**US Tax Rules Engine - Low Level Design (LLD)**

This LLD is designed for a **US Tax Filing Backend System** built using **NestJS**, **TypeScript**, **PostgreSQL**, and **Swagger**. The architecture covers taxpayer profiles, return submissions, and a tax rules engine.

**1. Database Schema Design**

**a. taxpayers**

* id UUID (Primary key)
* full\_name VARCHAR
* ssn VARCHAR (Unique)
* filing\_status ENUM (single, married, head\_of\_household, etc.)
* created\_at, updated\_at TIMESTAMPTZ

**b. tax\_returns**

* id UUID (Primary key)
* taxpayer\_id UUID (Foreign key to taxpayers)
* tax\_year INTEGER
* gross\_income NUMERIC
* deductions NUMERIC
* taxable\_income NUMERIC
* tax\_due NUMERIC
* created\_at, updated\_at TIMESTAMPTZ

**c. tax\_rules**

* id UUID (Primary key)
* rule\_code VARCHAR
* description TEXT
* filing\_status ENUM
* min\_income, max\_income NUMERIC
* rate NUMERIC (e.g., 0.1 = 10%)
* created\_at, updated\_at TIMESTAMPTZ

**2. NestJS Project Structure**

src/

├── taxpayers/

│ ├── taxpayers.controller.ts

│ ├── taxpayers.service.ts

│ ├── taxpayers.module.ts

│ ├── dto/

│ └── entities/

│

├── returns/

│ ├── returns.controller.ts

│ ├── returns.service.ts

│ ├── returns.module.ts

│ ├── dto/

│ └── entities/

│

├── rules-engine/

│ ├── rules.controller.ts

│ ├── rules.service.ts

│ ├── rules.module.ts

│ ├── dto/

│ └── entities/

│

├── shared/

│ ├── filters/

│ ├── guards/

│ ├── interceptors/

│ └── utils/

│

├── main.ts

└── app.module.ts

**3. API Endpoints Design**

**Taxpayer**

* POST /taxpayers : Create taxpayer
* GET /taxpayers : List taxpayers
* GET /taxpayers/:id : Get taxpayer by ID

**Tax Returns**

* POST /returns : Submit tax return (with calculation)
* GET /returns : List all returns (optional filters)
* GET /returns/:id : Return by ID

**Rules**

* POST /rules : Create rule
* GET /rules : List rules
* GET /rules/:id : Rule by ID

**4. DTOs**

**Taxpayer**

* CreateTaxpayerDto: name, ssn, filing\_status
* TaxpayerResponseDto: id, full\_name, timestamps

**Returns**

* CreateReturnDto: taxpayer\_id, gross\_income, deductions, tax\_year
* ReturnResponseDto: all fields, tax calculated

**Rules**

* CreateRuleDto: code, description, min/max income, filing\_status, rate
* RuleResponseDto: id, rule details

**5. Services**

**Taxpayer Service**

* create() : Create taxpayer
* findAll() / findOne()

**Return Service**

* create() : Validate, apply rules, compute tax
* findAll() / findOne()

**Rule Service**

* create(), findAll(), findOne()

**6. Business Logic / Rule Engine**

1. Get rules by filing status
2. Filter rules by taxable income range
3. Apply the correct rate

const rules = getRulesByFilingStatus(filing\_status);

for (const rule of rules) {

if (income >= rule.min\_income && income <= rule.max\_income) {

tax += income \* rule.rate;

}

}

**7. Validations**

* Positive number checks
* Taxable income >= 0
* Use class-validator decorators

**8. Swagger Integration**

import { DocumentBuilder, SwaggerModule } from '@nestjs/swagger';

const config = new DocumentBuilder()

.setTitle('US Tax API')

.setDescription('API for US federal tax returns')

.setVersion('1.0')

.build();

const document = SwaggerModule.createDocument(app, config);

SwaggerModule.setup('api', app, document);

**9. Deployment & Dev Tools**

* .env for configs
* Dockerfile + docker-compose.yml
* NestJS logger for monitoring
* Unit tests with Jest