Title: Youdemi App

The software requirement document for a course management platform.

Authors: Emmanuel Mojiboye

Date: July 2nd, 2025

Version: 1.0.0

Table of Contents

7.1 Unit Testing.
7.2 Integration Testing.
7.3 System Testing.
7.4 User Acceptance Testing.

8. Deployment.

9. Maintenance and Support

10. Appendices’

1. Introduction

1.1 Purpose

The Youdemi App aims to provide a platform where Users from across the world can come to learn a course. This Web Application will offer a flexible and smooth medium for Youdemi Instructors to interact with their Students.

1.2 Scope

The Youdemi Application will contain a range of features ranging from how Users sign up for an account to different roles the User can perform. There will be different roles in the application that a User can perform, ranging from User role, Admin role, Instructor, Students.

1.3 Project's structure.

/youdemi-app

```
|
├── /src
|   ├── /config          → Environment config, DB config, etc.
|   ├── /controllers     → Handle request/response logic
|   ├── /routes          → API route definitions
|   ├── /services        → Business logic (no direct DB calls here)
|   ├── /repositories    → DB logic, Prisma client usage
|   ├── /middlewares     → Auth, error handlers, etc.
|   ├── /models          → Prisma schema or types/interfaces
|   ├── /utils           → Reusable helper functions
|   ├── /validators      → Joi/Yup/Zod validation for requests
|   ├── /tests           → Unit & integration tests (Jest)
|   └── /app.js          → Express app entry
|
├── prisma/schema.prisma   → Prisma schema file
├── .env                   → Environment variables
├── Dockerfile             → Docker containerization
├── docker-compose.yml     → For running DB and app locally
├── package.json
└── README.md
```

1.4 Definitions, Acronyms and Abbreviations.

- **Web Application/Web App/Website/Application:** This is going to be the application that will be used to carry out various activities related to a course platform.
- **User:** This is an account owner on the web application. Everyone who creates an account on the application immediately becomes a User. But along the line, a User can be promoted to newer roles.
- **User Roles:** This explains the activities a **User** can perform on the application.

- **Authentication:** This is the process of verifying the identity of a person or a device. On this app, Authentication will be carried out by entering the username/email address and a password.
- **Username:** This is a unique identifier for user's account on a computer system.
- **Password:** This is a string of characters that performs part of user authentication on a computer system, along with a **Username.**
- **Login:** A registered User can enter their username/email and password.
- **Client:** An application that sends a request.
- **Server:** Another application that listens for requests and sends the required responses.
- **REST (Representational State Transfer):** A way in which APIs, API endpoints can be written, other examples include; gRPC, SOAP.
- **TDD (Test Driven Development):** This is an approach to building softwares. In TDD, the application is broken down into multiple specifications, tests will be written for each spec and the **RGR** workflow would be used in implementing each step. This TDD will be used for both Unit Testing and Integration Testing.

1.5 Technologies:

1. Front End: Html, CSS, JavaScript.
2. Backend: NodeJS
3. Database: PostgreSQL
4. ORM: Prisma
5. Authentication and Authorization: JWT (Json-Web-Tokens)
6. Docker for containerization
7. Testing: Jest
8. API Documentation: SwaggerUI
9. Development Tools: Visual Studio Code, Git for source control, GitHub

1.5 References

- Software Design Tutorial; **From Tech With Tim on Youtube.**

2. Overall Description

2.1 Product Perspective

Youdemi App is a Web Application where Users can come to take a course.

2.2 Product Features

- **User must be able to create an account**
- **User must be able to login to an account.**
- **User can either login with username or email address**
- **User must be able to view all courses**

- **User must be able to view a specific course**
- **User must be able to buy a course**
- **Admin must be able to edit a course**
- **Admin must be able to delete a course**
- **Admin must be able to assign more roles to user**

2.3 Stages for verifying a User Account:

To become a fully verified User on our web application, each individual must pass the following stages of verification:

i. **KYC Tier 1:** The following are the details needed from the User to pass this stage: username, email, password.
ii. **KYC Tier 2:** The following are the details needed from the User to pass this stage: dateOfBirth, country
iii. **KYC Tier 3:** The following are the details needed form the User to pass this stage: phoneNumber (verified via OTP).

After an individual completes all of the stages above, then such individual can be called a **fully verified User.**

2.4 Data Models and Relationships

User

Represents anyone who has an account on the platform. Users can be promoted to other roles (Student, Instructor, Admin).

Field	Type	Description
id	int	Unique Identifier
email	string	Unique email for user login
username	string	Unique username
Password	string	Hashed password
role	int	One of: **USER, STUDENT, INSTRUCTOR, ADMIN**
kycTier	int	0 to 3 indicating verification level
dateOfBirth	date?	Required in Tier 2
country	string?	Required in Tier 2
phoneNumber	string?	Required in Tier 3
phoneVerified	Boolean	Required in Tier 3
courses	Course[]	Courses created (if instructor)
orders	Order[]	Courses bought (if student)

Course

Represents a single course offered by an instructor.

Field	Type	Description
id	int	Unique Identifier
courseName	string	Name of the course
Completed	Boolean	Whether the course is fully uploaded
instructorId	Int	FK to User who created it
Instructor	User	The instructor who owns the course
Orders	Order[]	List of orders/purchases related to the course

Order

Represents a course purchase transaction made by a user.

Field	Type	Description
id	int	Unique Identifier
userId	Int	FK to the user who made the order
courseId	Int	FK to the course being purchased
status	Enum	PENDING, COMPLETED, FAILED
referenceId	User	The unique payment reference from Paystack
course	Course	Reference to purchased course
createdAt	datetime	Timestamp of order
updatedAt	datetime	

2.4.1 Enumerations

Role Enum

Value	Description
USER	Default role for all newly registered users
STUDENT	Role for users who purchase courses
INSTRUCTOR	Role for users who create courses
ADMIN	Highest role for managing all accounts and courses

OrderStatus Enum

Value	Description
PENDING	The order has been initiated but not completed
COMPLETED	The payment was successful and access is granted
FAILED	Payment failed or was cancelled

2.4.2 Entity Relationships

Entity	Relationship	Related Entity	Description
User	1-to-Many	Course	A user (as instructor) can own many courses
User	1-to-Many	Order	A user (as student) can place multiple orders
Course	Many-to-1	User	A course belongs to one instructor
Course	Many-to-1	Order	A course can be bought many times
Order	Many-to-1	User	Each order belongs to one user
Order	Many-to-1	Course	Each order is for one course

3. Functional Requirements

3.1 User Management

- The system shall allow a new user to register using a valid email, username, and password.
- The system shall allow users to log in using either their email or username.
- The system shall hash all passwords before storing them in the database.
- The system shall validate user credentials during login and return a JWT token on success.
- The system shall allow users to update their profile information.
- The system shall implement 3-tier KYC verification as described in section 2.3.
- The system shall restrict access to certain features based on the user's role and KYC level.
- The system shall send and validate an OTP for phone number verification to achieve Tier 3, using an SMS provider (e.g., Twilio, Nexmo, Firebase Auth)
- Only users with `phoneVerified = true` can access high-risk or premium features.

3.2 Admin Management

- The system shall allow users with the `ADMIN` role to log into a dedicated admin dashboard.
- The system shall allow the admin to view a list of all users.
- The system shall allow the admin to promote or demote a user's role (e.g., from `USER` to `INSTRUCTOR` or `STUDENT`).
- The system shall allow the admin to deactivate or delete any user account (except other admins, unless built-in protections are overridden).
- The system shall allow the admin to manage system-wide settings (e.g., platform policy changes, payment gateway toggles).
- The system shall allow the admin to view logs of KYC verification statuses for each user.

3.3 Order Management

- The system shall allow logged-in users to place an order for a course.
- The system shall ensure that the same user cannot place multiple orders for the same course.
- The system shall save the order with a status of `PENDING` by default until payment is confirmed.
- The system shall allow users to view their past and current orders from a dashboard.
- The system shall allow the system (or admin, or automated service) to update the order status to `COMPLETED` or `FAILED`.

3.3 Payment Management

- The system shall support integration with a payment gateway (e.g., Paystack, Stripe) to handle real money transactions.
- The system shall confirm payment status with the gateway and update order status accordingly.
- The system shall log each log each webhook event and status change.
- The system shall not allow course access unless payment is marked `COMPLETED`.
- The system shall restrict access to payment functionality unless the user has passed KYC Tier 3.

4. Non-Functional Requirements

4.1 Performance

- The system shall respond to user requests within 2 seconds under normal conditions.
- The application shall be able to support **at least 500 concurrent users** without crashing.
- The system shall maintain an uptime of **99.5%** monthly.
- Page load times shall be under **1.5 seconds** for key pages (Home, Course List, Dashboard).
- The database shall be optimized using indexes to support efficient query performance.

4.2 Security

- All user passwords shall be **hashed using a strong algorithm** (e.g., bcrypt) before storage.
- The system shall implement **JWT-based authentication** for securing API endpoints.
- Sensitive routes and data shall be **role-protected** (e.g., only Admins can assign roles).
- The system shall support **rate limiting** to prevent brute-force login attempts.
- The app shall enforce **HTTPS-only** access in production.

4.3 Usability

- The system shall provide a **clean and intuitive UI/UX** suitable for both desktop and mobile devices.
- Navigation shall be **clear and consistent** across the platform.The system shall maintain an uptime of **99.5%** monthly.
- User actions such as signing up, purchasing a course, or editing a profile shall be **achievable within 3 steps.**
- Error messages shall be **clear, actionable, and user-friendly**.
- The platform shall support **keyboard navigation and accessibility** wherever possible.

4.4 Scalability

- The backend system shall be containerized using Docker for easier scaling and deployment.
- The database and application shall be designed in a way that supports **horizontal scaling.**
- The application shall allow the addition of **new features and roles** with minimal changes to the existing codebase.
- Microservice splitting (e.g., Payments, Courses) shall be considered when traffic grows beyond a certain threshold.
- The system shall be capable of handling a **10x growth in traffic** without major architectural redesign.

5. System Architecture

5.1 Proposed Architecture: Modular 3-Tier Layered Architecture

The Youdemi App will adopt a **Modular 3-Tier Layered Architecture**, enhanced with **Domain specification,** to ensure clear separation of concerns, scalability, testability, and maintainability.

Layer Breakdown

i. **Presentation Layer (Client / Controllers)**
   - Handles HTTP requests from users (via UI or API clients like Postman).
   - Routes requests to the appropriate service logic.
   - Responsible for formatting responses (success, error, pagination, etc).

ii. **Business Logic (Services / Use Cases)**

- Contains the core logic of the application.
- Validates data, enforces rules (e.g., KYC Checks, role access, course purchase logic).
- Acts as the “brain” of each feature module (User, Course, Order, etc).

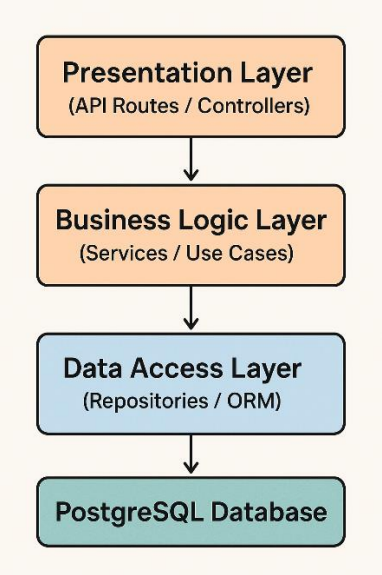
iii. Data Access Layer (Repositories)

- Interacts directly with the PostgreSQL database via **Prisma ORM**.
- Executes CRUD operations, complex queries, and joins.
- Abstracted so logic in the service layer is not tightly coupled to the database technology.

Why This Architecture?

Reason	Explanation
Separation of concern	Each layer has a specific responsibility; changes in one don’t affect others.
Testability	Services can be tested independently from controllers or the database.
Scalability	New Features (e.g., Wishlist, Messaging) can be added without breaking existing flow.
Security	Logic-based security enforcement is easier at the service layer.
Team Collaboration Friendly	Allows backend and frontend teams to work in parallel, even on separate features.

Visual Representation



5.2 System Components

i. **User Component**
   - Handles user registration, login, profile updates, and KYC verification.
   - Manages JWT authentication and session validation.

ii. **Role Component**
   - Manages different user roles: USER, STUDENT, INSTRUCTOR, ADMIN.
   - Handles role-based authorization logic for restricted access.
   - Allows role promotion/demotion (especially by admins).

iii. **KYC Component**
   - Handles 3-tier identity verification (Tier 1: Basic Info, Tier 2: DOB & Country, Tier 3: Phone Number).
   - Ensures progression and compliance with platform rules. features.

iv. **Course Component**
   - Manages creation, editing, publishing, and deletion of courses by instructors.
   - Handles course listing for all users.
   - Restricts full access to purchased users only.

v. **Order Component**
   - Handles course purchase requests.
   - Creates and tracks the status of orders (`PENDING`, `COMPLETED`, `FAILED`).
   - Prevents duplicate purchases and enforces access rules.

vi. **Payment Component**
   - Integrates with external payment gateways like Paystack or Stripe.

5.3 Data Storage

i. Database Technology

The Youdemi App will use PostgreSQL, a powerful open-source relational database system known for:

- Strong ACID compliance
- Support for complex queries and joins
- Rich data types and indexing
- Extensibility (e.g., full-text search support)

Data will be managed through Prisma ORM, which provides:

- Type-safe database queries
- Clear schema management
- Migration tools
- Easy integration with TypeScript/JavaScript-based backend (Node.js)
- Support for complex queries and joins

ii. Database Structure

All core entities will be stored as relational tables with foreign key relationships, including:

- User
- Course
- Order
- Enums like `Role`, `OrderStatus`, `CardType` are represented in the application logic, not as separate tables (handled by Prisma Enums).

Each model is fully defined in Section 2.4 and adheres to relational best practices, including:

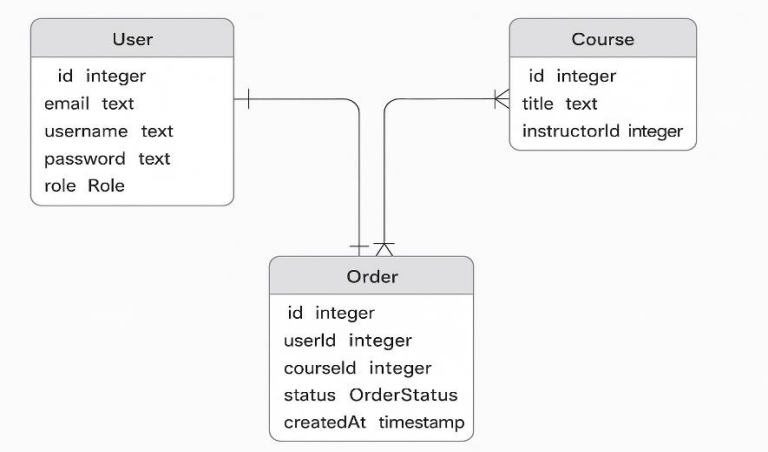
- Primary keys
- Foreign key constraints
- Optional and required fields

iii. Data Integrity

- Prisma migrations ensure schema versioning and consistency.
- All inputs will be validated at both the **API layer (validators)** and the **ORM layer (Prisma constraints).**
- Database-level constraints (e.g., NOT NULL, UNIQUE, FOREIGN KEY) will enforce consistency.

iii. Data Integrity

- In production, the database will be hosted on a managed service (e.g., Supabase, Railway, Render, or AWS RDS).
- Scheduled automated backups will be configured based on the host provider.
- Rollbacks will be supported via migration history and database snapshots.



5.4 Integration Points

The Youdemi App will integrate with the following external systems and services:

i. **Payment Gateway – Paystack (or Stripe)**
- Purpose: Handles all monetary transactions between users and the platform.
- Integration Type: RESTful API
- Endpoints Used:
  - Initialize transaction
  - Webhooks for payment status updates
- Reason: Outsources sensitive payment handling and compliance requirements.

ii. **Email Notification Service**
- Purpose: Send transactional emails for events such as registration, course purchases, KYC updates.
- Example of Tools to use: Mailgun, SendGrid, or Nodemailer
- Integration Type: SMTP or REST API
- Reason: Improve communication and trust with users.

iii. **Swagger UI**
- Purpose: Provides interactive API documentation.
- Integration Type: Internal middleware (Swagger Express or Swagger-UI-Express).
- Reason: Helps both internal and external developers test and understand the API.

iv. **GitHub + CI/CD**
- Purpose: Source control, pull requests, and continuous integration/deployment.
- Integration Type: GitHub Actions (for CI), Railway/Render/Netlify (for deployment).
- **Integration Type**: Streamlined, automated deployment pipeline.
-

v. **GitHub + CI/CD**
- **Analytics Platform (e.g., Mixpanel, Google Analytics)** – to track user behavior.
- **SMS Notification API (e.g., Twilio)** – for 2FA or KYC updates via text.
- **Cloud Storage (e.g., AWS S3)** – to store large course files (if you ever add videos).

6. User Interface Design

7. Test Plan

7.1 Unit Testing

Unit testing involves testing individual components or functions in isolation to ensure they behave as expected.

Tools: Jest (JavaScript Testing Framework)

Focus Areas:

- User registration & login functions
- Role checking utilities
- Course creation logic
- Payment reference verification
- Order status update logic

Approach:

- Use mocking to isolate services from external dependencies (e.g., Prisma, Paystack).
- Follow TDD approach where feasible.

7.2 Integration Testing

Integration testing verifies that different parts of the system (e.g., services, controllers, and DB) work correctly together.

Tools: Jest + Supertest

Focus Areas:

- Auth flow: Register → Login → Access token
- Course flow: Instructor creates → Student views
- Payment flow: Order placed → Webhook updates status
- Role assignment by admin

Approach:

- Use a test database (PostgreSQL) with Prisma test config
- Include setup/teardown hooks for test isolation

7.3 System Testing

System testing validates the end-to-end behavior of the complete application based on the requirements.

Scenarios Covered:

- A new user signs up, completes KYC, buys a course
- An admin logs in, promotes a user to instructor
- A student accesses a purchased course only
- A failed payment prevents course access

Tools: Manual testing (Postman + Browser) + Automated scripts (if needed)

7.4 System Testing

UAT checks whether the system meets user expectations and business requirements.

Stakeholders: Product owner, selected users/testers

Test Methods:

- Review completed features against the SRS
- Walkthroughs with test users
- Feedback-driven bug fixing and polishing

Success Criteria:

- All functional requirements in Section 3 behave as described
- App is stable with no critical bugs
- User feedback is generally positive
-

8. Deployment

This section describes **how the application will be deployed**, including tools, environments, and strategies.

8.1 Deployment Environment

The Youdemi App will be deployed in a cloud-based environment to ensure accessibility, scalability, and ease of management.

Deployment Targets:

- Backend API: Node.js server (Express) containerized with Docker
- Database: PostgreSQL hosted via Railway, Supabase, or Render
- Frontend: Static HTML/CSS/JS files hosted on Netlify, GitHub Pages, or Vercel (if separate)

8.2 Tools & Platforms

Tool/Platform	Purpose
Docker	Containerization of the backend server
GitHub	Source control & CI/CD triggers
Railway / Render	App & database hosting (backend + DB)
Prisma	ORM & migration tool for DB deployment
GitHub Actions	Automate tests and deployments on push

8.3 Deployment Strategy

- Development will take place locally with `.env` configuration files.
- The project will be **containerized using Docker** for environment consistency.

- On every stable feature or version push to the `main` branch:
    - Tests will run (via Jest)
    - App will be rebuilt and deployed to the hosting provider (e.g., Render)
- A staging version may be hosted separately for UAT before pushing to production.

8.4 Environment Configuration

All sensitive configuration values will be stored in `.env` files and managed via platform-specific secret managers. Examples include:

DATABASE_URL=postgres://user:pass@host:5432/db

JWT_SECRET=supersecretkey

PAYSTACK_SECRET=sk_test_xxx

8.5 Monitoring and Logs

- Basic logging will be enabled via `console.log()` or a logging library like `Winston`.
- Platform monitoring (e.g., Render or Railway) will provide insights into app health and logs.

9. Maintenance and Support

This section outlines how the application will be maintained over time, what kind of support is expected, and how issues or changes will be handled post-deployment.

9.1 Maintenance Plan

- **Bug Fixes**
  All reported bugs will be logged and resolved in a prioritized manner based on severity and impact on user experience.
- **Feature Updates**
  New features or improvements will be managed through a version-controlled Git workflow. Each update will go through development, testing, and deployment pipelines.
- **Dependency Management**
  Third-party packages (e.g., Prisma, Express, JWT) will be reviewed and updated regularly to maintain security and compatibility.
- **Database Migrations**
  Schema changes will be handled using Prisma Migrate, ensuring data consistency and rollback support.

9.2 Maintenance Plan

- **Issue Tracking**
  GitHub Issues will be used to track bugs, feature requests, and improvements.

- **Documentation**
  A `README.md` file and Swagger API docs will be maintained for developers. Comments and docstrings will be added to services and controllers.
- **Monitoring**
  Application logs and hosting platform metrics (e.g., Render/Railway) will be monitored for performance and uptime.
- **User Feedback Loop (Optional Future Plan)**
  Feedback may be collected via email forms or a support link in the UI to understand user pain points and areas for improvement.

9.3 Long-Term Considerations

- The system will be designed to allow **modular improvements** without requiring a full rewrite.
- A migration plan will be in place in case of future changes (e.g., moving to microservices or using a different database).
- Security patches and payment gateway updates will be applied promptly to reduce vulnerability risks.

10. Appendices

10.1 Glossary

A glossary of terms used throughout the document for clarification.

Term	Definition
KYC	Know Your Customer – identity verification process
JWT	JSON Web Token – a token format used for user authentication
ORM	Object-Relational Mapper – tool for interacting with databases
REST API	A standard architectural style for designing networked applications
Prisma	The ORM used in this project to interact with the PostgreSQL database

10.2 Diagrams

- **Entity-Relationship Diagram**: See Section 5.3
- **System Architecture Diagram**: See Section 5.1

10.3 Sample Data

A sample data to be used when performing User Acceptance Testing.

Field	Example Value
Email	johndoe@email.com

Password	SecurePass123
Course Name	JavaScript for Beginners
Reference ID	psk_txn_2394asd293

10.4 Document History

A table to show days in which this document was edited or new features were added into the application.

Version	Date	Author	Change Description
1.0.0	July 2, 2025	Emmanuel Mojiboye	Initial version completed