Low Level Design Document

Introduction

This Low Level Design (LLD) document outlines the core components and implementation details for **SafePass - Simple Auth API**. The project provides a backend authentication API with user registration, password hashing, JWT-based login, and protected routes, using Node.js, Express, and MongoDB.

---

1. System Components

Component	Description	Key Responsibilities
API Server	Express-based HTTP server	Route handling, middleware, error handling
Auth Controller	Handles auth logic	Register, login, JWT issuance
User Model	Mongoose schema/model for users	User data storage, password hash
Middleware	Auth & error middleware	JWT verification, error responses
Database	MongoDB	Persistent user data

---

2. Class/Interface Overview

Class/Module	Key Methods/Attributes	Relationships
User (Mongoose)	`email`, `passwordHash`, timestamps	Used by Auth Controller
AuthController	`register()`, `login()`	Uses User model, JWT, bcrypt
AuthMiddleware	`authenticateJWT()`	Used in protected routes

Key Methods (Pseudocode):

```
// AuthController
register(req, res)
  - Validate input
  - Hash password
  - Save user
  - Return success/error

login(req, res)
  - Validate input
  - Find user
  - Compare password
  - Issue JWT
  - Return token/error

// AuthMiddleware
authenticateJWT(req, res, next)
```

```
  - Verify JWT
  - Attach user to req
  - Call next() or return 401
```

---

3. Data Structure Overview

Field	Type	Description
_id	ObjectId	MongoDB unique identifier
email	String	User email (unique)
passwordHash	String	Hashed password
createdAt	Date	Timestamp
updatedAt	Date	Timestamp

---

4. Algorithms / Logic

User Registration Flow:

```
If email & password valid:
    If user exists:
        Return 409 Conflict
    Hash password (bcrypt)
    Save user to DB
    Return 201 Created
Else:
    Return 400 Bad Request
```

Login Flow:

```
If email & password valid:
    Find user by email
    If not found:
        Return 401 Unauthorized
    Compare password (bcrypt)
    If match:
        Issue JWT (userId, expiry)
        Return token
    Else:
        Return 401 Unauthorized
Else:
    Return 400 Bad Request
```

Protected Route:

```
Extract JWT from header
If valid:
    Attach user to request
    Proceed
Else:
    Return 401 Unauthorized
```

---

5. Error Handling

Scenario	Response Code	Handling/Message
Invalid input	400	“Invalid email or password”
User already exists (register)	409	“User already exists”
User not found / wrong password	401	“Invalid credentials”
JWT missing/invalid	401	“Unauthorized”
Server/database error	500	“Internal server error”

End of Document