```python
import os
from dotenv import load_dotenv

class Settings:
    def __init__(self) -> None:
        load_dotenv()
        self.SQLALCHEMY_DATABASE_URI = os.getenv(
            "DATABASE_URL",
            "postgresql+psycopg://postgres:postgres@localhost:5432/fintrack",
        )
        self.SECRET_KEY = os.getenv("SECRET_KEY", "change_me")
        self.JWT_SECRET_KEY = os.getenv("JWT_SECRET_KEY", "change_me_jwt")
        self.APP_NAME = os.getenv("APP_NAME", "Finance Tracker B3")
        self.APP_VERSION = os.getenv("APP_VERSION", "1.0.0")

def get_settings() -> Settings:
    return Settings()
```

`app/core/extensions.py`

```python
from flask_sqlalchemy import SQLAlchemy
from flask_migrate import Migrate
from flask_marshmallow import Marshmallow
from flask_jwt_extended import JWTManager
from flask_bcrypt import Bcrypt

db = SQLAlchemy()
migrate = Migrate()
ma = Marshmallow()
jwt = JWTManager()
bcrypt = Bcrypt()
```

`app/core/di.py`

```python
from typing import Callable
from flask_sqlalchemy.session import Session
from .extensions import db

# very light DI helpers

GetSession = Callable[[], Session]
```

```
def provide_session() -> Session:
    return db.session
```

`app/core/errors.py`

```
from flask import jsonify

class AppError(Exception):
    code = 400
    message = "Bad request"

    def __init__(self, message: str | None = None, code: int | None = None) ->
None:
        if message:
            self.message = message
        if code:
            self.code = code
        super().__init__(self.message)


def register_error_handlers(app):
    @app.errorhandler(AppError)
    def handle_app_error(e: AppError):
        return jsonify({"error": e.message}), e.code

    @app.errorhandler(404)
    def handle_404(_):
        return jsonify({"error": "Not found"}), 404

    @app.errorhandler(500)
    def handle_500(e):
        return jsonify({"error": "Internal server error"}), 500
```

---

4) App Factory & API Aggregator

`wsgi.py`

```
from flask import Flask, jsonify
from app.core.config import get_settings
from app.core.extensions import db, migrate, ma, jwt, bcrypt
from app.core.errors import register_error_handlers
from app.api.v1 import api_v1
```

```
def create_app() -> Flask:
    app = Flask(__name__)
    settings = get_settings()

    app.config.update(
        SQLALCHEMY_DATABASE_URI=settings.SQLALCHEMY_DATABASE_URI,
        SQLALCHEMY_TRACK_MODIFICATIONS=False,
        SECRET_KEY=settings.SECRET_KEY,
        JWT_SECRET_KEY=settings.JWT_SECRET_KEY,
        APP_NAME=settings.APP_NAME,
        APP_VERSION=settings.APP_VERSION,
    )

    # extensions
    db.init_app(app)
    migrate.init_app(app, db)
    ma.init_app(app)
    jwt.init_app(app)
    bcrypt.init_app(app)

    # api
    app.register_blueprint(api_v1, url_prefix="/api/v1")

    # errors
    register_error_handlers(app)

    @app.get("/")
    def root():
        return jsonify({"name": settings.APP_NAME, "version":
settings.APP_VERSION})

    return app

app = create_app()

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=8000)
```

app/api/v1.py

```
from flask import Blueprint

api_v1 = Blueprint("api_v1", __name__)

# Import & register module routes
```

```
from app.modules.users.routes import bp as users_bp  # noqa
from app.modules.categories.routes import bp as categories_bp  # noqa
from app.modules.transactions.routes import bp as transactions_bp  # noqa
from app.modules.budgets.routes import bp as budgets_bp  # noqa
from app.modules.transactions.reports_routes import bp as reports_bp  # noqa

api_v1.register_blueprint(users_bp)
api_v1.register_blueprint(categories_bp)
api_v1.register_blueprint(transactions_bp)
api_v1.register_blueprint(budgets_bp)
api_v1.register_blueprint(reports_bp)
```

---

5) Users Module

`app/modules/users/models.py`

```
from sqlalchemy.orm import Mapped, mapped_column, relationship
from sqlalchemy import String
from app.core.extensions import db

class User(db.Model):
    __tablename__ = "users"

    id: Mapped[int] = mapped_column(primary_key=True)
    email: Mapped[str] = mapped_column(String(120), unique=True, index=True,
nullable=False)
    name: Mapped[str] = mapped_column(String(120), nullable=False)
    password_hash: Mapped[str] = mapped_column(String(255), nullable=False)

    categories = relationship("Category", back_populates="owner", cascade="all,
delete-orphan")
    transactions = relationship("Transaction", back_populates="owner",
cascade="all, delete-orphan")
    budgets = relationship("Budget", back_populates="owner", cascade="all,
delete-orphan")
```

`app/modules/users/schemas.py`

```
from marshmallow import Schema, fields

class UserOut(Schema):
    id = fields.Int()
```