίσης interface που αντιγράφει κάθε method του concrete client. Το port εκφράζει τι χρειάζεται το application, όχι ό,τι μπορεί να κάνει ο vendor.

Πότε δεν χρειάζεται layer

Read-only endpoint που κάνει μία bounded query και projection μπορεί να έχει μικρό query service χωρίς πλούσιο domain model. Ένα health endpoint δεν χρειάζεται repository pattern. Structure προστίθεται όταν ξεχωρίζει ownership, test boundary ή failure semantics. Αν ένα νέο file απλώς προωθεί arguments χωρίς απόφαση, πιθανόν δεν προσθέτει αξία.

SOURCE TRACE

Το `get_request_handler` στο `fastapi/routing.py` λύνει dependencies, καλεί `run_endpoint_function` και μετά serializes το result. Αυτό είναι το φυσικό seam: πριν από την endpoint function βρίσκεται το FastAPI transport machinery, ενώ η εφαρμογή μπορεί να καλέσει plain use case code.

ΠΑΓΙΔΑ

Ένα folder `services` δεν απομονώνει το domain αν οι functions μέσα του διαβάζουν `Request`, `app.state` και global sessions. Τα imports αποκαλύπτουν το πραγματικό coupling.

ΜΗΧΑΝΙΣΜΟΣ

Ports ονομάζουν application ανάγκες. Adapters υλοποιούν τεχνικά protocols. Το composition root τα συνδέει και το use case παραμένει ανεξάρτητο από FastAPI.

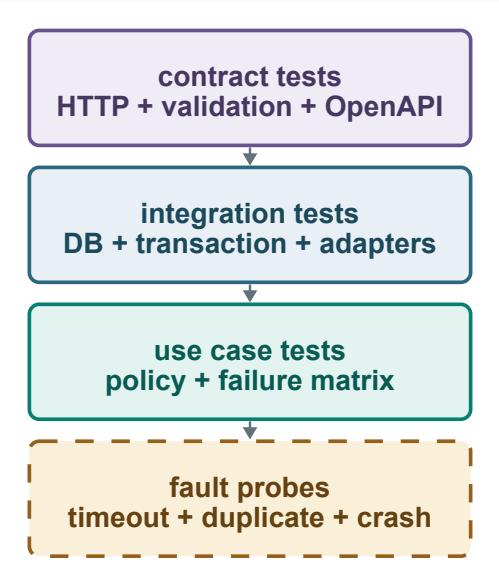
PROBE

Πάρε ένα route που κάνει database write και provider call. Γράψε σε μία γραμμή τον owner κάθε decision και failure. Μετέτρεψε το body σε command και κάλεσε το use case από test χωρίς app. Αν χρειαστεί να κατασκευάσεις Request, το boundary ακόμη διαρρέει.

ΚΕΦΑΛΑΙΟ 19

Tests, overrides και fault injection

Ένα καλό test suite ελέγχει το public HTTP contract, τα data invariants και τα failure windows. Δεν μετρά απλώς πόσες γραμμές εκτελέστηκαν.



Σχήμα 19.1 · Contract, integration, use case και fault probes καλύπτουν διαφορετικούς κινδύνους.

Το `TestClient` προσφέρει synchronous interface πάνω σε ASGI app και βασίζεται στο HTTPX/Starlette test stack. Χρησιμοποίησέ το ως context manager όταν θέλεις να τρέξει lifespan. Έτσι startup και shutdown resources ελέγχονται μαζί με τα requests.

PYTHON

```
from dataclasses import dataclass

from fastapi import Depends, FastAPI
from fastapi.testclient import TestClient


@dataclass(frozen=True)
class Clock:
    value: str


def clock() -> Clock:
    return Clock("production-time")


app = FastAPI()
```

```
@app.get("/time")
async def read_time(value: Clock = Depends(clock)) -> dict[str, str]:
    return {"time": value.value}


def test_clock_override() -> None:
    app.dependency_overrides[clock] = lambda: Clock("fixed-time")
    try:
        with TestClient(app) as client:
            response = client.get("/time")
        assert response.status_code == 200
        assert response.json() == {"time": "fixed-time"}
    finally:
        app.dependency_overrides.clear()
```

Override key είναι το αρχικό callable object. Αν το route χρησιμοποίησε άλλη wrapper instance, το override δεν εφαρμόζεται. Καθάριζε overrides σε fixture teardown ώστε ένα test να μη μολύνει το επόμενο. Σε parallel tests, ξεχωριστό app instance ανά test group είναι ασφαλέστερο από mutation ενός global app.

Τέσσερα είδη απόδειξης

Use case tests περνούν fakes και ελέγχουν policy γρήγορα. Integration tests τρέχουν πραγματική database schema, constraints και repository queries. HTTP contract tests περνούν από routing, validation, auth, error mapping και serialization. End-to-end smoke tests επαληθεύουν το deployed network path. Κανένα επίπεδο δεν αντικαθιστά τα άλλα.

Mock της session που επιστρέφει ό,τι περιμένει ο κώδικας δεν αποδεικνύει SQL, isolation ή unique constraint. Από την άλλη, κάθε edge case μέσω αργού full stack κάνει το suite δύσχρηστο. Βάλε κάθε invariant στο χαμηλότερο επίπεδο που ακόμη ελέγχει τον πραγματικό μηχανισμό.

Async HTTPX

Για async tests, το HTTPX `ASGITransport(app=app)` στέλνει requests χωρίς socket. Η μεταφορά από μόνη της δεν διαχειρίζεται πάντα ASGI lifespan. Χρειάζεται explicit lifespan manager ή test fixture που ανοίγει το app lifecycle. Μην περάσει test επειδή ποτέ δεν άνοιξε το production pool.

Το `base_url="http://test"` είναι απαραίτητο για relative URLs και Host header. Δοκίμασε επίσης headers, cookies, redirects και streaming με τις ίδιες ρυθμίσεις που έχουν σημασία στον πραγματικό client. In-process transport δεν ελέγχει proxy, TLS ή network timeout.

Database isolation

Rollback fixture είναι γρήγορο, αλλά application code που ανοίγει δική του connection ή κάνει commit μπορεί να ξεφύγει από την outer test transaction. Βεβαιώσου ότι η dependency override δίνει την ίδια session strategy που περιμένει το app. Για transaction και concurrency tests χρησιμοποίησε πραγματικά commits σε προσωρινή database και καθάρισε με scoped records ή schema.

Μην τρέχεις integration tests μόνο σε SQLite αν production είναι PostgreSQL. Locking, JSON, constraints, timezone και SQL syntax διαφέρουν. SQLite παραμένει χρήσιμο για tests που δεν ισχυρίζονται production database equivalence.

Fault injection

Fake provider πρέπει να μπορεί να αποτύχει πριν send, μετά από πιθανό accept, με invalid response και με timeout. Repository fake πρέπει να παράγει duplicate και stale version. Clock και id generator γίνονται injected ώστε tests να είναι deterministic.

Για κάθε side effect σχεδίασε crash points. Assert όχι μόνο response, αλλά durable state, outbox row, retry count και observability signal. Concurrency test χρειάζεται barrier ώστε δύο requests να μπουν στο ίδιο race window. Είκοσι τυχαίες concurrent κλήσεις χωρίς synchronization μπορεί να μην αναπαράγουν τίποτα.

Contract artifacts

Snapshot OpenAPI με canonical ordering και semantic review. Κράτα golden examples για error envelopes και webhook signatures. Μην snapshotάρεις timestamps, generated ids ή ολόκληρα headers που αλλάζουν χωρίς νόημα.

Τα tests πρέπει να αποδεικνύουν ότι secrets λείπουν από output και logs. Ένα canary credential στο fixture βοηθά automated scan. Κλείδωσε επίσης query count και maximum response size σε endpoints με γνωστό performance risk.

SOURCE TRACE

Το `fastapi/testclient.py` επανεξάγει το Starlette `TestClient`. Τα dependency overrides διαβάζονται μέσα στο `solve_dependencies` του `fastapi/dependencies/utils.py`, όπου το original callable αντικαθίσταται και αναλύεται νέο dependent graph.

ΠΑΓΙΔΑ

Test που κάνει override όλη την authentication dependency δεν ελέγχει token verification. Κράτα ξεχωριστά verifier tests και λίγα full HTTP auth tests.

ΜΗΧΑΝΙΣΜΟΣ

Δοκίμασε contract με ASGI, invariants με πραγματική data layer και failure windows με ελεγχόμενο fault injection. Τα fakes χρειάζονται συγκεκριμένο scope.

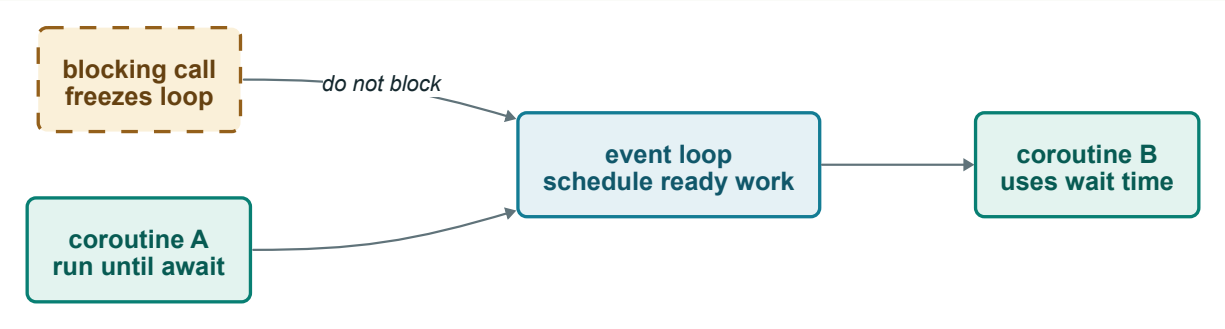
PROBE

Πάρε ένα create endpoint και φτιάξε matrix: valid, malformed, unauthorized, duplicate, database timeout, provider unknown outcome και response serialization bug. Για κάθε γραμμή assert HTTP, durable rows, external calls και log category. Αυτό είναι πιο χρήσιμο από έναν μοναδικό coverage αριθμό.

ΚΕΦΑΛΑΙΟ 20

Async mental model και blocking work

Async κερδίζει όταν πολλές εργασίες περιμένουν I/O και παραχωρούν τον έλεγχο. Δεν κάνει CPU work γρηγορότερο και δεν μετατρέπει blocking library σε async.



Σχήμα 20.1 · Το event loop εναλλάσσει coroutines μόνο στα σημεία συνεργασίας.

Μία coroutine τρέχει μέχρι να συναντήσει `await` που δεν είναι έτοιμο. Τότε το event loop εκτελεί άλλη έτοιμη εργασία. Δεν υπάρχει preemption ανά γραμμή Python. Ένα synchronous sleep, βαρύ JSON transform ή blocking database call μέσα σε `async def` παγώνει όλα τα requests του ίδιου loop.

Το FastAPI εξετάζει αν endpoint και dependency είναι async callable. Τα async καλούνται με `await`. Τα sync περνούν στο thread pool. Αυτό επιτρέπει σταδιακή χρήση sync libraries, αλλά ο pool έχει tokens και τα threads μπορούν να εξαντληθούν. Επίσης ένα thread που συνεχίζει provider call μετά από client disconnect καταναλώνει resource.

Structured concurrency

Παράλληλες κλήσεις έχουν νόημα μόνο όταν είναι ανεξάρτητες και χωρούν στο ίδιο deadline. Το AnyIO task group δίνει κοινό lifecycle: αν ένα child αποτύχει, τα άλλα ακυρώνονται και ο parent περιμένει cleanup.

```python
from typing import Any

from anyio import create_task_group, fail_after


async def load_dashboard(client) -> dict[str, Any]:
    result: dict[str, Any] = {}

    async def load(name: str, path: str) -> None:
        response = await client.get(path)
        response.raise_for_status()
        result[name] = response.json()

    with fail_after(0.8):
        async with create_task_group() as tasks:
            tasks.start_soon(load, "orders", "/orders/summary")
            tasks.start_soon(load, "balance", "/billing/balance")
    return result
```

Το shared dict είναι ασφαλές εδώ μόνο επειδή κάθε task γράφει διαφορετικό key χωρίς `await` ανάμεσα στο read-modify-write. Αυτό δεν είναι γενική thread safety εγγύηση. Για πιο σύνθετη συνεργασία χρησιμοποίησε channels ή lock με explicit invariants.

Deadline, timeout και cancellation

Timeout δεν είναι απλώς exception. Δηλώνει ότι ο caller σταματά να περιμένει. Η downstream operation μπορεί να συνεχίζει ή να έχει ήδη κάνει effect. Βάλε connect, pool, write και read timeout στον HTTP client και συνολικό deadline στο workflow. Το άθροισμα retries και backoff πρέπει να χωρά πριν λήξει το upstream request.

Cancellation πρέπει να διαδίδεται. Cleanup μπαίνει σε `finally` και πρέπει να είναι σύντομο. Shield cancellation μόνο γύρω από μικρό critical cleanup που πρέπει να ολοκληρωθεί, όχι γύρω από ολόκληρο provider call. Αν background εργασία πρέπει να επιβιώσει του request, γράψ' την durable σε queue.

Thread-pool cancellation δεν σταματά αυθαίρετη synchronous function που ήδη τρέχει. Η coroutine μπορεί να πάψει να περιμένει, αλλά το thread συχνά συνεχίζει μέχρι να επιστρέψει. Γι' αυτό η ίδια η library χρειάζεται timeout.

CPU-bound εργασία

Image processing, compression μεγάλου payload, PDF generation και machine learning inference μπορεί να κρατούν CPU. Περισσότερες coroutines δεν βοηθούν. Μικρό bounded CPU work ίσως πάει σε thread αν η native library ελευθερώνει το GIL. Βαρύτερη εργασία χρειάζεται process pool, ξεχωριστό worker service ή job system με admission control.

Μη δημιουργείς unbounded task για κάθε item ενός request. Χίλια items επί είκοσι requests μπορούν να ανοίξουν είκοσι χιλιάδες downstream calls. Βάλε semaphore ή bounded worker pool κοντά στο scarce resource και συνολικό request limit.

Sync stack είναι έγκυρη επιλογή

Αν όλο το stack είναι sync και το traffic μέτριο, plain `def` endpoints με μετρημένο thread pool μπορεί να είναι απλούστερα. Το async αξίζει όταν οι βιβλιοθήκες είναι πραγματικά async και η workload έχει

πολύ concurrent I/O. Μην ξαναγράψεις στα τυφλά. Μέτρα throughput, tail latency, pool waits και failure behavior.

Async database driver δεν επιτρέπει να μοιράζεσαι session σε child tasks. Κάθε task θέλει ανεξάρτητη unit of work ή sequential access. Παράλληλα queries στην ίδια database μπορεί απλώς να διπλασιάσουν connection pressure.

SOURCE TRACE

Η `run_endpoint_function` στο `fastapi/routing.py` κάνει `await` σε coroutine callable και χρησιμοποιεί `run_in_threadpool` για sync callable. Το ίδιο split εφαρμόζεται στα dependencies μέσα στο `solve_dependencies`. Το thread-pool implementation προέρχεται από Starlette και AnyIO.

ΠΑΓΙΔΑ

`async def` με `time.sleep()` ή synchronous SDK μπλοκάρει το event loop. Το annotation της function δεν αλλάζει τη φύση της εσωτερικής I/O.

ΜΗΧΑΝΙΣΜΟΣ

Κάθε blocking boundary πρέπει να είναι γνωστό. Κάθε concurrent fan-out πρέπει να είναι bounded. Κάθε deadline πρέπει να φτάνει μέχρι τον πραγματικό client ή driver.

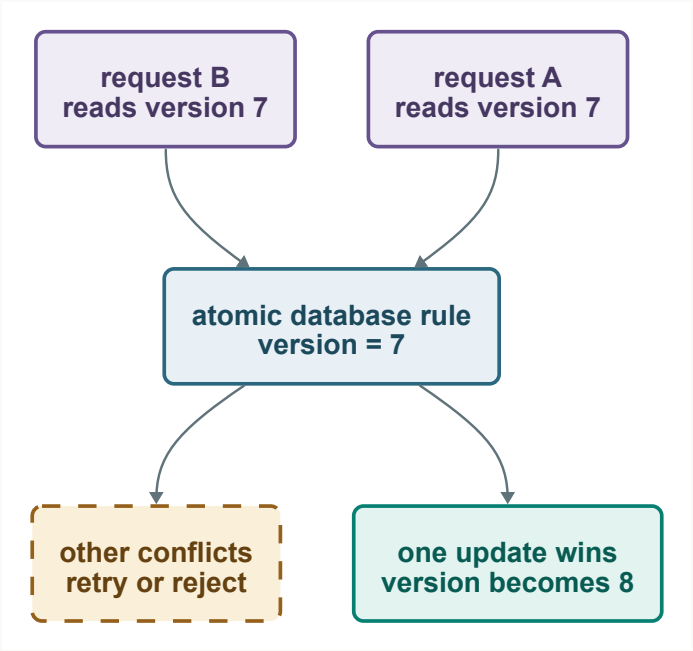
PROBE

Φτιάξε endpoint με `time.sleep(0.2)` μέσα σε `async def` και δεύτερο με `await anyio.sleep(0.2)`. Στείλε 20 concurrent requests και μέτρα total time. Μετά βάλε sync endpoint στο thread pool, περιόρισε τα tokens και παρατήρησε queueing αντί να κοιτάς μόνο average latency.

ΚΕΦΑΛΑΙΟ 21

Concurrency, shared state και race windows

Το event loop δεν εξαλείφει races. Ένα invariant μπορεί να αμφισβητηθεί από άλλη coroutine, thread, process, worker ή υπηρεσία ανάμεσα σε δύο βήματα.



Σχήμα 21.1 · Δύο requests διαβάζουν την ίδια version και η database επιλέγει έναν νικητή.

Στην async Python μία coroutine δεν διακόπτεται σε τυχαία bytecode instruction, αλλά παραχωρεί έλεγχο στα awaits. Αυτό αρκεί για race. Αν κάνεις read, περιμένεις I/O και μετά write, άλλο request μπορεί να αλλάξει το ίδιο state. Με sync endpoints προστίθενται threads. Με πολλούς Uvicorn workers προστίθενται processes με ανεξάρτητη μνήμη.

Το κλασικό bug είναι check-then-act:

```python
async def reserve_seat(repo, seat_id: str, user_id: str):
    seat = await repo.get(seat_id)
    if seat.reserved_by is not None:
        raise ValueError("seat_taken")
    await repo.reserve(seat_id, user_id)
    return await repo.get(seat_id)
```

Δύο callers μπορούν να δουν `None` πριν γράψει κάποιος. Python lock γύρω από τη function προστατεύει μόνο tasks που περνούν από το ίδιο lock object στον ίδιο process. Δεν προστατεύει δεύτερο worker, job consumer ή admin script.

Η ασφαλής operation μεταφέρει το precondition στη database:

```sql
UPDATE seats
SET reserved_by = :user_id, version = version + 1
WHERE id = :seat_id AND reserved_by IS NULL;
```

Αν affected rows είναι ένα, κέρδισες. Αν είναι μηδέν, το seat λείπει ή έχει ήδη κρατηθεί. Unique constraints, conditional updates και κατάλληλο isolation είναι cluster-wide arbiters επειδή όλοι οι writers περνούν από το ίδιο durable system.

Shared memory ανά process

Module globals, `app.state`, `lru_cache` και in-memory dictionaries είναι process-local. Κάθε worker έχει άλλο copy. Είναι χρήσιμα για immutable config, connection pools και performance cache όπου stale ή missing value δεν σπάει correctness. Δεν είναι σωστά για global quota, session truth ή distributed idempotency.

Μετά από fork, objects που άνοιξαν sockets πριν το fork μπορεί να έχουν προβληματικό shared file descriptor state. Άνοιξε pools μετά τη δημιουργία του worker μέσω lifespan, εκτός αν η συγκεκριμένη library τεκμηριώνει άλλο μοντέλο.

In-process cache χρειάζεται expiry, size bound και concurrency policy. Cache stampede συμβαίνει όταν πολλά requests βλέπουν expiry και ξαναφορτώνουν μαζί. Single-flight lock μειώνει τοπικό stampede, αλλά όχι across workers. Randomized TTL, stale-while-revalidate ή distributed coordination μπορεί να χρειαστούν ανάλογα με το κόστος.

Locks και awaits

Το `asyncio.Lock` ή AnyIO lock είναι για coroutines στο ίδιο loop. Μην κρατάς lock σε αργό external call. Αυξάνει tail latency και μπορεί να δημιουργήσει deadlock αν callback χρειάζεται τον ίδιο lock. Κράτα critical section μικρό και χωρίς απρόβλεπτο I/O.

Threading lock μέσα σε async function μπορεί να μπλοκάρει το event loop. Async lock δεν προστατεύει sync thread. Αν shared structure προσπελαύνεται και από τα δύο execution models, απλοποίησε ownership ή χρησιμοποίησε thread-safe primitive έξω από το event loop με μεγάλη προσοχή.

Isolation και write skew

Μία transaction δεν εγγυάται κάθε business invariant. Σε common isolation, δύο transactions μπορούν να διαβάσουν συμβατό snapshot και να γράψουν διαφορετικές rows, παραβιάζοντας cross-row rule. Προστάτεψε με constraint, advisory lock, serializable retry ή redesign που συγκεντρώνει το invariant σε μία row.

Serializable transaction μπορεί να αποτύχει νόμιμα και να χρειάζεται retry. Retry ολόκληρη την transaction με bounded attempts και μόνο αν τα external effects βρίσκονται έξω. Μην επαναλάβεις τυφλά provider charge επειδή η database ζήτησε serialization retry.

Lost updates και versions

Optimistic concurrency επιστρέφει version ή ETag στον client. Το update δίνει expected version και η database αλλάζει row μόνο αν συμφωνεί. Αυτό προστατεύει και από stale browser tab, όχι μόνο από requests που έφτασαν στο ίδιο millisecond.

Pessimistic row lock ταιριάζει σε σύντομες contention-heavy αποφάσεις. Θέλει σταθερή lock order, statement timeout και monitoring lock waits. Κανένα lock δεν πρέπει να κρατιέται όσο ο client διαβάζει stream ή ο provider περιμένει.

SOURCE TRACE

Το FastAPI εκτελεί async callables στο event loop και sync callables στο thread pool μέσω `run_endpoint_function` και `solve_dependencies` στο `fastapi/routing.py` και `fastapi/dependencies/utils.py`. Δεν προσθέτει application locks ή distributed synchronization. Αυτά ανήκουν στη data και workflow design.

ΠΑΓΙΔΑ

Ένα `asyncio.Lock` που «έλυσε» το test με έναν worker μπορεί να αποτύχει αμέσως όταν το deployment αυξηθεί σε δύο workers. Το correctness boundary πρέπει να καλύπτει όλους τους writers.

ΜΗΧΑΝΙΣΜΟΣ

Βρες το race window ανάμεσα σε read και write. Μετέφερε το precondition σε atomic durable operation και μετέφρασε καθαρά winner, conflict και retry.

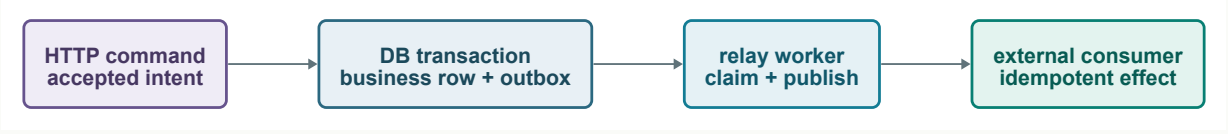
PROBE

Χρησιμοποίησε barrier ώστε 20 requests να ολοκληρώσουν το read πριν επιτραπεί το write. Τρέξε πρώτα με έναν worker και μετά με δύο. Assert ένα transition, 19 conflicts και σωστή τελική version. Επανάλαβε μετά από process restart.

ΚΕΦΑΛΑΙΟ 22

BackgroundTasks, queues και transactional outbox

Το «μετά το response» δεν σημαίνει durable. Η σημασία της εργασίας αποφασίζει αν αρκεί in-process task ή χρειάζεται transaction και queue.



Σχήμα 22.1 · Business row και outbox γράφονται μαζί πριν ο relay δημοσιεύσει το effect.

Το `BackgroundTasks` συλλέγει callables κατά το request και τα εκτελεί αφού σταλεί το response. Είναι βολικό για μικρή best-effort εργασία που μπορεί να χαθεί χωρίς business ζημιά, όπως hint για local cache cleanup. Τρέχει στον ίδιο process. Αν ο worker πέσει μετά το 202 ή 200, δεν υπάρχει durable record που να υποχρεώνει retry.

PYTHON

```
from fastapi import BackgroundTasks, FastAPI, status

app = FastAPI()


def refresh_local_hint(order_id: str) -> None:
    # Best effort only. Durable truth does not depend on this function.
    pass


@app.post("/orders/{order_id}/refresh", status_code=status.HTTP_202_ACCEPTED)
async def refresh_order(
    order_id: str,
    background: BackgroundTasks,
) -> dict[str, str]:
    background.add_task(refresh_local_hint, order_id)
    return {"state": "accepted"}
```

Ακόμη και εδώ, το 202 υπόσχεται κάποιο accepted workflow. Αν δεν υπάρχει τρόπος να δεις status και η task μπορεί να χαθεί, ίσως το 204 ή 200 με σαφή best-effort meaning είναι ειλικρινέστερο. Status code δεν μετατρέπει τη μνήμη σε queue.

Γιατί το απλό publish έχει δύο crash windows

Αν κάνεις database commit και μετά publish event, ο process μπορεί να πέσει ανάμεσα: το order υπάρχει αλλά event χάθηκε. Αν κάνεις publish και μετά commit, consumer μπορεί να δει event για

order που τελικά έγινε rollback. Distributed transaction συνήθως δεν είναι διαθέσιμη ή δεν αξίζει το coupling.

Το transactional outbox γράφει business αλλαγή και event row στην ίδια database transaction. Relay worker διαβάζει pending rows, κάνει publish και σημειώνει πρόοδο. Το write είναι atomic. Το publish είναι at-least-once, επειδή worker μπορεί να δημοσιεύσει και να πέσει πριν το mark. Ο consumer πρέπει να είναι idempotent.

Outbox row κρατά event id, aggregate identity, type, schema version, payload, created time και attempt state. Μην αποθηκεύεις ολόκληρο ORM object ή secret. Το payload είναι deployed contract. Χρειάζεται compatibility policy όπως κάθε άλλο API.

Claiming και retries

Πολλοί relay workers πρέπει να διεκδικούν rows atomically, για παράδειγμα με `FOR UPDATE SKIP LOCKED` και μικρές transactions. Μην κρατάς row lock κατά το network publish. Claim με lease, commit, publish και μετά mark. Αν lease λήξει, άλλος worker μπορεί να ξαναδοκιμάσει.

Retry μόνο classified transient failures με exponential backoff, jitter και maximum attempts ή age. Permanent invalid payload πηγαίνει σε dead-letter state με alert και repair tooling. Infinite hot retry μπορεί να γονατίσει τον provider και να κρύψει νέο event πίσω από poison message.

Ordering χρειάζεται συγκεκριμένο scope. Global ordering είναι ακριβό. Συνήθως αρκεί σειρά ανά aggregate key. Partition key, claim algorithm και consumer dedupe πρέπει να συμφωνούν. Αν δύο διαφορετικά event types μπορούν να έρθουν ανάποδα, ο consumer χρειάζεται version check.

Background task failures

Exception σε background task συμβαίνει αφού έχει φύγει το response. Exception handler δεν μπορεί να αλλάξει τον client outcome. Χρειάζεσαι log, metric και ίσως error tracker. Επειδή οι tasks τρέχουν σειριακά στο Starlette background collection, failure μπορεί να εμποδίσει επόμενες tasks. Μην στοιβάζεις critical workflow σε μία λίστα callbacks.

Το `asyncio.create_task` μέσα σε endpoint είναι ακόμη πιο αδύναμο αν δεν κρατάς handle. Το task μπορεί να ακυρωθεί στο shutdown, να χάσει exception ή να συνεχίσει μετά το request χωρίς context owner. Long-lived consumers ξεκινούν σε lifespan με task group και bounded shutdown. Durable jobs πάνε σε queue.

Exactly-once είναι σύνθεση invariants

Broker delivery σπάνια είναι exactly-once από μόνο του. Business exactly-once effect χτίζεται με stable event id, consumer inbox ή unique constraint και atomic state transition. Αν ο external provider δέχεται idempotency key, πέρασε το event id. Αν δεν δέχεται, χρειάζεται lookup ή reconciliation για ambiguous timeouts.

Metrics που αξίζουν: outbox age, pending count, claim lease expiry, attempts, dead letters και end-to-end time μέχρι consumer application. Queue depth μόνο δεν δείχνει αν ένα παλιό event έχει κολλήσει.

> **SOURCE TRACE**
> Το `fastapi/background.py` εκθέτει το Starlette `BackgroundTasks` με FastAPI documentation. Στο `fastapi/routing.py`, τα solved dependencies μπορούν να συγκεντρώσουν background tasks και να τις προσαρτήσουν στο Response. Η durable queue και το outbox είναι application infrastructure, όχι FastAPI feature.

ΠΑΓΙΔΑ

Email, payment ή webhook που υπάρχει μόνο σε `BackgroundTasks` μπορεί να χαθεί με process crash αφού ο client έχει ήδη πάρει success response.

ΜΗΧΑΝΙΣΜΟΣ

Best-effort callback μπορεί να μείνει in-process. Business obligation γράφεται durable μαζί με την state change και εκτελείται idempotently από worker.

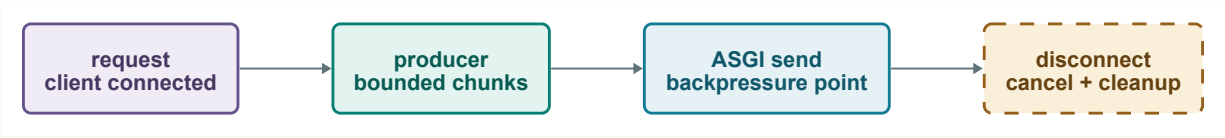
PROBE

Βάλε controlled crash μετά το database commit, μετά το publish και πριν το outbox mark. Επανεκκίνησε worker και assert ότι κανένα event δεν χάνεται, ο consumer δεν εφαρμόζει δύο φορές το effect και το outbox age επιστρέφει στο normal.

ΚΕΦΑΛΑΙΟ 23

Uploads, downloads και streaming

Τα μεγάλα bodies αλλάζουν το resource model. Bytes, προσωρινά files, open handles, αργοί clients και disconnects χρειάζονται όρια και cleanup.



Σχήμα 23.1 · Producer, ASGI send και disconnect σχηματίζουν το stream lifecycle.

Για multipart upload, το `UploadFile` δίνει filename, content type και file-like object που μπορεί να γίνει spooled από μνήμη σε προσωρινό file. Αυτό είναι καλύτερο από `bytes` για μεγάλα αρχεία, επειδή δεν απαιτεί να κρατήσεις όλο το payload ως ένα Python bytes object. Το multipart parsing απαιτεί το optional `python-multipart` dependency.

PYTHON

```
from hashlib import sha256

from fastapi import FastAPI, HTTPException, UploadFile

app = FastAPI()


@app.post("/documents")
async def upload_document(file: UploadFile) -> dict[str, object]:
    digest = sha256()
    total = 0
    while chunk := await file.read(64 * 1024):
        total += len(chunk)
        if total > 10 * 1024 * 1024:
            raise HTTPException(status_code=413, detail="file_too_large")
        digest.update(chunk)
    await file.close()
    return {
        "bytes": total,
        "sha256": digest.hexdigest(),
        "media_type": file.content_type,
    }
```

Το endpoint δείχνει incremental read, όχι πλήρες storage workflow. Βάλε close σε `finally` αν υπάρχει κώδικας μετά το read που μπορεί να αποτύχει. Ακόμη καλύτερα, βασίσου στο documented request lifecycle αλλά κράτα δικό σου cleanup για handles που ανοίγεις εσύ.

Το filename και content type έρχονται από client και δεν είναι αξιόπιστα. Μην χρησιμοποιείς raw filename ως filesystem path. Δημιούργησε server-side id, αποθήκευσε display name χωριστά και κάνε safe escaping όταν το επιστρέφεις σε `Content-Disposition`. Αν επιτρέπεις συγκεκριμένους file types, έλεγξε magic bytes και πραγματικό parser σε sandboxed ή isolated processing path.

Size limits σε περισσότερα από ένα layers

Το loop κόβει αφού bytes έχουν ήδη φτάσει στον application process και ο multipart parser μπορεί να έχει γράψει προσωρινό data. Βάλε request body limit στο reverse proxy ή server για πρώτο φράγμα. Κράτα application limit για business policy. Ρύθμισε disk quota και cleanup για προσωρινά files.

Decompression bombs και parser complexity δεν μετριούνται μόνο σε compressed upload bytes. Image dimensions, archive entry count και expanded size χρειάζονται δικά τους bounds. Βαριά scanning ή conversion πάει σε job worker με resource limits, όχι στο request event loop.

Downloads

Για πραγματικό file στο disk, το Starlette `FileResponse` χειρίζεται metadata και efficient sending καλύτερα από χειροποίητο loop. Για object storage, συχνά είναι καλύτερο short-lived signed URL ώστε τα bytes να μη διέρχονται από web workers. Η authorization πρέπει να γίνει πριν εκδοθεί το URL και η expiry να είναι μικρή.

Private content θέλει `Cache-Control: private` ή `no-store` ανά use case. Filename, media type, ETag και range behavior είναι public protocol. Αν proxy κάνει caching, βεβαιώσου ότι cache key περιλαμβάνει όσα διαφοροποιούν authorized representation ή απενεργοποίησε shared caching.

StreamingResponse

Το `StreamingResponse` δέχεται async iterable ή sync iterator. Κάθε yielded chunk γίνεται ASGI body event. Async producer πρέπει να φτάνει συχνά σε await ώστε να παρατηρεί backpressure και cancellation. Sync iterator περνά από thread pool. Μην παράγεις άπειρα chunks γρηγορότερα από το network.

Response headers και status στέλνονται πριν ή μαζί με την έναρξη. Exception στο δέκατο chunk δεν μπορεί να γίνει κανονικό JSON 500. Για newline-delimited JSON ή SSE, το protocol μπορεί να ορίζει terminal error event, αλλά partial data έχει ήδη καταναλωθεί. Consumers πρέπει να ξέρουν αν επιτρέπεται retry και από ποιο offset.

Στην έκδοση αναφοράς το FastAPI διαθέτει και SSE support στο `fastapi/sse.py`, πάνω στο ASGI response model. SSE χρειάζεται event ids, heartbeat, reconnect policy και bounded per-client buffer. Δεν είναι durable message queue.

Disconnect και cleanup

Client μπορεί να φύγει ενώ producer διαβάζει database ή provider. Cancellation πρέπει να κλείσει cursor, file και child tasks. Μην καταπίνεις cancellation με broad handler. Αν operation έκανε durable command πριν ξεκινήσει το download, το disconnect δεν το αναιρεί αυτόματα.

Resource dependency με request scope μένει ανοιχτή μέχρι να τελειώσει το response. Αυτό επιτρέπει lazy stream, αλλά κρατά pool slot για τον αργό client. Προτίμησε stream-specific resource με capacity limit ή precomputed object storage artifact.

> **SOURCE TRACE**
> Το FastAPI επανεξάγει `UploadFile`, `FileResponse` και `StreamingResponse` από τα Starlette-based layers, ενώ body field handling βρίσκεται στο `fastapi/routing.py` και `fastapi/dependencies/utils.py`. Το SSE response της έκδοσης αναφοράς βρίσκεται στο `fastapi/sse.py`. Η αποστολή γίνεται τελικά με ASGI body events.

ΠΑΓΙΔΑ
Το client-supplied filename μπορεί να περιέχει path traversal ή control characters. Μην το χρησιμοποιήσεις ποτέ ως trusted local path.

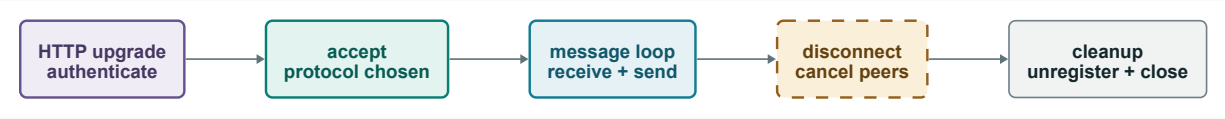
ΜΗΧΑΝΙΣΜΟΣ
Βάλε limits πριν, κατά και μετά το parsing. Κάνε streaming με bounded chunks, backpressure και cancellation cleanup. Μετά το response start, το error protocol πρέπει να είναι stream-aware.

PROBE
Στείλε upload ακριβώς στο limit, ένα byte πάνω, αργό upload και disconnect στη μέση. Για download βάλε client που διαβάζει ένα chunk ανά δευτερόλεπτο. Μέτρα open files, temp disk, pool usage και χρόνο cleanup μετά το disconnect.

ΚΕΦΑΛΑΙΟ 24

WebSockets, lifespan και disconnects

Ένα WebSocket είναι μακρόβια συνομιλία, όχι endpoint που επιστρέφει μία φορά. Authentication, backpressure, task ownership και shutdown γίνονται protocol.



Σχήμα 24.1 · Handshake, message loop, disconnect και cleanup έχουν συγκεκριμένη σειρά.

Ο client αρχίζει με HTTP upgrade. Η εφαρμογή μπορεί να ελέγξει credentials και να δεχτεί ή να αρνηθεί τη σύνδεση. Μετά το `accept`, status codes δεν είναι το κύριο error mechanism. Υπάρχουν WebSocket messages και close codes. Μην σηκώνεις τυχαία HTTP exception βαθιά μέσα στο accepted message loop.

PYTHON

```
from fastapi import FastAPI, WebSocket, WebSocketDisconnect

app = FastAPI()


@app.websocket("/ws/orders/{order_id}")
async def order_updates(websocket: WebSocket, order_id: str) -> None:
    token = websocket.headers.get("Authorization")
    if token != "Bearer verified-demo-token":
        await websocket.close(code=1008)
        return

    await websocket.accept(subprotocol="orders.v1")
    try:
        while True:
            message = await websocket.receive_json()
            if message.get("type") == "ping":
                await websocket.send_json(
                    {"type": "pong", "order_id": order_id}
                )
    except WebSocketDisconnect:
        pass
```

Το token είναι demo. Σε browser δεν μπορείς πάντα να ορίσεις arbitrary Authorization header στο WebSocket constructor. Cookie ή negotiated short-lived ticket είναι συνηθισμένες επιλογές. Απόφυγε long-lived access token στο query string επειδή URLs καταγράφονται σε proxies, analytics και history.

Το subprotocol version είναι σαφέστερο από implicit message evolution. Ο server πρέπει να διαλέξει μόνο προσφερόμενο και υποστηριζόμενο subprotocol. Κάθε message χρειάζεται discriminator, schema validation, maximum bytes και authorization για τη συγκεκριμένη action.

Connection registry

Ένα set από sockets στο `app.state` βλέπει μόνο connections του ίδιου worker. Είναι αρκετό για local demo, όχι για cluster broadcast. Με πολλούς workers, publisher γράφει σε broker ή pub-sub και κάθε worker προωθεί στους δικούς του local subscribers.

Registry mutation χρειάζεται lock ή single-owner task. Remove πρέπει να είναι idempotent επειδή disconnect, exception και shutdown μπορεί να φτάσουν από διαφορετικά paths. Μην κρατάς strong references για πάντα μετά από broken connection.

Presence δεν είναι απόλυτη αλήθεια. Heartbeat, network partition και mobile suspend δημιουργούν delay. Αν business rule απαιτεί durable online state, όρισε lease και expiry αντί για boolean που αλλάζει μόνο σε clean disconnect.

Slow consumers και backpressure

Αν ένα broadcast loop κάνει `await send` διαδοχικά, ένας αργός client καθυστερεί όλους. Αν δημιουργεί unbounded task ανά message, η μνήμη αυξάνει. Δώσε bounded queue ανά connection και policy: drop old transient update, disconnect slow consumer ή block συγκεκριμένο producer. Η σωστή επιλογή εξαρτάται από το αν τα events είναι snapshots ή non-repeatable commands.

Για σημαντικά events, ο client χρειάζεται sequence number ή durable cursor και HTTP resync endpoint. WebSocket delivery μόνο του δεν εγγυάται ότι κάθε message έφτασε ή εφαρμόστηκε. Reconnect μπορεί να προκαλέσει gaps και duplicates.

Task ownership

Συχνά η σύνδεση έχει δύο concurrent εργασίες: receive client messages και forward broker events. Βάλ' τες σε task group. Αν μία τελειώσει ή αποτύχει, ακύρωσε την άλλη και περίμενε cleanup. Ούτε orphan broker subscription ούτε task που στέλνει σε κλειστό socket πρέπει να επιβιώνει.

Το lifespan κατέχει shared broker connection και top-level consumers. Κάθε WebSocket handler κατέχει τη subscription του. Στο shutdown, σταμάτα admission, στείλε service-restart close code αν ταιριάζει, δώσε bounded drain time και κλείσε broker μετά τις connections.

Deployments και stickiness

Rolling deploy κόβει μακρόβιες connections όταν παλιός worker φύγει. Οι clients πρέπει να κάνουν reconnect με jitter και resync. Load-balancer stickiness μπορεί να βοηθά local session state, αλλά δεν αντικαθιστά shared truth και δυσκολεύει rebalance. Σχεδίασε protocol ώστε connection να μπορεί να ανοίξει σε οποιονδήποτε healthy worker.

Metrics: active connections ανά worker, handshake rejects, close codes, queue depth, dropped messages, reconnect rate και message lag. User id δεν είναι metric label. Χρησιμοποίησε safe logs για συγκεκριμένη διάγνωση.

> **SOURCE TRACE**
> Το `FastAPI.websocket` καταχωρίζει `APIWebSocketRoute`. Το `websocket_session` στο `fastapi/routing.py` δημιουργεί Starlette `WebSocket`, στήνει dependency `AsyncExitStack` και τυλίγει exception handling. Τα δημόσια WebSocket objects και disconnect exceptions προέρχονται από Starlette.

ΠΑΓΙΔΑ

In-memory connection manager δεν μπορεί να κάνει broadcast σε sockets άλλου worker. Το bug εμφανίζεται συχνά μόνο μετά το πρώτο horizontal scale-out.

ΜΗΧΑΝΙΣΜΟΣ

WebSocket correctness χρειάζεται versioned messages, bounded queues, reconnect και resync. Lifespan κατέχει shared infrastructure, κάθε handler τη δική του connection και tasks.

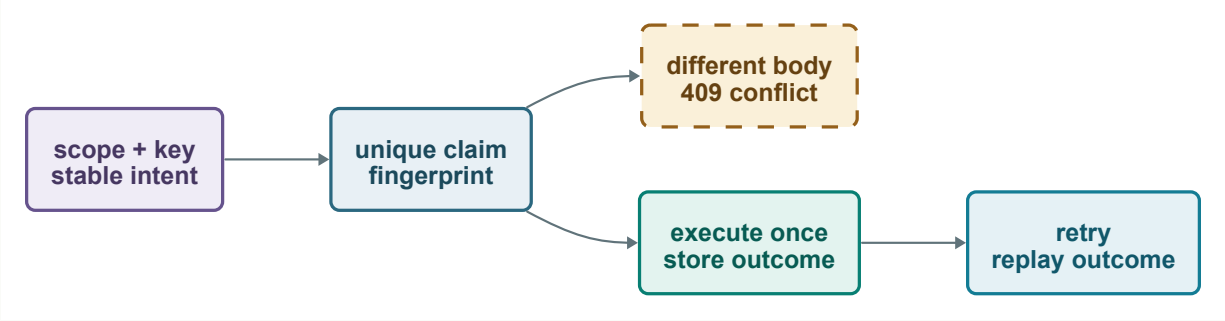
PROBE

Άνοιξε γρήγορο και αργό client, σταμάτα τον ένα χωρίς clean close και κάνε restart worker ενώ στέλνονται events. Assert bounded queue, cleanup registry, close behavior και resync από τελευταίο sequence. Επανάλαβε με δύο workers.

ΚΕΦΑΛΑΙΟ 25

Caching, idempotency και versioning

Cache validator αναγνωρίζει representation. Idempotency key αναγνωρίζει intent. Version αναγνωρίζει state που είδε ο caller. Μην τα συγχέεις.



Σχήμα 25.1 · Stable key και fingerprint οδηγούν σε execute, replay ή conflict.

HTTP caching μπορεί να αφαιρέσει δουλειά από το FastAPI process. Ένα ETag αναγνωρίζει συγκεκριμένη representation. Ο client στέλνει `If-None-Match` και ο server επιστρέφει 304 χωρίς body αν δεν άλλαξε. Για private user-specific data, το `Cache-Control` πρέπει να εμποδίζει shared cache ή να ορίζει σωστό key.

PYTHON

```
from typing import Annotated

from fastapi import FastAPI, Header, Response
from fastapi.encoders import jsonable_encoder
from fastapi.responses import JSONResponse
from pydantic import BaseModel

app = FastAPI()


class OrderView(BaseModel):
    id: str
    state: str
    version: int


@app.get("/orders/{order_id}", response_model=OrderView)
async def get_order(
    order_id: str,
    if_none_match: Annotated[
        str | None, Header(alias="If-None-Match")
    ] = None,
) -> Response:
    view = OrderView(id=order_id, state="pending", version=7)
    etag = f'"order-{view.version}"'
    if if_none_match == etag:
        return Response(status_code=304, headers={"ETag": etag})
```

```
    return JSONResponse(
        content=jsonable_encoder(view),
        headers={"ETag": etag, "Cache-Control": "private, max-age=0"},
    )
```

Το direct response παρακάμπτει runtime response-model validation, άρα σε πραγματικό code κάνε explicit model construction όπως εδώ. Πλήρης parser για conditional headers πρέπει να χειρίζεται πολλαπλά tags και wildcard αν το contract τα υποστηρίζει.

Optimistic update

Για mutation, ο client στέλνει το ETag που διάβασε σε `If-Match`. Η database κάνει atomic update μόνο στην expected version. Αν άλλαξε, 412 Precondition Failed είναι φυσική HTTP απάντηση. Ένα business conflict που δεν σχετίζεται με request precondition μπορεί να είναι 409.

Μην ελέγχεις το ETag σε Python και μετά κάνεις unconditional update. Υπάρχει race ανάμεσα. Το version comparison πρέπει να είναι μέρος της ίδιας SQL operation ή lock-protected transaction.

Idempotency για commands

Client μπορεί να χάσει response μετά την εκτέλεση POST και να ξαναστείλει. Idempotency key πρέπει να έχει scope, συνήθως tenant και caller, bounded grammar και retention window. Ο server υπολογίζει fingerprint από canonical accepted command fields. Ίδιο key με ίδιο fingerprint κάνει replay. Ίδιο key με άλλο fingerprint επιστρέφει 409.

Durable table έχει unique `(scope, key)`, fingerprint, state, outcome reference, created time και expiry. Πρώτος caller κάνει claim. Concurrent caller διαβάζει processing ή completed state. Το read-then-insert χωρίς unique constraint έχει race. Η database αποφασίζει ποιος κέρδισε.

Αποθήκευσε domain outcome αντί για τυχαία raw response bytes, εκτός αν exact byte replay είναι contract. Έτσι μπορείς να παρουσιάσεις outcome με συμβατή serializer version. Αν το response schema αλλάξει μέσα στο retention window, κράτα version ώστε ο replay να είναι προβλέψιμος.

Unknown outcomes

Αν business write και idempotency row είναι στην ίδια database transaction, η local ακριβώς-μία state change είναι εφικτή. External effect δημιουργεί ambiguity. Πέρασε το ίδιο stable key στον provider αν το υποστηρίζει ή κάνε lookup/reconciliation. Μετά από read timeout δεν ξέρεις αν ο provider δέχτηκε το request.

Stuck `processing` claim χρειάζεται lease ή recovery state. Μην το θεωρείς μόνιμα busy. Worker ή request μπορεί να πέσει. Recovery εξετάζει durable business state και provider receipt πριν αποφασίσει retry.

API και event versioning

Public path version, schema version, resource version και event sequence είναι διαφορετικά. Το `/v2` δεν αντικαθιστά row version. Ένα `event_version` δεν είναι idempotency key. Ονόμασε κάθε έννοια συγκεκριμένα.

Για backward compatibility, πρώτα πρόσθεσε optional input/output, deploy code που αντέχει παλιό και νέο schema, κάνε backfill αν χρειάζεται και αφαίρεσε μόνο όταν όλοι οι readers έχουν μετακινηθεί. Σε rolling release, τουλάχιστον δύο application versions συνυπάρχουν.

Cache invalidation και truth

Cache είναι derived state. Write path ενημερώνει database truth και μετά invalidates ή αλλάζει versioned key. Αν invalidation χαθεί, TTL περιορίζει τη ζημιά. Για correctness-critical read, version token ή direct database read μπορεί να απαιτείται. Μην χρησιμοποιείς cache lock ως μοναδικό inventory guard.

SOURCE TRACE

Το FastAPI παρέχει typed `Header`, `Response` και serialization tools, αλλά δεν υλοποιεί application caching ή idempotency store. Η route pipeline στο `fastapi/routing.py` περνά response headers και direct Response όπως δηλώσει η εφαρμογή. Τα HTTP semantics και durable claims είναι δικό σου contract.

ΠΑΓΙΔΑ

Ίδιο idempotency key με διαφορετικό body δεν είναι replay. Αν επιστρέψεις την παλιά επιτυχία, ο caller μπορεί να πιστέψει ότι εκτελέστηκε νέο amount ή target.

ΜΗΧΑΝΙΣΜΟΣ

ETag προστατεύει representation και stale writes. Idempotency identity είναι `scope + key + fingerprint`. Και τα δύο χρειάζονται durable ή protocol-wide owner, όχι in-memory dict.

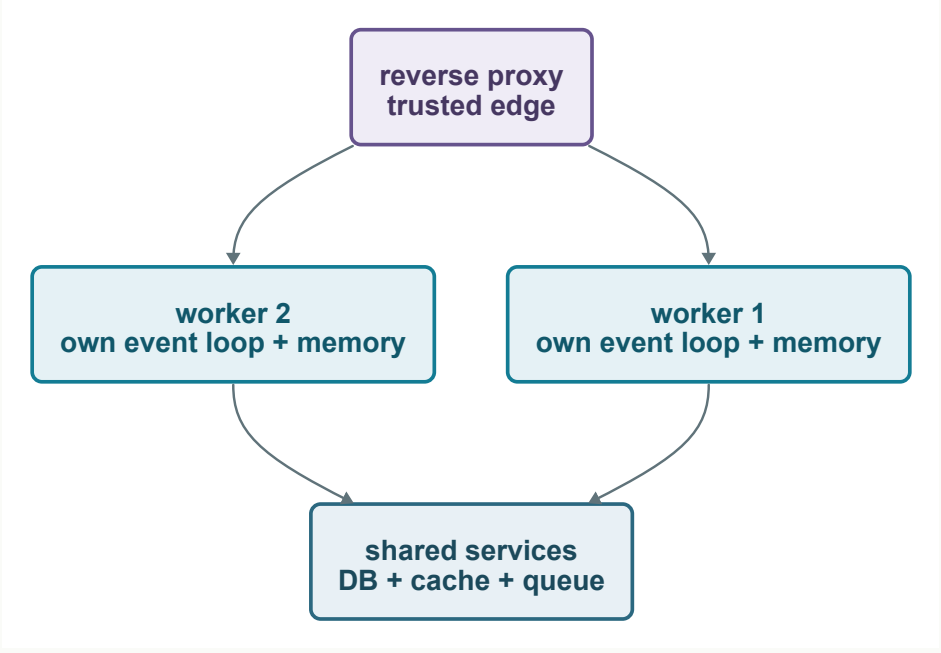
PROBE

Στείλε 20 concurrent POSTs με ίδιο key και body, μετά ίδιο key με άλλο body. Βάλε crash μετά το claim και μετά το business commit. Για update, στείλε δύο `If-Match` με ίδια version. Assert ένα transition, stable replay, conflict και bounded recovery.

ΚΕΦΑΛΑΙΟ 26

Uvicorn, workers, proxies και graceful shutdown

Production FastAPI είναι topology από proxy, ASGI workers και shared services. Κάθε process έχει δικό του event loop, memory και resource budget.



Σχήμα 26.1 · Ο proxy μοιράζει requests σε ανεξάρτητους workers που μοιράζονται μόνο durable services.

Ο Uvicorn είναι ASGI server. Ανοίγει sockets, μεταφράζει HTTP και WebSocket σε ASGI events και κινεί το lifecycle της εφαρμογής. Το FastAPI app είναι το callable που φορτώνει. Ένα απλό production command μπορεί να μοιάζει με:

TERMINAL

```
uvicorn app.main:app --host 0.0.0.0 --port 8000 --workers 4
```

Το `4` δεν είναι universal σωστή τιμή. Με container orchestrator μπορεί να τρέχει ένας worker ανά container και horizontal replicas. Σε μία VM μπορεί να χρησιμοποιούνται πολλοί workers. Η επιλογή εξαρτάται από CPU quota, memory, blocking work, connection budgets και failure isolation.

Το `--reload` είναι development feature. Παρακολουθεί files και επανεκκινεί process. Δεν είναι production deployment strategy. Ο κώδικας και τα resources πρέπει να ξεκινούν deterministic χωρίς να εξαρτώνται από state του reloader.

Process model

Κάθε worker κάνει import το app και τρέχει δικό του lifespan. Άρα έχει δικό του database pool, HTTP client, cache, locks και connection registry. Τέσσερις workers με pool size 15 μπορούν να ανοίξουν 60 database connections ανά replica. Πολλαπλασίασε και με replicas, job workers, admin tools και migration process.

Process-local cache μπορεί να έχει διαφορετικές values. Process-local rate limiter εφαρμόζει όριο ανά worker, όχι ανά account σε όλο το cluster. Το ίδιο ισχύει για semaphore και WebSocket rooms. Για correctness ή global policy χρειάζεται shared durable owner.

Workers δεν αυξάνονται μόνο με CPU count. Async I/O worker μπορεί να κρατήσει πολλές concurrent connections, ενώ CPU-heavy Python work χρειάζεται processes ή ξεχωριστή υπηρεσία. Πολλοί workers αυξάνουν memory και downstream pressure. Μετράς throughput και tail latency με πραγματικό quota.

Reverse proxy και forwarded trust

Συνήθως TLS τερματίζει σε load balancer ή reverse proxy. Η εφαρμογή βλέπει εσωτερική connection και χρειάζεται forwarded scheme, host και client address για redirects, URL generation και audit. Αυτά τα headers είναι forgeable αν ο client φτάνει απευθείας στον server ή ο server εμπιστεύεται οποιαδήποτε IP.

Ρύθμισε proxy header support και `forwarded-allow-ips` μόνο για πραγματικούς trusted proxies. Κλείσε direct public πρόσβαση στο app port. Αν η εφαρμογή σερβίρεται κάτω από path prefix, ρύθμισε ASGI `root_path` και δοκίμασε docs, redirects και generated URLs μέσω του πραγματικού proxy.

Trusted host validation είναι ξεχωριστό. Το forwarded client IP δεν πρέπει να χρησιμοποιείται σαν authentication identity. NAT, chains και proxy rewrites κάνουν αυτή την τιμή contextual signal.

Admission και protocol limits

Ο server μπορεί να βάλει όριο concurrent connections ή tasks, keep-alive timeout και maximum requests ανά worker. Ο edge βάζει request body, header και idle time limits. Η εφαρμογή βάζει domain quotas και downstream semaphores. Τα layers συνεργάζονται. Αν το proxy περιμένει 60 δευτερόλεπτα αλλά η εφαρμογή έχει 5 δευτερόλεπτα deadline, το failure είναι γρήγορο και ελεγχόμενο.

Keep-alive μειώνει handshake cost αλλά κρατά connections. WebSockets και SSE είναι πολύ πιο μακρόβια. Load balancer idle timeout, heartbeat και application protocol πρέπει να συμφωνούν. Ένα proxy που κόβει στα 30 δευτερόλεπτα ακυρώνει οποιαδήποτε heartbeat ανά 60.

Graceful shutdown

Σε deploy, πρώτα το instance πρέπει να βγει από readiness ώστε να σταματήσει νέο traffic. Μετά ο server λαμβάνει termination, σταματά admission και περιμένει bounded χρόνο για in-flight requests. Το lifespan cleanup κλείνει consumers, clients και pools. Τέλος ο orchestrator σκοτώνει ό,τι ξεπέρασε grace period.

Το grace period πρέπει να είναι μεγαλύτερο από το αναμενόμενο application drain αλλά όχι άπειρο. Requests έχουν deadlines μικρότερα από αυτό. Background durable jobs επιστρέφουν στην queue αν ο worker φύγει. In-process tasks μπορεί να χαθούν.

Για WebSockets, στείλε close και άφησε clients να reconnect με jitter. Για streaming downloads, αποφάσισε αν τα αφήνεις να τελειώσουν ή αν ο client έχει range/resume protocol. Long request χωρίς idempotency κάνει shutdown δύσκολο.

Health probes

Startup probe δίνει χρόνο στο process να φορτώσει. Readiness δηλώνει admission. Liveness εντοπίζει κολλημένο process χωρίς να εξαρτάται από κάθε downstream. Αν η database πέσει, restart όλων των workers μπορεί να χειροτερέψει την κατάσταση με connection storm. Προτίμησε not-ready ή controlled 503 όταν εξαρτάται από το failure model.

SOURCE TRACE

Το FastAPI app υλοποιεί ASGI μέσω Starlette-compatible stack και lifespan. Ο Uvicorn κατέχει worker processes, sockets, protocol parsing, proxy header trust και graceful server shutdown. Το `fastapi/cli.py` παρέχει convenience commands, αλλά τα production semantics παραμένουν του ASGI server και της πλατφόρμας.

ΠΑΓΙΔΑ

Το `--proxy-headers` χωρίς περιορισμένο trusted proxy boundary επιτρέπει σε client να πλαστογραφήσει scheme ή IP που μπορεί να μπει σε redirects, logs και security policy.

ΜΗΧΑΝΙΣΜΟΣ

Υπολόγισε κάθε pool και limiter ανά process επί συνολικούς workers. Κάνε trust μόνο τον γνωστό edge και σχεδίασε termination σαν κανονικό runtime event.

PROBE

Τρέξε δύο workers και endpoint που επιστρέφει process id και local counter. Δες την ανεξάρτητη μνήμη. Στείλε SIGTERM με ένα αργό request και WebSocket ανοιχτό. Μέτρα πότε κόβεται readiness, πότε σταματά admission και αν το cleanup τελειώνει πριν το hard kill.

ΚΕΦΑΛΑΙΟ 27

Capacity, pools και backpressure

Όταν η άφιξη εργασίας ξεπερνά την ικανότητα εξυπηρέτησης, έχεις δύο επιλογές: bounded αναμονή ή έγκαιρη απόρριψη. Η unbounded ουρά απλώς κρύβει την κατάρρευση.

Σχήμα 27.1 · Admission, workers και downstream pools μοιράζονται ένα latency budget.

Ο Little's Law δίνει χρήσιμο πρώτο estimate: average concurrency ισούται με arrival rate επί average time in system. Στα 200 requests ανά δευτερόλεπτο και 0,1 δευτερόλεπτο μέσο χρόνο έχεις περίπου 20 concurrent requests. Στο tail των 2 δευτερολέπτων όμως μια μικρή αιχμή μπορεί να δεσμεύσει πολύ περισσότερα resources. Τα averages δεν σχεδιάζουν μόνα τους capacity.

Το request περνά από πολλές ουρές: load balancer, server admission, thread pool, database pool, provider pool και ίσως lock. Αν κάθε layer περιμένει χωρίς deadline, το total latency γίνεται άθροισμα κρυφών waits. Μέτρα queue wait ξεχωριστά από execution time.

Bounded concurrency κοντά στο scarce resource

Αν provider αντέχει 20 concurrent calls ανά worker, βάλε limiter γύρω από τον adapter και μικρό acquisition timeout. Το limit πρέπει να προκύπτει από deployment-wide budget, όχι τυχαίο constant.

PYTHON

```
from anyio import CapacityLimiter, fail_after


class Provider:
    def __init__(self, client, concurrency: int = 20) -> None:
        self.client = client
        self.limiter = CapacityLimiter(concurrency)

    async def quote(self, payload: dict[str, object]) -> dict[str, object]:
        with fail_after(1.0):
            async with self.limiter:
                response = await self.client.post("/quotes", json=payload)
                response.raise_for_status()
                return response.json()
```

Αυτός ο limiter είναι process-local. Τέσσερις workers επιτρέπουν έως 80 calls. Αν provider quota είναι global 50, μοίρασε budget ή χρησιμοποίησε edge/shared admission. Distributed limiter έχει δικό του availability και consistency tradeoff. Συχνά αρκεί conservative per-worker όριο και provider 429 handling.

Database pool ως queue

Μεγάλο pool δεν κάνει αργή database γρήγορη. Επιτρέπει περισσότερα concurrent queries που μπορεί να αυξήσουν contention. Μικρό pool δημιουργεί wait στο app. Θέλεις bounded checkout timeout, metrics για wait και αρκετό headroom για jobs και maintenance.

Pool sizing ξεκινά από database capacity και μοιράζεται σε όλα τα processes. Αν autoscaler διπλασιάσει replicas, doubles connection demand πριν προλάβει η database να μεγαλώσει. Βάλε maximum replicas συμβατό με connection budget ή proxy/pooler με κατανοητά transaction semantics.

Load shedding

Όταν δεν μπορείς να εξυπηρετήσεις μέσα στο latency SLO, γρήγορο 503 με `Retry-After` μπορεί να είναι καλύτερο από timeout μετά από 30 δευτερόλεπτα. Το 429 είναι για rate ή quota policy του caller. Το 503 είναι προσωρινή ανικανότητα υπηρεσίας. Και τα δύο χρειάζονται client backoff με jitter.

Μην απορρίπτεις health και cancellation control πίσω από την ίδια saturated queue. Κράτα operational endpoints φθηνά. Προτεραιότητες χρειάζονται προσοχή για να μη λιμοκτονούν κανονικά requests. Η απλούστερη λύση είναι μικρότερα deadlines και separate worker pool για βαριά jobs.

Fan-out multiplication

Ένα incoming request που κάνει πέντε παράλληλα provider calls μετατρέπει 100 concurrent requests σε έως 500 downstream operations. Cache misses, retries και hedged requests αυξάνουν τον multiplier. Γράψε capacity equation για κάθε endpoint: maximum fan-out, attempts, pool slots και bytes.

Retries πρέπει να καταναλώνουν retry budget, όχι να διπλασιάζουν traffic κατά outage. Περιορισμός ανά caller, circuit state και global attempt rate μπορούν να προστατεύσουν provider. Circuit breaker δεν είναι απλό boolean. Θέλει half-open probes, classification και observability.

Memory και response size

Concurrency επί per-request memory δίνει peak pressure. Αν κάθε JSON response χτίζει 20 MB buffer και έχεις 100 concurrent requests, μόνο αυτά πλησιάζουν 2 GB πριν από runtime και pools. Pagination, streaming και output bounds είναι capacity tools. Streaming μειώνει buffering αλλά κρατά connection και resources περισσότερο.

Garbage collection pause, allocator fragmentation και Pydantic model count φαίνονται μόνο με representative payloads. FastAPI releases βελτιώνουν συχνά route-build ή dependency memory, αλλά application data shape παραμένει ο μεγαλύτερος μοχλός σε πολλά συστήματα.

Load test που απαντά ερώτηση

Warm up workers, χρησιμοποίησε production-like CPU/memory quota και ελεγχόμενο downstream. Αύξησε arrival rate, όχι απλώς fixed concurrency. Μέτρα p50, p95, p99, errors, pool waits, event-loop lag, CPU και memory. Βρες saturation knee, μετά όρισε safe operating point αρκετά χαμηλότερα.

Load generator δεν πρέπει να βρίσκεται στον ίδιο μικρό host. Τα test δεδομένα πρέπει να προκαλούν πραγματικές query shapes. Ένα `/health` benchmark δεν λέει capacity του checkout workflow.

SOURCE TRACE

Το FastAPI χειρίζεται request graph και validation, αλλά δεν ορίζει production admission policy. Sync callables χρησιμοποιούν Starlette/AnyIO thread-pool helpers. Server concurrency limits ανήκουν στον Uvicorn, ενώ database και HTTP pool limits ανήκουν στους αντίστοιχους clients.

ΠΑΓΙΔΑ

Autoscaling web replicas χωρίς database connection budget μπορεί να μετατρέψει μια αιχμή σε connection storm και να ρίξει και τα ήδη healthy instances.

ΜΗΧΑΝΙΣΜΟΣ

Κάθε ουρά είναι bounded, κάθε wait έχει deadline και κάθε retry καταναλώνει budget. Capacity υπολογίζεται σε ολόκληρη την topology, όχι ανά endpoint μόνο.

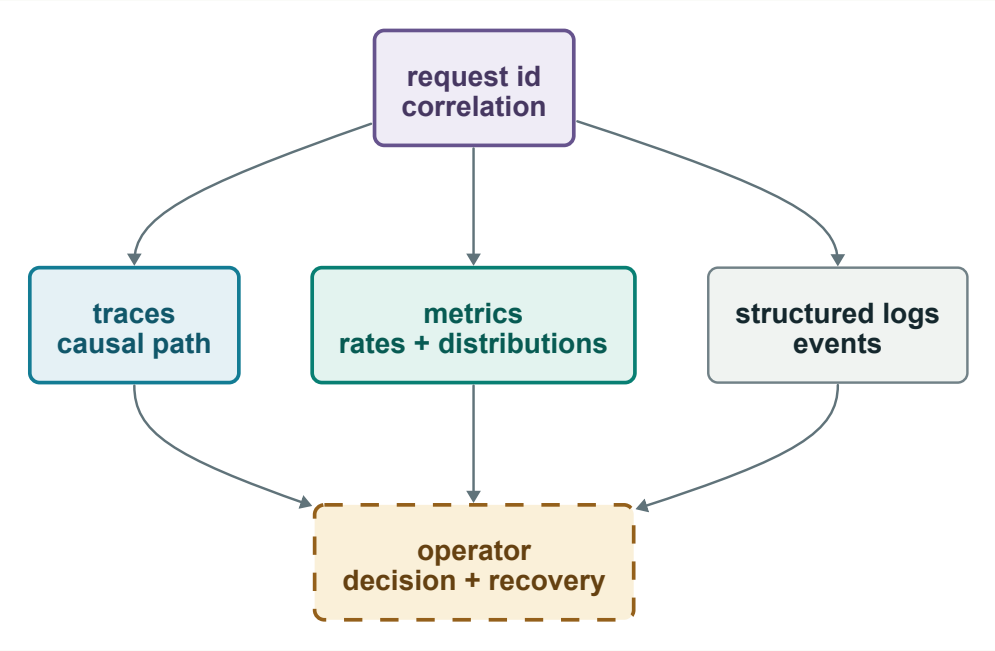
PROBE

Τρέξε load με fixed arrival rate και αύξησέ το σταδιακά. Πρόσθεσε 500 ms latency στον provider και μείωσε database pool στο μισό. Βρες ποια ουρά μεγαλώνει πρώτη, πότε εμφανίζεται 503 και αν η latency επιστρέφει γρήγορα όταν φύγει το fault.

ΚΕΦΑΛΑΙΟ 28

Logs, metrics και traces

Observability δεν είναι συλλογή θορύβου. Είναι η δυνατότητα να συνδέσεις ένα client symptom με route, dependency, query, provider call και durable outcome.



Σχήμα 28.1 · Request id, logs, metrics και traces οδηγούν σε operator decision.

Τα logs περιγράφουν διακριτά events. Τα metrics δείχνουν rates και distributions. Τα traces συνδέουν causal spans ενός request ή job. Request id βοηθά manual correlation, ενώ trace context διαδίδεται ανάμεσα σε υπηρεσίες. Κανένα από αυτά δεν αντικαθιστά τα άλλα.

Ένα access event χρειάζεται timestamp από platform, level, service, deployment revision, request id, method, route template, status, duration και ίσως safe tenant class. Το raw path μπορεί να περιέχει UUID ή προσωπικά δεδομένα και έχει υψηλό cardinality. Χρησιμοποίησε `/orders/{order_id}` όταν το route έχει γίνει match.

PYTHON

```
import logging
from time import perf_counter

from fastapi import FastAPI, Request

app = FastAPI()
logger = logging.getLogger("api.access")
```

```
@app.middleware("http")
async def access_event(request: Request, call_next):
    started = perf_counter()
    response = await call_next(request)
    duration_ms = (perf_counter() - started) * 1000
    route = request.scope.get("route")
    template = getattr(route, "path", "unmatched")
    logger.info(
        "request_complete",
        extra={
            "http_method": request.method,
            "route_template": template,
            "status_code": response.status_code,
            "duration_ms": round(duration_ms, 2),
        },
    )
    return response
```

Το snippet θέλει outer error observation ώστε unexpected exception να δίνει επίσης event. Σε production χρησιμοποίησε structured formatter που παράγει JSON, όχι string concatenation. Το `extra` δεν εγγυάται από μόνο του JSON.

RED και USE

Για HTTP service ξεκίνα με Rate, Errors και Duration ανά route template και method. Status family είναι χρήσιμη, αλλά expected 409 δεν είναι ίδιο με 500. Κράτα business outcome counter όπου χρειάζεται. Histogram buckets πρέπει να ταιριάζουν στα SLOs. Average latency κρύβει tail.

Για resources, Utilization, Saturation και Errors: CPU quota usage, memory, event-loop lag, thread-pool tokens, database pool checked-out/wait, queue age και provider connection pool. Saturation πριν από errors εξηγεί γιατί το p99 ανέβηκε ενώ το success rate φαίνεται ακόμη καλό.

Μην χρησιμοποιείς user id, raw URL, exception message ή idempotency key ως metric label. Το cardinality μπορεί να ρίξει το monitoring backend. Κράτα αυτά σε sampled/redacted logs με access control.

Traces και boundaries

Ένα server span τυλίγει το request. Child spans καλύπτουν SQL, cache και external HTTP. Πρόσθεσε attributes με semantic conventions και low-cardinality values. Domain event όπως `order.created` μπορεί να γίνει span event χωρίς όλο το body.

Διέδωσε standard trace headers σε outbound calls μόνο σε trusted destinations. Μην αντιγράφεις αυθαίρετα incoming baggage σε database labels ή τρίτους. Async task group πρέπει να διατηρεί context. Durable queue χρειάζεται inject στο message metadata και νέο consumer span linked ή parented σύμφωνα με το processing model.

Sampling 1% μπορεί να χάσει σπάνια errors. Tail-based ή error-biased sampling βοηθά αλλά κοστίζει. Metrics παραμένουν unsampled αλήθεια για rates. Exemplars συνδέουν histogram outlier με trace όταν το backend το υποστηρίζει.

Error recording

Expected 404 ή validation 422 δεν χρειάζεται stack trace σε κάθε request. Unexpected 500 χρειάζεται exception type, stack και endpoint context. Η έκδοση αναφοράς του FastAPI κρατά endpoint file, line και function context για response validation errors, κάτι χρήσιμο για diagnosis. Μην επιστρέφεις αυτή τη διαδρομή στον public client.

Κατέγραψε exception μία φορά στο σωστό boundary. Αν external adapter μετατρέπει HTTPX timeout σε application failure, βάλε provider name, timeout phase και attempt. Όχι full URL αν περιέχει secrets. Status code provider χωρίς bounded body preview είναι συχνά αρκετό.

SLO και alerts

SLO ορίζει τι σημαίνει καλό για consumer: availability και latency σε συγκεκριμένο window, ίσως ανά critical endpoint class. Error budget δείχνει πόσο failure επιτρέπεται. Alert σε burn rate έχει περισσότερο νόημα από alert σε κάθε μεμονωμένο 500.

Operational dashboard πρέπει να απαντά: άλλαξε traffic, ποιο route, ποιο deployment, ποιο dependency και υπάρχει durable backlog; Release marker πάνω στα charts συνδέει symptom με change. Runbook λέει ποια ασφαλή ενέργεια κάνει ο operator και πώς επιβεβαιώνει recovery.

Logs ως data product

Ονόμασε events σταθερά και version fields αν τα καταναλώνει automation. Βάλε retention ανά sensitivity. Scrubbing μετά την ingestion είναι αργά αν raw secret έχει ήδη ταξιδέψει. Allowlist logging από την πηγή.

Δοκίμασε observability. Captured logs δεν πρέπει να περιέχουν canary password. Metrics για injected timeout πρέπει να αυξάνονται μία φορά. Trace πρέπει να συνδέει request με provider span και outbox event id χωρίς raw payload.

SOURCE TRACE

Το FastAPI δεν επιβάλλει observability vendor. Το request scope και route objects προέρχονται από το Starlette stack. Στο `fastapi/routing.py`, η `_extract_endpoint_context` και η `ResponseValidationError` προσθέτουν source context για server-side διάγνωση. Η ασφαλής export policy παραμένει της εφαρμογής.

ΠΑΓΙΔΑ

Ολόκληρα headers και bodies σε logs μετατρέπουν το observability system σε δεύτερη, συχνά χειρότερα προστατευμένη βάση προσωπικών δεδομένων και secrets.

ΜΗΧΑΝΙΣΜΟΣ

Χρησιμοποίησε route templates και stable outcome codes. Σύνδεσε unsampled metrics, structured events και causal traces με deployment revision.

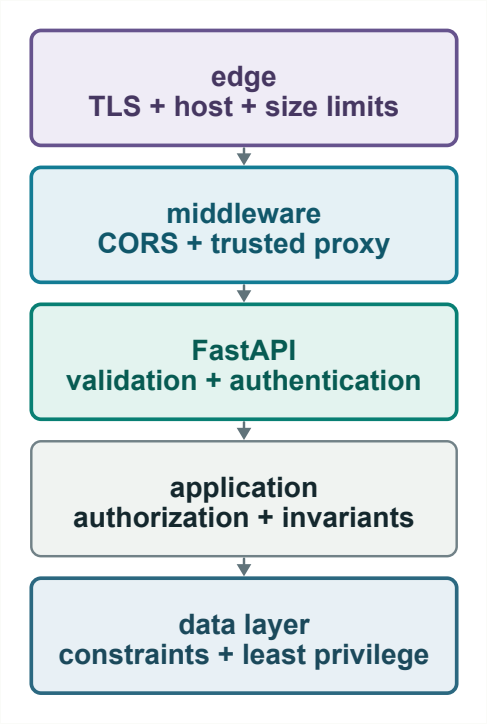
PROBE

Κάνε ένα request που αποτυγχάνει σε validation, ένα σε database timeout και ένα μετά το commit. Από το client request id βρες log, metric και trace. Assert ότι φαίνεται το failure phase και durable outcome χωρίς token ή body leak.

ΚΕΦΑΛΑΙΟ 29

Security hardening σε όλα τα layers

Δεν υπάρχει ένα «security middleware» που κλείνει το θέμα. Edge, ASGI stack, authentication, application policy και database προστατεύουν διαφορετικά όρια.



Σχήμα 29.1 · Η άμυνα ξεκινά στον edge και φτάνει μέχρι τα data constraints.

Ξεκίνα από topology. TLS τερματίζει σε γνωστό edge, το app port δεν είναι δημόσιο και forwarded headers γίνονται trusted μόνο από αυτόν. Το Host header ελέγχεται ώστε password reset links και absolute redirects να μη χρησιμοποιούν attacker domain. Το request size κόβεται πριν από ακριβό parsing.

Το FastAPI επανεξάγει Starlette middleware για κοινές πολιτικές:

PYTHON

```python
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
from fastapi.middleware.httpsredirect import HTTPSRedirectMiddleware
from fastapi.middleware.trustedhost import TrustedHostMiddleware

app = FastAPI(docs_url=None, redoc_url=None)
app.add_middleware(
    TrustedHostMiddleware,
    allowed_hosts=["api.example.test"],
)
app.add_middleware(HTTPSRedirectMiddleware)
app.add_middleware(
    CORSMiddleware,
    allow_origins=["https://app.example.test"],
    allow_credentials=True,
    allow_methods=["GET", "POST", "PATCH"],
    allow_headers=["Authorization", "Content-Type", "If-Match"],
)
```

Αν TLS τερματίζει στον proxy, το HTTPS redirect middleware πρέπει να βλέπει σωστό trusted forwarded scheme, αλλιώς δημιουργεί redirect loop. Η σειρά του middleware επηρεάζει error responses. Δοκίμασε το πραγματικό deployed path.

CORS δεν είναι authorization

CORS είναι browser policy για το αν JavaScript από origin μπορεί να διαβάσει response. Δεν εμποδίζει curl, mobile app ή server να στείλει request. Wildcard και credentials έχουν περιορισμούς. Δήλωσε exact origins και θυμήσου ότι origin περιλαμβάνει scheme, host και port.

Cookie-authenticated state changes χρειάζονται CSRF προστασία. SameSite βοηθά αλλά δεν καλύπτει κάθε integration. Έλεγξε Origin για browser mutations και χρησιμοποίησε anti-CSRF token όπου χρειάζεται. Bearer header που δεν προστίθεται ambiently από browser έχει διαφορετικό risk.

Authentication και authorization

Επαλήθευσε password με reviewed hashing library, JWT με fixed algorithm policy, issuer, audience, expiry και key rotation. Μην καταγράφεις credential. Μετά χτίσε principal και έλεγξε object-level permission μέσα στο consistent workflow. Admin routes δεν προστατεύονται επειδή έχουν `/admin` prefix.

Rate limit login, token refresh και expensive search. Global IP limit μπορεί να τιμωρεί χρήστες πίσω από NAT και παρακάμπτεται με botnet. Συνδύασε account, credential fingerprint, network signal και behavior χωρίς να κάνεις IP identity.

Injection και outbound requests

Χρησιμοποίησε SQLAlchemy bound parameters, όχι string interpolation. Validation του type δεν κάνει safe μια SQL fragment για sort column. Κάνε allowlist των γνωστών column expressions. Το ίδιο ισχύει για shell command, template και log injection.

Server-side request forgery εμφανίζεται όταν client ελέγχει outbound URL. Allowlist scheme και destination, κάνε DNS/IP policy σε κάθε redirect και μπλόκαρε metadata/internal ranges σύμφωνα με

topology. URL parse πριν από DNS δεν αρκεί λόγω rebinding και redirect. Η καλύτερη λύση είναι often να δέχεσαι provider id αντί για arbitrary URL.

HTTP client έχει redirect limit, response size limit και timeouts. Μην κάνεις fetch ολόκληρο unknown body στη μνήμη. Μην προωθείς incoming Authorization header σε άλλο host.

Files και parsers

Client filename δεν γίνεται local path. Έλεγξε πραγματικό format, expanded size, dimensions και archive entry count. Αποθήκευσε uploads εκτός executable path με server-generated ids. Scanning και conversion τρέχουν με ελάχιστα permissions και resource bounds.

Ένα image ή PDF parser είναι supply-chain και memory-safety boundary. Κράτα versions ενημερωμένες, παρακολούθησε advisories και απομόνωσε processing όταν το risk το απαιτεί. Content type header δεν είναι απόδειξη format.

Response και browser hardening

APIs που επιστρέφουν JSON χρειάζονται σωστό `Content-Type` και συχνά `X-Content-Type-Options: nosniff`. HTML surfaces χρειάζονται Content Security Policy, output escaping και frame policy. Μη βάζεις generic headers που σπάνε docs χωρίς να καταλαβαίνεις τον consumer.

Cookies έχουν `Secure`, `HttpOnly`, κατάλληλο `SameSite`, στενό Path και χωρίς υπερβολικό Domain. Cache policy εμποδίζει shared storage private responses. Error body δεν αποκαλύπτει stack, SQL ή filesystem path.

Docs και management surfaces

Το να απενεργοποιήσεις `/docs` δεν ασφαλίζει routes. Μπορεί όμως να μειώσει public discovery ή accidental exposure σε production. Αν κρατήσεις docs, προστάτεψέ τα με πραγματική access policy και βεβαιώσου ότι το OpenAPI δεν περιέχει internal server URLs ή descriptions με secrets.

Health, metrics, profiling και debug endpoints θέλουν ξεχωριστό network ή auth boundary. Το `debug=True` δεν μπαίνει production. Error trackers και traces έχουν redaction πριν από export.

Supply chain και runtime identity

Κλείδωσε direct και transitive versions με hashes όπου υποστηρίζει το tooling. Επιθεώρησε package provenance και release notes. Χτίσε reproducible image, σάρωσε dependencies και τρέξε ως non-root με read-only filesystem όπου γίνεται. Ο runtime database user παίρνει μόνο τα permissions της εφαρμογής, όχι schema owner.

Secret injection γίνεται από πλατφόρμα, όχι image ή repository. Rotation και revocation δοκιμάζονται. Backup επίσης είναι sensitive data και recovery test είναι μέρος security, επειδή ransomware και destructive bugs είναι threats.

> **SOURCE TRACE**
> Τα `fastapi/middleware/cors.py`, `trustedhost.py` και `httpsredirect.py` επανεξάγουν Starlette middleware. Τα security scheme helpers είναι στο `fastapi/security/`, ενώ input parsing και validation βρίσκονται στο route και dependency pipeline. Κανένα από αυτά δεν υποκαθιστά edge limits, database permissions ή application authorization.

ΠΑΓΙΔΑ

`allow_origins=["*"]` ή κρυμμένο `/docs` δεν είναι access control. Ένας μη browser client αγνοεί CORS και γνωρίζει routes χωρίς documentation.

ΜΗΧΑΝΙΣΜΟΣ

Δώσε σε κάθε layer συγκεκριμένο threat: edge περιορίζει transport, FastAPI επικυρώνει shape και identity, use case αποφασίζει permission, database κρατά invariants και least privilege.

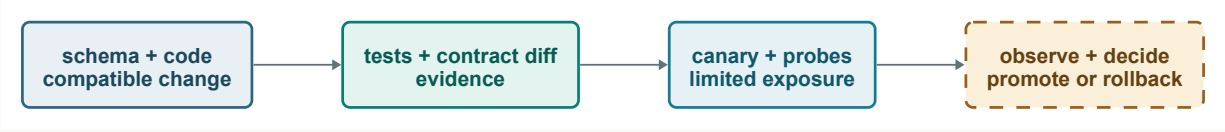
PROBE

Στείλε forged Host και forwarded headers, oversize body, foreign-tenant id, malicious sort field, internal URL και upload με ψεύτικο content type. Assert ότι κάθε ένα κόβεται στο αναμενόμενο layer και δεν εμφανίζεται secret στα logs. Επανάλαβε μέσω του πραγματικού proxy.

ΚΕΦΑΛΑΙΟ 30

Releases, upgrades και correctness packet

Το τελευταίο advanced concept είναι να μπορείς να αλλάζεις το σύστημα χωρίς να μαντεύεις. Κάθε release χρειάζεται compatibility plan, evidence και rollback.



Σχήμα 30.1 · Compatible change, contract evidence, canary και απόφαση release.

Ένα production release αλλάζει περισσότερα από Python code. Μπορεί να αλλάξει OpenAPI, Pydantic coercion, SQL, indexes, worker memory, proxy behavior και queue consumers. Το correctness packet είναι μικρό σύνολο artifacts που λέει τι αλλάζει, ποια invariants κινδυνεύουν και πώς ξέρουμε ότι το deployment είναι καλό.

Πριν από merge, κράτα τουλάχιστον:

Artifact	Αποδεικνύει	Δεν αποδεικνύει
OpenAPI semantic diff	public schema αλλαγές	πραγματικά runtime bodies
Migration plan	schema compatibility	application behavior
Contract tests	statuses, headers, payloads	proxy και network path
Concurrency probes	γνωστά race windows	κάθε πιθανό interleaving
Capacity result	measured operating point	μελλοντικό traffic mix
Rollback note	ασφαλές reversal	recovery μη αναστρέψιμων effects

Το packet δεν είναι μεγάλο document για compliance theater. Είναι links σε συγκεκριμένο schema artifact, migration, test run, load graph και dashboard. Ο reviewer πρέπει να μπορεί να απαντήσει γρήγορα «τι θα δω αν πάει στραβά;».

Expand και contract

Rolling deployment σημαίνει ότι παλιά και νέα app version τρέχουν μαζί. Για rename database column, πρώτα πρόσθεσε νέα column ή συμβατό schema. Μετά deploy code που γράφει/διαβάζει σύμφωνα με migration strategy. Κάνε bounded backfill με progress και load control. Μόνο όταν κανένας παλιός reader δεν εξαρτάται από το παλιό field κάνε contract migration.

Το ίδιο ισχύει για events. Producer δεν στέλνει νέο required field assumption πριν όλοι οι consumers είναι tolerant. Event payload έχει schema version και consumer μετρά unknown versions. Queue retention μπορεί να κρατά παλιά messages πολύ μετά το deploy.

Destructive migration και application release στο ίδιο βήμα καταστρέφουν το άμεσο rollback. Προτίμησε additive, reversible βήματα. Index creation σε μεγάλο table χρειάζεται online/concurrent mechanism της database και παρακολούθηση, όχι απλό startup command από κάθε worker.

Upgrade το framework με συμπεριφορικά probes

Μη διαβάζεις μόνο changelog και πατάς update. Κράτα μικρά probes για όσα εξαρτάται η εφαρμογή: validation shape, dependency caching και cleanup order, route precedence, OpenAPI output, streaming cancellation, middleware headers σε 500 και TestClient lifespan.

Η διαδρομή από παλιό σε FastAPI 0.141.1 περιλαμβάνει αλλαγές που αξίζουν στοχευμένο έλεγχο. Από το 0.132.0 το strict JSON content-type behavior είναι ενεργό από default. Η σειρά 0.137 αναδιοργάνωσε router internals διατηρώντας APIRouter/APIRoute hierarchy. Το 0.139.2 ενίσχυσε thread-safe route building. Η σειρά 0.140 βελτίωσε memory και OpenAPI/dependency generation. Αυτά δεν σημαίνουν ότι η δική σου custom route class ή monkey patch παραμένει συμβατή.

Τα internals που χρησιμοποιήσαμε στα source traces εξηγούν behavior, δεν είναι extension contract. Αν η εφαρμογή εισάγει private names από `fastapi.routing` ή monkey-patches `solve_dependencies`, βάλε το ως explicit upgrade risk και προτίμησε public hooks.

Dependency policy

Κλείδωσε tested set FastAPI, Starlette, Pydantic, Uvicorn, HTTPX και database driver. Το FastAPI δηλώνει ranges, αλλά η επίλυση σήμερα μπορεί να δώσει άλλο set αύριο. Αναβάθμισε σε μικρά batches όταν γίνεται και κράτα lock diff στο review.

Για κάθε νέα package επαλήθευσε identity, maintainers, source repository, release provenance, install behavior και advisories. Μην εκτελείς τυχαίο post-install code σε production build χωρίς έλεγχο. SBOM και image digest συνδέουν το deployed artifact με τα dependencies που δοκιμάστηκαν.

Canary και rollout

Πριν από traffic, τρέξε migration compatibility check και startup/readiness. Στείλε μικρό ποσοστό traffic σε canary με ίδια secrets policy και production downstreams. Σύγκρινε error rate, p95/p99, pool waits, event-loop lag, memory, outbox age και business outcomes με control.

Promotion έχει automatic και human gates. Rollback trigger ορίζεται πριν το deploy, όχι όταν όλοι αγχώνονται. Αν migration είναι backward-compatible, rollback code είναι άμεσο. Αν external effects έγιναν με bug, rollback binary δεν τα αναιρεί. Χρειάζεται reconciliation και repair command με dry-run, idempotency και audit.

Shutdown ως μέρος release

Canary που ξεκινά σωστά αλλά δεν τερματίζει cleanly θα χάσει tasks σε κάθε deploy. Δοκίμασε SIGTERM με in-flight request, database transaction, background consumer, SSE και WebSocket. Το grace period πρέπει να χωρά τα bounded deadlines. Μετά το deadline η πλατφόρμα μπορεί να σκοτώσει τον process, άρα durable recovery παραμένει απαραίτητο.

Το τελικό production checklist

Πριν ονομάσεις την υπηρεσία έτοιμη, μπορείς να δείξεις:

1. request και response models με stable error taxonomy
2. authentication και object-level authorization tests
3. μία explicit transaction ανά command και database constraints για races
4. idempotency ή reconciliation σε retryable side effects
5. bounded bodies, lists, pools, fan-out, retries και queues
6. graceful startup, readiness, drain και cleanup
7. redacted logs, low-cardinality metrics, traces και actionable alerts
8. OpenAPI diff, migration order, canary gates και rollback/repair plan

Αν ένα στοιχείο λείπει, δεν σημαίνει ότι χρειάζεται αμέσως νέο service ή framework. Σημαίνει ότι υπάρχει άγνωστο ownership. Κάν' το visible, βάλε το μικρότερο σωστό mechanism και δοκίμασε το failure που φοβάσαι.

Πηγές έκδοσης αναφοράς

Η έκδοση του βιβλίου βασίζεται στο FastAPI 0.141.1 και στο source tag με commit `95f8322ee1dcda7ceace7b1c4f6c9915b36d748f`. Οι κύριες δημόσιες πηγές είναι:

- `https://pypi.org/project/fastapi/`
- `https://fastapi.tiangolo.com/release-notes/`
- `https://github.com/fastapi/fastapi/tree/0.141.1`
- `https://pypi.org/project/pydantic/`
- `https://pypi.org/project/starlette/`
- `https://pypi.org/project/uvicorn/`
- `https://pypi.org/project/httpx/`
- `https://pypi.org/project/SQLAlchemy/`

Οι version numbers είναι snapshot του Σεπτεμβρίου 2026. Πριν χρησιμοποιήσεις production command ή security option, έλεγξε τα docs της ακριβούς locked έκδοσης. Το βιβλίο σε μαθαίνει πού να κοιτάς και ποιο behavior να επιβεβαιώνεις, όχι να εμπιστεύεσαι για πάντα ένα screenshot έκδοσης.

SOURCE TRACE
Η version του FastAPI δηλώνεται στο `fastapi/__init__.py` και οι dependency ranges στο `pyproject.toml` του tag. Τα release notes καταγράφουν behavioral αλλαγές. Οι κρίσιμες route, dependency, OpenAPI και exception διαδρομές που χρησιμοποιήσαμε βρίσκονται στα `fastapi/routing.py`, `fastapi/dependencies/utils.py`, `fastapi/openapi/utils.py` και `fastapi/exception_handlers.py`.

ΠΑΓΙΔΑ
Green unit tests με migration που αφαιρεί column δεν δίνουν rollback. Η παλιά app version πρέπει να μπορεί να τρέξει πάνω στο νέο schema σε όλο το rollout window.

ΜΗΧΑΝΙΣΜΟΣ
Advanced FastAPI δεν σημαίνει περισσότερα decorators. Σημαίνει ότι ξέρεις ποιος κατέχει κάθε resource και invariant, τι συμβαίνει σε κάθε failure window και ποια evidence επιτρέπει ασφαλή αλλαγή.

PROBE
Κάνε rehearsal ολόκληρου release σε staging: schema expand, old/new coexistence, canary, injected provider timeout, SIGTERM με in-flight traffic και rollback. Παράδωσε το correctness packet σε άλλον engineer και δες αν μπορεί να αποφασίσει promote ή rollback χωρίς προφορική πληροφορία.