Technical Design Document for Bartr

Contents:

8. Services

1. Introduction

1.1 Purpose:

This Technical Design Document (TDD) elaborates on the technical architecture, design patterns, data model, and technology stack required to implement the insurance platform outlined in the Functional Requirements Document (FRD) version It serves as a blueprint for the development team to build and deploy the system.

1.2 Intended Audience:

This document is intended for the development team (software engineers, database administrators, DevOps engineers), architects, technical leads, and quality assurance engineers involved in the insurance platform project.

1.3 Scope:

This TDD covers the high-level and low-level technical design aspects of the core functionalities described in the FRD, including creating account, login to the account, view profile, creating a course, enrolling into a course, search for a course, purchasing of XPs. It also addresses the non-functional requirements related to performance, usability, reliability, and scalability.

1.4 Relationship to FRD:

This document directly references and expands upon the functional and non-functional requirements defined in the FRD. Each technical design decision aims to fulfil one or more of the stated requirements.

2. Technologies Used

1. Frontend – HTML5, CSS3, Bootstrap, Angular
2. Backend API – Spring Boot (Java)
3. Database – Appwrite (Cloud/Hosted)
4. Authentication – Appwrite Auth
5. Payment Gateway – Stripe API

3. Database Implementation

The database will be designed to store and retrieve data related to users, categories, payments, transactions, courses, enrollments. Below is the design of key entities and their attributes:

3.1 users:

- id int primary Key
- username varchar
- email varchar
- password varchar
- phone varchar
- fullname varchar
- xp int
- avatar_url varchar
- created_at timestamp

3.2 categories:

- id int
- name varchar
- description text
- xp_cost int

3.3 courses:

- id int primary key
- title varchar
- description text
- category_id int foreign key referencing categories.id
- creator_id int foreign key referencing users.id
- created_at timestamp
- level varchar

3.4 enrollments:

- id int primary key
- course_id int foreign key references courses.id

- learner_id int foreign key references users.id
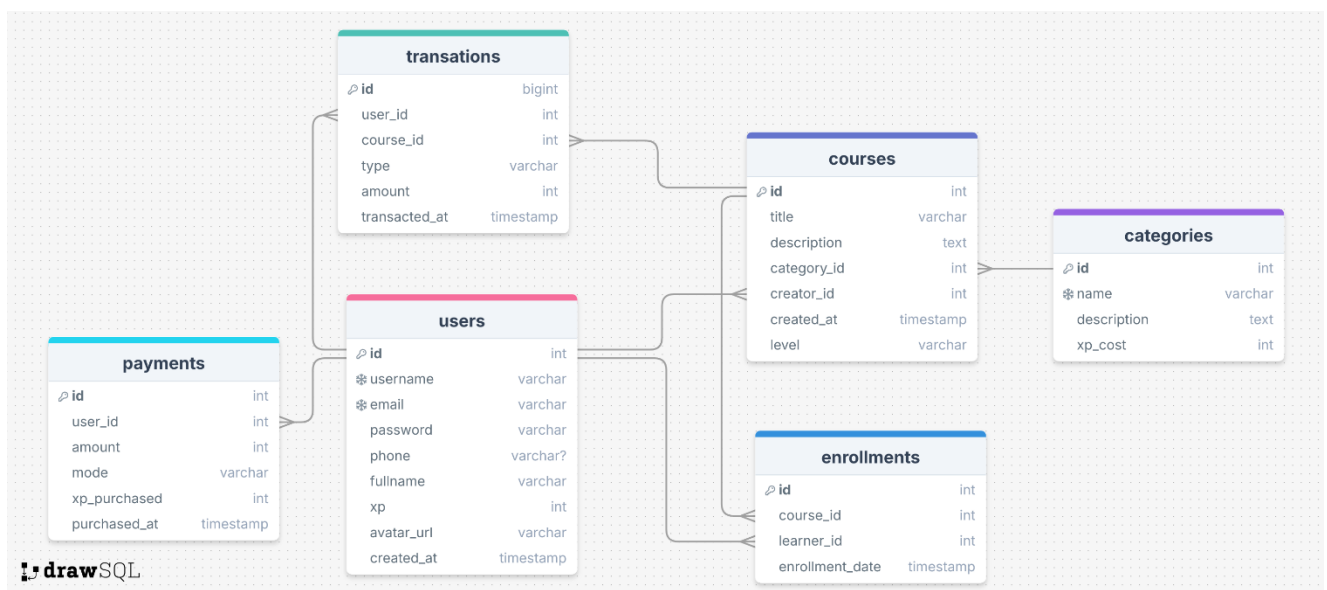- enrollment_date timestamp

3.5 transactions:

- id int primary key
- user_id int foreign key references users.id
- course_id int foreign key references courses.id
- type varchar
- amount int
- transacted_at timestamp

3.6 payments:

- id int primary key
- user_id int foreign key references users.id
- amount int
- mode varchar
- xp_purchased int
- purchased_at timestamp

3.7 ER Diagram:

4.UI Implementation:

4.1 Overview:

The UI architecture is designed in Angular and follows a modular component-based approach. It uses services for API communication, guards for route protection, and state management to improve data flow and performance.

4.2 Component Structure:

- **App Module (Root):**
  - Shared Module (reusable components like button, headers)
- **Navbar Module:**
  - Bartr logo
  - Create a course button
  - Log in button (if user is not logged in)
  - Sign up button (if user is not logged in)
- **Auth Module:**
  - Login Component:
    - Username field
    - Password field
    - Forgot password button
    - Login button
    - Alternative login methods button
    - Register now button (if account is not present)
  - Register Component:
    - Full name field
    - Username field
    - Email field
    - Password field
    - Create account button
    - Alternative registration buttons (Google, Twitter)
    - Login button (if already a user)
  - Auth Service
- **Dashboard Module:**
  - User Profile Component:

- Username avatar image
- Username text
- Email text
- Name card with profile name text
- Contact card with contact text
- Xp card with xp amount text
- Country card with country text
  - Sidebar Component:
    - Personal Information button
    - Courses Enrolled button
    - Courses Created button
    - Profile Settings button
- **Course Module:**
  - Course List Document
  - Course Detail Component
  - Create Course Component
  - Enroll Course Component
- **XP Module:**
  - Purchase XP Component
  - XP History Component
- **Footer Module:**
  - Bartr logo text
  - Social media handle buttons
  - Basic Information texts

4.3 Routing Configuration:

- **Public Routes:**
  - /login: Login Component
  - /register: Register Component
- **Protected Routes:**
  - /profile: User Profile Component
  - /courses: Course List Component
  - /course/id: Course Detail Component

- o /create-course: Create Course Component
- o /purchase-xp: Purchase XP Component

4.4 UI Libraries Used:

- **Angular Material:**
  - o Buttons
  - o Form Fields
  - o Dialog
  - o Table
- **Ngx-toaster:**
  - o Alerts
  - o Feedback
- **Ngx-spinner:**
  - o Loading Indicators

4.5 Error Handling:

- Global Error handler for API Failures
- Route fallback (404 page)
- Loading and empty states for components

5. Backend Implementation:

5.1 Models (Entities / POJOs):

5.1.1 User:

Represents the user's core details and XP balance.

Fields: id, username, email, password, phone, fullname, avatarUrl, createdAt, xp

5.1.2 Course:

Represents a category/course available for barter.

Fields: id, title, description, skill, category (reference to Skill), creator (reference to User), createdAt, level

5.1.3 Category:

Defines course categories and the XP cost for each.

Fields: id, name, description, xpCost

5.1.4 Enrollment:

Represents which users have enrolled in which courses.

Fields: id, course (reference), learner (reference), enrollmentDate

5.1.5 Payment:

Logs real-money transactions to buy XP.

Fields: id, user (reference), amount, mode, xpPurchased, purchasedAt

5.1.6 Transaction:

Logs in-app XP movements for transparency.

Fields: id, user (reference), course (reference), type (XP Spend, Refund), amount, transactedAt

5.2 Services (Business Logic Layer):

5.2.1 UserService:

- Fetch, create, update user details
- Update XP balance
- Handle user-related business logic

5.2.2 CourseService:

- Create courses
- Fetch courses (all, by category, by creator)
- Course-related validations

5.2.3 EnrollmentService:

- Enroll users in courses
- Deduct XP from learners, credit XP to course creators
- Save enrollment data

5.2.4 PaymentService:

- Handle XP top-up logic
- Create payment records
- Update XP balance after payment verification

5.2.5 AppwriteService (Integration Layer):

- Wrap Appwrite SDK calls to create, update, delete, and read documents
- Map Appwrite JSON responses to Java models

5.3 Controllers (REST API Entry Points):

5.3.1 AuthController:

Register, login, return current user info.

Endpoints: /api/auth/*

5.3.2 UserController:

Fetch user profile, XP balance, user's created/enrolled courses.

Endpoints: /api/users/*

5.3.3 CourseController:

Course creation, course details, course listings by category.

Endpoints: /api/courses/*

5.3.4 EnrollmentController:

Enroll users in courses.

Endpoints: /api/courses/{id}/enroll

5.3.5 **PaymentController:**

Initiate payments, verify webhooks, log XP purchase.

Endpoints: /api/payments/*

5.4 Appwrite Integration (Database & Files):

Appwrite replaces direct DB (like JPA repositories).

Interactions:

- CreateDocument → insert new user, course, enrollment, etc.
- ListDocuments → fetch records for listing APIs.
- UpdateDocument → update XP or user profile fields.
- DeleteDocument → (if required for admin features).
- File APIs → store thumbnails, avatars, videos.

6. Validations:

The Bartr Skill Barter System implements robust validations at both the frontend (Angular) and backend (Spring Boot) layers to ensure data integrity, enhance security, and deliver a seamless user experience.

Validation Types

- Client-side (Angular): For immediate user feedback using reactive forms and validation messages.

- Server-side (Spring Boot): Final enforcement of rules before writing to the database or processing logic.
- Database-level (Appwrite): Schema validation where applicable.

Validation Rules by module:

- **User Registration and login:**

Field	Validation Rule	Message
Username	Required, min 3 characters	"username is required (min 3 characters)"
Email	Required, valid email format	"Enter a valid email address"
Password	Required, min 8 characters	"Password must be at least 8 characters long"
Phone	Numeric, 10-15 digits	"Enter a valid phone number"
Full Name	Required	"Full name is required"

- Course Creation:

Field	Validation Rule	Message
Title	Required, min 5, max 100 characters	"Course title is required (5-100 characters)"
Description	Required, min 10 characters	"Description must be at least 10 characters"
Skill/ Category	Must be selected from existing skill list	"Please select a valid skill category"
Level	One of ["Beginner", "Intermediate", "Advanced"]	"Select a valid difficulty level"
Video URL	Valid URL format (stored via Appwrite)	"Enter a valid course link"

- **XP Transaction (Course Enrollment):**

Field	Validation Rule	Message
User XP	Must be >= course XP cost	"Insufficient XP balance"
Duplicate Enroll	User must not be already enrolled in same course	"Already enrolled in same course"
Course ID	Must exist in DB	"Course not found"
Transaction Log	Log must be generated for each XP deduction/credit	"Transaction logging failed'

- **XP Purchase (Payment Integration):**

Field	Validation Rule	Message
XP Amount	Required numeric, >= minimum package (eg. 10xp)	"Enter a valid XP amount (minimum 10)"
Payment Status	Verified via Stripe/ Webhook	"Payment verification failed"
Order ID	Required during verification	"Missing order reference"
Payment Signature	Required and matched with HMAC validation	"Invalid payment signature"

Backend API Validations:

- @Valid annotations for DTOs
- @NotNull, @Size, @Email, @Min/@Max used for field constraints
- Exception handlers for MethodArgumentNotValidException
- Custom validation: XPTransferValidator, CourseOwnershipValidator

Frontend Form Validations (Angular):

- Angular reactive forms with FormBuilder and Validators
- Error messages displayed on blur or submit
- Fields disabled until all validations are met

- Use of Angular Material mat-error messages

Database Validations (Appwrite):

- Required fields configured via Appwrite's document schema
- Unique constraints: email, username
- Referential integrity: creator_id → users, skill_id → skills

7. API Design:

7.1 Authentication APIs:

Base Path: /api/auth

Method Endpoint Request Body Response Description POST /register {username, email, password } 201 Created or 400 Error Registers a new user POST /login {email, password } { token, userInfo } Logs in user, returns JWT token GET /me Authorization: Bearer token { user details } Returns current logged-in user.

7.2 User APIs:

Base Path: /api/users

Method Endpoint Auth Description GET /{id} Yes Fetch user profile GET /{id}/xp Yes Fetch user's XP balance GET /{id}/courses Yes Fetch courses created by user GET /{id}/enrolled Yes Fetch courses user is enrolled in.

7.3 Course APIs:

Base Path: /api/courses

Method Endpoint Auth Request Body Response Description GET / No - List of all courses Get all available courses GET /{id} No - Course detail Get details of a specific course POST / Yes { title, description, categoryId } Course created Create a new course POST /{id}/enroll Yes { userId } Enrollment response Enroll

in a course if XP is sufficient GET /category/{name} No - Courses list Get courses filtered by category name

7.4 Categories APIs:

Base Path: /api/categories

Method Endpoint Auth Description GET / No List all available skill categories and XP values

7.5 XP & Transactions APIs:

Base Path: /api/xp

Method Endpoint Auth Description GET /transactions Yes Get current user's XP transaction history POST /transfer Yes Transfer XP between users (internal use).

7.6 Payment APIs:

Base Path: /api/payments

Method Endpoint Auth Request Body Description POST /create Yes { userId, xpAmount } Initiates Razorpay payment and returns orderId POST /verify Verifies payment and updates XP GET /history Yes - Returns all payment history for user

7.7 Status Codes:

200 OK – Request successful

201 Created – Resource created

400 Bad Request – Invalid input

401 Unauthorized – Auth failure

403 Forbidden – Insufficient permissions

404 Not Found – Resource does not exist

500 Internal Server Error – Unexpected issue

8. Services: