**Tài liệu đã tạo:**

1. **📊 Test Plan Document** (

 docs/TEST\_PLAN.md

)

* + Scope và objectives cho authentication & display functions
  + Test strategy và environment setup
  + Test cases chi tiết cho login/logout, shop display, item display
  + Entry/exit criteria và success metrics

1. **⚫ Blackbox Test Cases** (

 docs/BLACKBOX\_TEST\_CASES.md

)

* + Test cases cho user registration, login, logout
  + Shop listing và search functionality
  + Item display và filtering
  + Security và performance test cases

1. **⚪ Whitebox Test Cases** (

 docs/WHITEBOX\_TEST\_CASES.md

)

* + Unit tests cho password hashing, JWT tokens
  + Database model operations
  + API controller logic testing
  + Frontend component testing với coverage targets

1. **🛠️ Environment Setup Guide** (

 docs/ENVIRONMENT\_SETUP\_GUIDE.md

)

* + Cài đặt Node.js, Docker, Git từ đầu
  + Project setup và environment variables
  + Database setup với MongoDB
  + IDE configuration và verification steps

1. **🧪 Unit Testing Guide** (

 docs/UNIT\_TESTING\_GUIDE.md

)

* + Setup Jest (Backend) và Vitest (Frontend)
  + Viết unit tests cho authentication utils
  + Component testing với Testing Library
  + Coverage analysis và best practices

1. **🔗 Integration Testing Guide** (

 docs/INTEGRATION\_TESTING\_GUIDE.md

)

* + API endpoint integration tests
  + Database integration testing
  + Frontend-Backend communication tests
  + Model relationship testing

1. **🚀 CI/CD Implementation Guide** (

 docs/CI\_CD\_IMPLEMENTATION\_GUIDE.md

)

* + GitHub Actions workflows setup
  + Docker configuration cho testing
  + Quality gates và coverage requirements
  + Deployment pipeline và monitoring

Sau khi bạn xong ENVIRONMENT\_SETUP\_GUIDE.md, hãy làm theo thứ tự sau với 3 tài liệu còn lại:

1. UNIT\_TESTING\_GUIDE.md – Thiết lập và chạy Unit Test

* Mục cần làm:
  + 1.1–1.3: Cài dependencies, tạo jest.config.js (Backend đã có), tạo BackEnd/tests/setup.js (đã có).
  + 3.1–3.3: Cài thư viện test cho Frontend và tạo src/tests/setup.js.
* Lệnh chạy nhanh:
  + Backend:
    - cd BackEnd
    - npm test
    - npm run test:coverage
  + Frontend:
    - cd FrontEnd
    - npm install -D @testing-library/react @testing-library/jest-dom @testing-library/user-event jsdom
    - Tạo file FrontEnd/src/tests/setup.js (nếu bạn muốn, tôi sẽ tạo giúp)
    - npm test

1. INTEGRATION\_TESTING\_GUIDE.md – Test tích hợp API/DB và FE↔BE

* Mục cần làm:
  + 2.1: Tạo BackEnd/tests/integration/setup.js (kết nối MongoMemoryServer).
  + 2.2–2.3: Viết các test tích hợp cho Auth/Shop (Supertest gọi API thật của app).
  + 3.1–3.2: FE integration test cho services/components có gọi API (mock fetch).
* Lệnh chạy nhanh:
  + Backend:
    - cd BackEnd
    - npm test -- tests/integration
    - npm run test:coverage -- tests/integration
  + Frontend:
    - cd FrontEnd
    - npm test -- --run src/\*\*/\*.integration.test.jsx

1. DEPLOYMENT\_GUIDE.md – Triển khai Staging/Production

* Dành cho giai đoạn sau khi CI xanh.
* Lộ trình đề xuất:
  + Staging:
    - Sử dụng .github/workflows/deploy-staging.yml (mục 4.1) + Railway (4.2) hoặc Vercel cho FE (4.3).
    - Thêm secrets như RAILWAY\_TOKEN, STAGING\_URL.
  + Production:
    - Chọn 1 phương án: AWS ECS (5.1) hoặc Kubernetes (5.2).
    - Thiết lập MongoDB Atlas (6.1), SSL (7), Monitoring (8), Backup (9), Rollback (10).

Đề xuất thứ tự thực hiện tổng thể

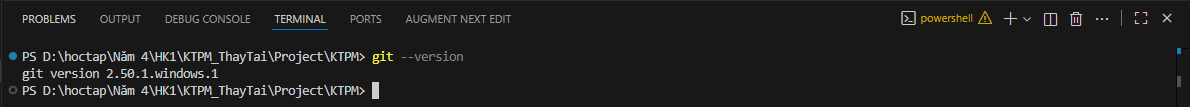
* B1: UNIT\_TESTING\_GUIDE – Hoàn tất unit test Backend trước, rồi Frontend.
* B2: INTEGRATION\_TESTING\_GUIDE – Viết và chạy test tích hợp API/DB; sau đó FE integration test.
* B3: CI/CD (đã có roadmap trong CI\_CD\_IMPLEMENTATION\_GUIDE.md) – Thêm job chạy unit/integration tests.
* B4: DEPLOYMENT\_GUIDE – Thiết lập Staging; khi ổn định thì làm Production.

**RUN ENVIRONMENT\_SETUP\_GUIDE**

**Install Node.js và npm**  
**A black screen with a black background

AI-generated content may be incorrect.**

**Install Git**



**Install Docker**

A black screen with white text

AI-generated content may be incorrect.

Create development branch

A screenshot of a computer

AI-generated content may be incorrect.

**Install Dependencies**

cd BackEnd

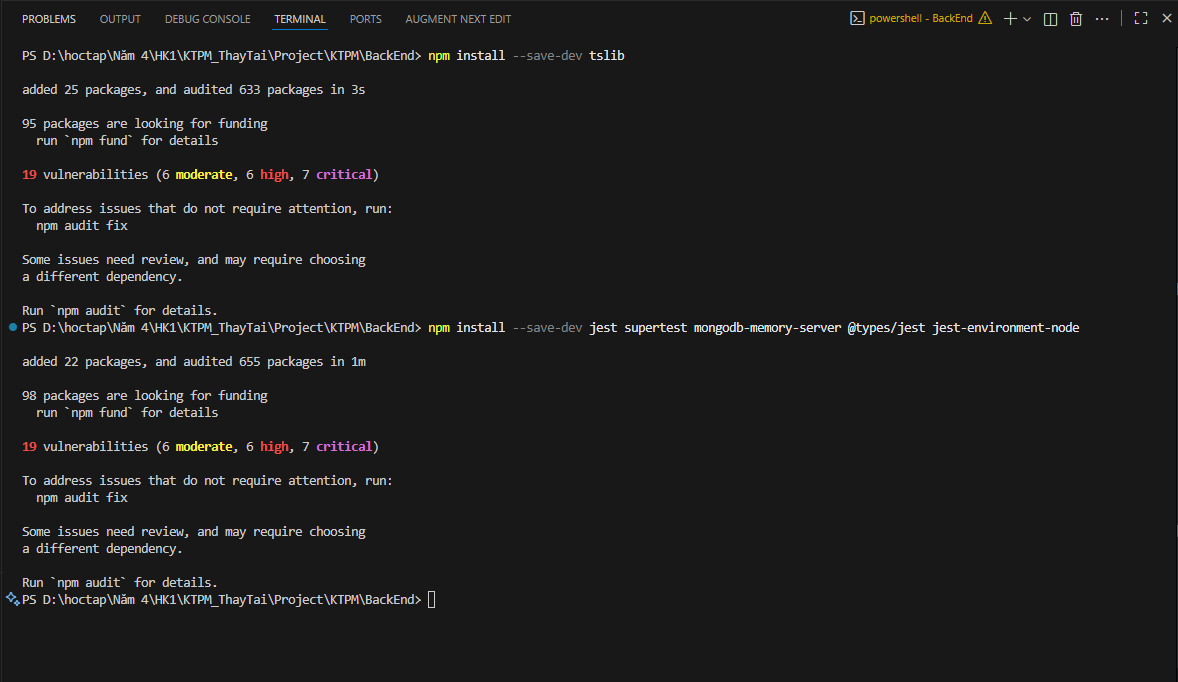
# Install production dependencies

npm install

# Install additional testing dependencies

npm install --save-dev jest supertest mongodb-memory-server @types/jest jest-environment-node

| **Gói Thư Viện** | **Mục Đích Chính** | **Vai Trò Trong Kiểm Thử** |
| --- | --- | --- |
| **jest** | **Framework kiểm thử** hàng đầu cho JavaScript. | Cung cấp bộ công cụ để chạy, tổ chức, và viết các bài kiểm thử (Unit/Integration). |
| **supertest** | Giúp **kiểm thử HTTP assertions** dễ dàng. | Được sử dụng để gửi các HTTP request đến API Express.js (Backend) và kiểm tra response (status code, body, headers). Cần thiết cho **Integration Tests** API. |
| **mongodb-memory-server** | Cung cấp một phiên bản **MongoDB ảo** chỉ chạy trong bộ nhớ (RAM). | Được sử dụng để tạo một database tạm thời, **sạch sẽ và độc lập** cho mỗi lần chạy kiểm thử mà không cần cài đặt MongoDB vật lý. Giúp tăng tốc độ và độ tin cậy của Unit/Integration Tests. |
| **@types/jest** | **Type definitions** cho Jest. | Cần thiết nếu dự án sử dụng TypeScript (cung cấp gợi ý code và kiểm tra kiểu). |
| **jest-environment-node** | Định cấu hình môi trường chạy test. | Đảm bảo rằng Jest chạy trong môi trường **Node.js** (chứ không phải trình duyệt), phù hợp cho việc kiểm thử Backend. |



**Database Setup**

**3.1. MongoDB với Docker (Cho Phát Triển và Integration Tests)**

**# Start MongoDB container**

**docker run -d --name ktpm-mongodb-test -p 27017:27017 -e MONGO\_INITDB\_ROOT\_USERNAME=admin -e MONGO\_INITDB\_ROOT\_PASSWORD=password -e MONGO\_INITDB\_DATABASE=ktpm\_test mongo:7.0**

Mục đích của bước này là cung cấp một **môi trường cơ sở dữ liệu thực tế** và **ổn định** để bạn phát triển ứng dụng và chạy các bài kiểm thử tích hợp (Integration Tests).

| **Lệnh** | **Mục Đích** | **Chi Tiết** |
| --- | --- | --- |
| docker run -d --name ktpm-mongodb-test ... | **Khởi động Container MongoDB**. |  |
| -d | Chạy container ở chế độ **detached** (nền). |  |
| --name ktpm-mongodb-test | Đặt tên cho container để dễ quản lý. |  |
| -p 27017:27017 | **Ánh xạ cổng** (Port Mapping). | Mở cổng 27017 trên máy tính cục bộ của bạn để kết nối với cổng 27017 bên trong container MongoDB. |
| -e MONGO\_INITDB\_... | **Thiết lập biến môi trường**. | Tạo người dùng admin và đặt tên database khởi tạo là ktpm\_test. |
| mongo:7.0 | **Image Docker** được sử dụng. | Kéo và chạy phiên bản MongoDB 7.0. |
| docker ps | **Kiểm tra trạng thái**. | Xác minh rằng container đang chạy. |
| docker logs ktpm-mongodb-test | **Xem nhật ký**. | Kiểm tra xem MongoDB đã khởi động thành công bên trong container hay chưa. |

A screenshot of a computer program

AI-generated content may be incorrect.

| **Mục Đích** | **Kết Quả trong Ảnh** | **Ý Nghĩa** |
| --- | --- | --- |
| **Pull Image** | Pulling from library/mongo... Pull complete | Docker đã không tìm thấy image mongo:7.0 cục bộ, nên đã **tự động tải** image này từ Docker Hub (library/mongo) thành công. |
| **Status** | Status: Downloaded newer image for mongo:7.0 | Xác nhận việc tải image thành công. |
| **Container ID** | 08ae...7f83f1 | Đây là Container ID **đã được khởi tạo** thành công ở chế độ nền (-d). |
| **Lỗi** | Không có thông báo lỗi docker run nào xuất hiện. | Lệnh chạy container đã được thực thi thành công. |

**Verify MongoDB is running**

docker ps

docker logs ktpm-mongodb-test

**3.2. MongoDB Memory Server (Cho Unit Tests)**

Mục đích của phần này là cung cấp một **database ảo, nhanh chóng và độc lập** cho các bài **Unit Tests**. Đã cài đặt ở bước trước. Sẽ tự động start/stop trong test files

| **Đặc Điểm** | **Mô Tả** | **Lợi Ích** |
| --- | --- | --- |
| **Database trong bộ nhớ (RAM)** | mongodb-memory-server sẽ tải một phiên bản MongoDB nhị phân và chạy nó hoàn toàn trong RAM của máy tính. | Tốc độ cực nhanh vì không cần ghi/đọc từ ổ đĩa. |
| **Tự động start/stop** | Server sẽ được khởi tạo trong hook beforeAll và dừng lại trong afterAll hoặc afterEach của Jest. | Đảm bảo mỗi bài kiểm thử có một **database sạch** và độc lập, tránh xung đột dữ liệu giữa các test. |
| **Không cần Docker** | Server này hoạt động độc lập với Docker. | Giúp các bài Unit Test có thể chạy mà không cần Docker hoặc bất kỳ dịch vụ database nào khác đang chạy nền. |

**4. Testing Framework Setup ( tương tự mục 4 trong ENVIRONMENT\_SETUP\_GUIDE.md):**

**4.1. Jest Configuration (Backend)**

| **File** | **Mục đích** | **Vai trò chính** |
| --- | --- | --- |
| BackEnd/jest.config.js | **Cấu hình Jest**. | Định nghĩa cách Jest hoạt động, bao gồm: |
|  |  | \* **testEnvironment: 'node'**: Chỉ định môi trường chạy test là Node.js, phù hợp cho Backend. |
|  |  | \* **setupFilesAfterEnv**: Chỉ định file thiết lập database/môi trường chạy trước khi các test file được thực thi. |
|  |  | \* **collectCoverageFrom**: Xác định những file nào cần thu thập dữ liệu coverage (controllers, models, utils, middlewares). |
|  |  | \* **forceExit, clearMocks**: Đảm bảo môi trường test được dọn dẹp sạch sẽ sau mỗi lần chạy. |

4.2. Vitest Configuration (Frontend)

| **File** | **Mục đích** | **Vai trò chính** |
| --- | --- | --- |
| FrontEnd/vite.config.js | **Cấu hình Vitest** (thông qua file cấu hình Vite). | Định nghĩa cách Vitest hoạt động cho Frontend React: |
|  |  | \* **environment: 'jsdom'**: Mô phỏng môi trường DOM của trình duyệt (cần thiết cho React Component Testing). |
|  |  | \* **globals: true**: Cho phép sử dụng các hàm test toàn cục (describe, test) mà không cần import. |
|  |  | \* **setupFiles**: Chỉ định file thiết lập môi trường giả lập (mock) trình duyệt. |
|  |  | \* **coverage**: Thiết lập định dạng báo cáo coverage (text, json, html). |

Các bước trong mục **"4. Testing Framework Setup"** này nhằm mục đích **thiết lập và cấu hình hoàn chỉnh** môi trường kiểm thử **Unit Testing** và **Integration Testing** cho cả Backend (sử dụng Jest) và Frontend (sử dụng Vitest) của dự án.

Nó là bước then chốt để đảm bảo các bài kiểm thử có thể chạy tự động, độc lập và cung cấp báo cáo coverage chính xác.

**⚙️ Chi Tiết Về Mục Đích Của Từng Bước**

**4.1. Jest Configuration (Backend)**

| **File** | **Mục đích** | **Vai trò chính** |
| --- | --- | --- |
| BackEnd/jest.config.js | **Cấu hình Jest**. | Định nghĩa cách Jest hoạt động, bao gồm: |
|  |  | \* **testEnvironment: 'node'**: Chỉ định môi trường chạy test là Node.js, phù hợp cho Backend. |
|  |  | \* **setupFilesAfterEnv**: Chỉ định file thiết lập database/môi trường chạy trước khi các test file được thực thi. |
|  |  | \* **collectCoverageFrom**: Xác định những file nào cần thu thập dữ liệu coverage (controllers, models, utils, middlewares). |
|  |  | \* **forceExit, clearMocks**: Đảm bảo môi trường test được dọn dẹp sạch sẽ sau mỗi lần chạy. |

**4.2. Vitest Configuration (Frontend)**

| **File** | **Mục đích** | **Vai trò chính** |
| --- | --- | --- |
| FrontEnd/vite.config.js | **Cấu hình Vitest** (thông qua file cấu hình Vite). | Định nghĩa cách Vitest hoạt động cho Frontend React: |
|  |  | \* **environment: 'jsdom'**: Mô phỏng môi trường DOM của trình duyệt (cần thiết cho React Component Testing). |
|  |  | \* **globals: true**: Cho phép sử dụng các hàm test toàn cục (describe, test) mà không cần import. |
|  |  | \* **setupFiles**: Chỉ định file thiết lập môi trường giả lập (mock) trình duyệt. |
|  |  | \* **coverage**: Thiết lập định dạng báo cáo coverage (text, json, html). |

**4.3. Test Setup Files (Môi trường ảo)**

Phần này tạo ra các file thiết lập cần thiết để cô lập môi trường test, đảm bảo tính **độc lập** và **tái tạo** của các bài kiểm thử.

**Backend Test Setup (BackEnd/tests/setup.js)**

File này thiết lập **Database ảo cho Backend Tests**:

* Sử dụng **mongodb-memory-server** để tạo một instance MongoDB tạm thời, chỉ chạy trong bộ nhớ (RAM).
* Sử dụng các hooks beforeAll, afterAll, và afterEach để:
  + **Khởi tạo** kết nối database trước tất cả các test (beforeAll).
  + **Đóng và dừng** database sau tất cả các test (afterAll).
  + **Xóa sạch** dữ liệu trong tất cả các collection sau *mỗi* test (afterEach), đảm bảo các test độc lập với nhau.

**Frontend Test Setup (FrontEnd/src/tests/setup.js)**

File này thiết lập **môi trường trình duyệt ảo** cho Frontend Tests:

* Chứa các **mocks** cho các API trình duyệt không có sẵn trong môi trường Node.js/JSDOM.
* **window.matchMedia**: Mock cho các truy vấn media (thường được dùng bởi các thư viện UI như Ant Design, Material UI).
* **global.IntersectionObserver**: Mock cho API theo dõi khả năng hiển thị của phần tử (thường dùng cho lazy loading).
* import '@testing-library/jest-dom' giúp mở rộng các hàm expect trong Vitest để dễ dàng kiểm tra DOM (ví dụ: toBeInTheDocument()).

**5.Setup github**

**6. Docker Development Environment**

**6.1 Start Development Environment (Sử Dụng Docker)**

Phần này tập trung vào việc khởi động toàn bộ kiến trúc (Backend, Frontend, Database) bằng Docker Compose, mô phỏng gần nhất với môi trường Production/Staging

| **Lệnh** | **Mục đích** | **Lợi ích** |
| --- | --- | --- |
| docker-compose up -d | **Khởi động tất cả dịch vụ** (Backend, Frontend, MongoDB). | Chạy tất cả các dịch vụ trong các container độc lập ở chế độ nền (-d). |
| docker-compose ps | **Kiểm tra trạng thái dịch vụ**. | Xác minh rằng tất cả các container đều đang ở trạng thái Up (đang chạy). |
| docker-compose logs -f [service] | **Xem nhật ký (logs)** theo thời gian thực. | Giúp bạn theo dõi quá trình khởi động và gỡ lỗi (debug) cho từng dịch vụ (backend hoặc frontend). |

**6.2 Development Workflow (Làm Việc Hàng Ngày)**

Phần này cung cấp các tùy chọn để chạy ứng dụng và các bài kiểm thử, phù hợp cho quy trình làm việc hàng ngày của lập trình viên.

Khởi chạy Server Phát Triển

| **Lệnh** | **Mục đích** | **Khi nào sử dụng** |
| --- | --- | --- |
| cd BackEnd && npm run dev & | Khởi động **Backend server cục bộ**. | Khi bạn muốn chỉnh sửa code và thấy kết quả thay đổi ngay lập tức (thường có Hot Reload). Yêu cầu phải có MongoDB Docker (hoặc dịch vụ database khác) đang chạy. |
| cd FrontEnd && npm run dev & | Khởi động **Frontend server cục bộ**. | Tương tự, để phát triển giao diện người dùng với Hot Module Replacement. |
| docker-compose up | **Khởi động mọi thứ bằng Docker**. | Thay thế cho các lệnh npm run dev cục bộ nếu bạn muốn cả Backend và Frontend chạy bên trong container, mô phỏng môi trường giống CI/CD hơn. |

Chạy Kiểm Thử (Testing)

| **Lệnh** | **Mục đích** | **Vai trò** |
| --- | --- | --- |
| cd BackEnd && npm test | **Chạy tất cả Unit/Integration Tests** cho Backend. | Đảm bảo logic và API Backend hoạt động chính xác trước khi commit code. |
| cd FrontEnd && npm test | **Chạy tất cả Unit/Integration Tests** cho Frontend. | Kiểm tra các components và logic phía Client. |
| npm run test:watch | **Chạy test Backend ở chế độ Watch**. | Tự động chạy lại các bài test liên quan mỗi khi bạn lưu file, giúp phát triển dựa trên test (TDD). |
| npm run test:ui | **Chạy test Frontend ở chế độ UI**. | Cung cấp giao diện đồ họa (web interface) để xem kết quả test, debug dễ dàng hơn. |

**7. IDE Configuration**

Tạo các file setting config như hướng dẫn

**8. Verification Steps**

Quy trình **xác minh** bao gồm kiểm tra tài liệu, thiết kế, mã và chương trình, trong khi quy trình Xác thực bao gồm kiểm tra và xác nhận sản phẩm thực tế. Việc xác minh không liên quan đến việc thực thi mã, trong khi Xác thực liên quan đến việc thực thi mã.

**8.1 Environment Verification**

A screenshot of a computer

AI-generated content may be incorrect.

**1. Backend Unit Testing Setup (Thiết lập Unit Test cho Backend)**

Mục này hướng dẫn bạn cài đặt môi trường và các công cụ cần thiết để có thể viết và chạy test cho phần Backend (Node.js/Express).

* **1.1 Install Testing Dependencies (Cài đặt các dependency cho test)**:
  + Bạn sẽ cài đặt jest (framework chính để chạy test), supertest (công cụ để giả lập và gửi request HTTP đến API của bạn, giống như Postman nhưng tự động).
  + mongodb-memory-server là một công cụ cực kỳ quan trọng: nó tạo ra một database MongoDB ảo, chạy hoàn toàn trên RAM. Điều này giúp test chạy rất nhanh và đảm bảo mỗi lần test đều bắt đầu với một database "sạch", không ảnh hưởng đến database thật của bạn.
  + Các lệnh npm pkg set dùng để thêm các script (lệnh tắt) vào file package.json của bạn, như npm test để chạy test, npm run test:watch để tự động chạy lại test khi bạn sửa code.
* **1.2 Create Jest Configuration (Tạo file cấu hình Jest)**:
  + Lệnh này tạo file jest.config.js. Đây là file "luật chơi" cho Jest, nói cho nó biết:
    - testEnvironment: 'node': Chạy test trong môi trường Node.js (cho Backend).
    - setupFilesAfterEnv: Chỉ định file tests/setup.js sẽ tự động chạy trước tất cả các file test.
    - testMatch: Tìm các file test ở đâu (bất kỳ file nào có đuôi .test.js trong thư mục tests/).
    - collectCoverageFrom: Khi đo "độ bao phủ" (coverage), Jest nên phân tích những file nào (ví dụ: controllers, models) và bỏ qua những file nào (node\_modules).
    - forceExit: Đảm bảo Jest thoát hoàn toàn sau khi chạy xong, rất hữu ích khi có các kết nối (như database) còn treo.
* **1.3 Create Test Setup File (Tạo file cài đặt test)**:
  + Đây là file tests/setup.js sẽ chạy trước mọi thứ.
  + **beforeAll**: Chạy *một lần duy nhất* trước khi tất cả các test bắt đầu. Lệnh này khởi động MongoMemoryServer (DB ảo) và kết nối Mongoose vào đó.
  + **afterAll**: Chạy *một lần duy nhất* sau khi tất cả các test kết thúc. Lệnh này dọn dẹp bằng cách xóa DB, ngắt kết nối Mongoose và tắt DB ảo.
  + **afterEach**: Chạy *sau mỗi bài test (mỗi test case)*. Lệnh này xóa sạch dữ liệu trong tất cả các collection. Điều này cực kỳ quan trọng để đảm bảo **tính cô lập** (isolation) – test A không bị ảnh hưởng bởi dữ liệu rác do test B tạo ra.

**2. Writing Backend Unit Tests (Viết Unit Tests cho Backend)**

Mục này hướng dẫn bạn viết các bài test cụ thể cho từng phần của logic Backend.

* **2.1 Authentication Utils Tests (Test các hàm tiện ích xác thực)**:
  + Mục này test các hàm đơn lẻ, "thuần túy" trong utils/auth.js.
  + Nó kiểm tra xem hashPassword có thực sự băm mật khẩu không, comparePassword có so sánh đúng không, và generateToken (JWT) có tạo ra token hợp lệ không. Đây là các unit test cổ điển: kiểm tra input và output.
* **2.2 User Model Tests (Test cho Model User)**:
  + Mục này test trực tiếp Mongoose Model của bạn.
  + Nó kiểm tra các quy tắc bạn định nghĩa trong Schema: User.create có thành công không? Nếu tạo 2 user trùng email (duplicate key error) thì có báo lỗi không? Nếu thiếu trường email thì có báo lỗi không?
* **2.3 Authentication Controller Tests (Test cho Controller xác thực)**:
  + Đây là các bài test *tích hợp (integration test)*, không phải unit test thuần túy, vì nó kiểm tra luồng đi của một request HTTP qua nhiều lớp (route -> controller -> model).
  + Nó dùng supertest để gửi request POST thật đến /api/auth/register và /api/auth/login.
  + Nó kiểm tra *HTTP status code* (ví dụ: expect(201) - Created) và *nội dung JSON* trả về (response.body.success). Đây là cách test API hiệu quả nhất.

**3. Frontend Unit Testing Setup (Thiết lập Unit Test cho Frontend)**

Mục này chuyển sang phần Frontend và cài đặt các công cụ để test component React của bạn.

* **3.1 Install Testing Dependencies (Cài đặt dependency cho test)**:
  + Cài đặt vitest (framework test nhanh dành cho Vite).
  + @testing-library/react: Bộ thư viện "vàng" để test React. Nó giúp bạn render component trong một môi trường ảo và tương tác với chúng như một người dùng thật (tìm nút, click, gõ chữ).
  + jsdom: Tạo ra một môi trường trình duyệt "giả" (fake browser) ngay trong terminal để React có thể render.
  + @vitest/ui: Tạo ra một giao diện web đẹp mắt để bạn xem kết quả test.
* **3.2 Update Vite Configuration (Cập nhật cấu hình Vite)**:
  + Bạn cập nhật file vite.config.js để tích hợp vitest vào.
  + globals: true: Cho phép Vitest sử dụng các hàm test (như test, expect) ở mọi nơi mà không cần import.
  + environment: 'jsdom': Sử dụng jsdom (trình duyệt giả).
  + setupFiles: './src/tests/setup.js': Chỉ định file setup cho frontend.
* **3.3 Create Test Setup File (Tạo file cài đặt test cho Frontend)**:
  + File src/tests/setup.js này chạy trước các test của React.
  + Nó "mock" (giả lập) các API của trình duyệt mà jsdom không có, ví dụ window.matchMedia hoặc IntersectionObserver, để đảm bảo các thư viện component (như Ant Design, Chakra UI) không bị lỗi khi chạy test.

**4. Writing Frontend Unit Tests (Viết Unit Tests cho Frontend)**

Mục này hướng dẫn viết test cho các component React.

* **4.1 Login Component Tests (Test component LoginForm)**:
  + Test này kiểm tra xem component LoginForm có render đúng các ô input (Email, Password) và nút "Login" không (expect(screen.getByLabelText(/email/i)).toBeInTheDocument()).
  + Nó giả lập hành vi người dùng: userEvent.type (gõ chữ vào ô input), user.click (nhấn nút).
  + Nó kiểm tra các trạng thái: Khi click mà chưa điền gì thì có báo lỗi validation không? Khi đăng nhập thất bại có hiện thông báo lỗi không? Khi đang loading thì nút Login có bị vô hiệu hóa (disabled) không?
* **4.2 Shop List Component Tests (Test component ShopList)**:
  + Test này kiểm tra component ShopList dựa trên các props đầu vào.
  + Nó kiểm tra các kịch bản: Khi loading={true} thì có hiện spinner không? Khi shops={[]} (mảng rỗng) thì có hiện thông báo "No shops found" không? Khi có data (mockShops) thì có render đúng tên các cửa hàng không?

**5. Running Tests (Chạy Tests)**

Mục này cung cấp các lệnh (đã được định nghĩa trong package.json) để bạn thực thi các bài test của mình.

* npm test: Chạy tất cả test một lần.
* npm run test:watch: Chế độ "canh" – tự động chạy lại test khi bạn lưu file.
* npm run test:coverage: Chạy test và tạo báo cáo "độ bao phủ" (xem Mục 6).
* npm test -- auth.test.js: Chỉ chạy một file test cụ thể (rất hữu ích khi debug).

**6. Test Coverage Analysis (Phân tích Độ bao phủ Test)**

* **6.1 Coverage Reports (Báo cáo độ bao phủ)**:
  + Sau khi chạy npm run test:coverage, một thư mục coverage/ sẽ được tạo ra. Mở file index.html trong đó, bạn sẽ thấy một báo cáo chi tiết, chỉ rõ file nào đã được test, dòng code nào đã được chạy, và dòng nào còn "sót" (màu đỏ).
* **6.2 Coverage Targets (Mục tiêu độ bao phủ)**:
  + Đây là các mục tiêu chất lượng mà dự án đặt ra (ví dụ: 90% các dòng code phải được test). Đây là một thước đo quan trọng để đánh giá chất lượng test.

**7. Best Practices (Các Quy tắc Tốt nhất)**

Mục này đưa ra các lời khuyên chung để viết test tốt, dễ đọc và dễ bảo trì.

* **Test Structure (Cấu trúc Test) - AAA Pattern**: Đây là quy tắc vàng:
  1. **Arrange (Sắp đặt)**: Chuẩn bị dữ liệu, mock các hàm cần thiết.
  2. **Act (Hành động)**: Gọi hàm, chạy API, hoặc click nút (hành động chính cần test).
  3. **Assert (Khẳng định)**: Dùng expect để kiểm tra xem kết quả có đúng như mong đợi không.
* **Mocking Guidelines (Hướng dẫn Mocking)**: Khuyên bạn nên "mock" (giả lập) các thành phần bên ngoài (như database, API bên thứ ba) và chỉ tập trung test logic của chính đơn vị (unit) bạn đang viết.

**-Trong quá trình chạy npm test trên backend ta sửa được 1 số lỗi sau :**

Dưới đây là tổng hợp theo trình tự các lỗi gặp phải từ “checkpoint 23” (lúc đầu npm test) đến khi toàn bộ backend test pass, kèm thay đổi và giải thích:

1) Jest không hiểu import (ESM) trong file setup

- Lỗi:

- SyntaxError: Cannot use import statement outside a module tại tests/setup.js

- Nguyên nhân: Dự án dùng ESM ("type": "module"), nhưng Jest mặc định không load file setup ESM.

- Sửa:

- Tạo tests/setup.mjs và trỏ jest.config.js sang setup.mjs.

- Thêm cấu hình ESM vào jest.config.js (ban đầu dùng extensionsToTreatAsEsm: ['.js'] nhưng Jest 30 cảnh báo vì .js đã được suy ra từ package.json).

- Kết quả: Hết lỗi setup ban đầu nhưng tiếp tục gặp lỗi khác liên quan ESM trong test.

2) Vẫn lỗi ESM ở test files (import/export trong .test.js)

- Lỗi:

- “Cannot use import statement outside a module” trong các file test.

- Sửa (chuyển hướng an toàn):

- Dùng setup CommonJS cho Jest để giảm rủi ro ESM: tạo tests/setup.cjs.

- Cập nhật jest.config.js dùng setupFilesAfterEnv: ['<rootDir>/tests/setup.cjs'].

- Cài Babel cho Jest để transform ESM trong test: npm i -D babel-jest @babel/preset-env @babel/core.

- Thêm babel.config.cjs với preset-env (targets: node current).

- Sửa jest.config.js: transform: { '^.+\\.[jt]sx?$': 'babel-jest' }.

- Giải thích: babel-jest giúp Jest hiểu import/export trong test .js khi dự án là ESM.

3) Lỗi “Cannot find module '../../utils/auth.js'”

- Lỗi:

- tests/utils/auth.test.js và tests/controllers/auth.test.js import ../../utils/auth.js nhưng file chưa tồn tại.

- Sửa:

- Tạo utils/auth.js với các hàm:

- hashPassword, comparePassword (dùng bcryptjs)

- generateToken, verifyToken (dùng jsonwebtoken; fallback JWT\_SECRET test)

- Giải thích: Tạo đúng module được test mong đợi.

4) Lỗi validate Mongoose trong test model

- Lỗi:

- ValidationError: thiếu fullName, mobile, role ở User model khi test tạo user (do schema yêu cầu).

- Sửa:

- Cập nhật tests/models/user.test.js:

- Thêm fullName, mobile, role khi tạo user.

- Cập nhật tests/controllers/auth.test.js:

- Đổi name → fullName, phone → mobile, thêm role khi seed user.

- Giải thích: Đồng bộ test data với schema thực tế (User yêu cầu các trường bắt buộc).

5) Xung đột kết nối MongoDB khi chạy test controller

- Lỗi:

- MongooseError: Can't call openUri() on an active connection... (connect nhiều lần).

- Nguyên nhân: index.js khởi động server và connect DB khi chạy test (Jest đã tự kết nối in-memory trong setup).

- Sửa:

- Sửa index.js: chỉ app.listen và connectDB nếu NODE\_ENV !== 'test'. Export default app để supertest dùng.

- Tăng cường setup.cjs: set process.env.NODE\_ENV='test' sớm.

- Giải thích: Test dùng MongoMemoryServer riêng, không nên để app tự connect đến DB ngoài.

6) Lỗi 404 cho các route auth trong test controller

- Lỗi:

- expected 201/200/401… got 404 "Not Found".

- Nguyên nhân: test gọi /api/auth/register và /api/auth/login nhưng routes đang là /signup và /signin.

- Sửa:

- routes/auth.routes.js: thêm alias:

- POST /register → signUp

- POST /login → signIn

- Giải thích: Khớp endpoint với test.

7) Chuẩn hóa response format và status code trong auth.controllers

- Lỗi:

- Test kỳ vọng response có success và status code cụ thể; controller trả message khác hoặc thiếu success.

- Ví dụ: login user không tồn tại test mong 401, code trả 400; message khác; thiếu success.

- Sửa:

- signUp:

- Trả 201 và { success: true, message: 'User registered successfully', user: { ... } }

- Mặc định role = 'user' nếu không gửi.

- Các lỗi trả 400 với { success: false, message: ... }.

- signIn:

- User không tồn tại hoặc mật khẩu sai → 401 với { success: false, message: 'Invalid credentials' | 'Invalid password' }.

- Thành công trả 200 và { success: true, message: 'Login successful', user: { ... } }.

- Giải thích: Đồng bộ thông điệp và status code với kỳ vọng của test.

8) Thiết lập JWT\_SECRET cho môi trường test

- Lỗi:

- JsonWebTokenError: secret or public key must be provided trong test utils.

- Sửa:

- tests/setup.cjs: set process.env.JWT\_SECRET = process.env.JWT\_SECRET || 'test-secret-key'.

- generateToken/verifyToken trong utils/auth.js cũng fallback key test nếu thiếu env.

- Giải thích: Đảm bảo ký/verify token trong môi trường test không phụ thuộc .env thực tế.

Kết quả cuối

- Tất cả test pass: 3/3 suite, 15/15 tests.

- Thời gian chạy ~6-7s.

- Không cần chạy Docker vì dùng mongodb-memory-server cho test.

Các file chính đã thay đổi/tạo

- jest.config.js: dùng setup.cjs + babel-jest transform.

- tests/setup.cjs: thiết lập NODE\_ENV='test', JWT\_SECRET test, khởi tạo MongoMemoryServer.

- babel.config.cjs: preset-env cho Jest.

- utils/auth.js: thêm các hàm auth utils.

- tests/models/user.test.js: đồng bộ field với schema.

- tests/controllers/auth.test.js: đồng bộ field + dùng đúng endpoint; assert theo response mới.

- routes/auth.routes.js: thêm alias /register, /login.

- controllers/auth.controllers.js: chuẩn hóa response/status, default role.

- index.js: không listen/connect DB khi test; export app.

**🌟 Giải thích Chi tiết Hướng dẫn Triển khai CI/CD**

Hướng dẫn này phác thảo một quy trình **Tích hợp Liên tục (CI)** và **Triển khai Liên tục (CD)** mạnh mẽ sử dụng **GitHub Actions** và **Docker** cho một ứng dụng có cấu trúc **BackEnd** và **FrontEnd**.

**1. CI/CD Pipeline Overview (Tổng quan về Pipeline CI/CD)**

Đây là phần trình bày kiến trúc và các bước cốt lõi của quy trình tự động.

* **1.1 Pipeline Architecture (Kiến trúc Pipeline):** Sơ đồ này minh họa luồng công việc:
  + **Code Push** (Đẩy code lên Git) hoặc **Pull Request** (Yêu cầu hợp nhất code) kích hoạt **GitHub Actions (CI Pipeline)**.
  + CI Pipeline thực hiện **Docker Build** (Tạo ảnh container) và tạo **Test Results** (Báo cáo kết quả kiểm thử).
  + Sau đó, nó dẫn đến **Deployment** (Triển khai) tới các môi trường **Staging/Prod**.
  + **Mục đích:** Cung cấp cái nhìn tổng quan trực quan về cách mọi thứ hoạt động cùng nhau.
* **1.2 Workflow Stages (Các Giai đoạn của Workflow):** Liệt kê 6 bước chính (và 1 tùy chọn) mà CI Pipeline sẽ thực hiện:
  1. **Code Quality:** Đảm bảo code tuân thủ tiêu chuẩn (linting, định dạng, kiểm tra bảo mật).
  2. **Unit Testing:** Kiểm thử các đơn vị code riêng lẻ (BackEnd dùng **Jest**, FrontEnd dùng **Vitest**).
  3. **Integration Testing:** Kiểm thử sự tương tác giữa các thành phần (API, DB).
  4. **Build (Docker):** Đóng gói ứng dụng thành các ảnh Docker (containers) để đảm bảo môi trường nhất quán.
  5. **Deploy (Staging):** Tự động triển khai lên môi trường thử nghiệm.
  6. **E2E Testing (Optional):** Kiểm thử trải nghiệm người dùng cuối.
  + **Mục đích:** Xác định rõ ràng trình tự và mục tiêu của từng bước tự động hóa.

**2. GitHub Actions Setup (Thiết lập GitHub Actions)**

Đây là nơi xác định các tệp cấu hình (YAML) để chạy CI/CD trên GitHub.

* **2.1 Create Workflow Directory (Tạo thư mục Workflow):** Lệnh tạo thư mục .github/workflows là **bắt buộc** vì GitHub Actions tự động tìm kiếm các tệp workflow YAML trong thư mục này.
* **2.2 Main CI Workflow (ci.yml):** Tệp này định nghĩa pipeline CI chính, chạy trên các sự kiện push và pull\_request tới nhánh main và develop.
  + **env:** Định nghĩa các biến môi trường dùng chung (NODE\_VERSION, REGISTRY, IMAGE\_NAME).
  + **jobs:** Chia pipeline thành các tác vụ độc lập, chạy song song (trừ build-images có needs).
    - **code-quality:** Thực hiện cài đặt dependency, chạy **lint** và **security audit** (npm audit) cho cả BackEnd và FrontEnd. **Tại sao?** Để ngăn chặn code kém chất lượng và lỗi bảo mật xâm nhập vào nhánh chính.
    - **backend-tests:** Chạy **Unit** và **Integration Tests** cho BackEnd. Nó sử dụng dịch vụ **MongoDB Docker container** trong GitHub Actions (**services**) để mô phỏng môi trường DB, đảm bảo môi trường kiểm thử thực tế hơn. Sau đó **Upload Coverage** lên Codecov.
    - **frontend-tests:** Tương tự, chạy **Unit** và **Integration Tests** cho FrontEnd. Sau đó **Build** FrontEnd và **Upload Coverage**.
    - **build-images:** Tác vụ này **chỉ chạy khi** các tác vụ kiểm thử và chất lượng (needs) đã hoàn thành thành công. Nó sử dụng ma trận (strategy: matrix) để xây dựng và đẩy (push) ảnh Docker cho cả **backend** và **frontend** lên **GitHub Container Registry (ghcr.io)**.
  + **Mục đích:** Tự động hóa quá trình kiểm thử, kiểm tra chất lượng và xây dựng ảnh Docker sau mỗi lần thay đổi code.
* **2.3 Testing Workflow (test.yml):** Một workflow riêng biệt để chạy các bộ kiểm thử chuyên sâu hơn.
  + **on: schedule:** Cho phép chạy kiểm thử toàn diện **hàng ngày lúc 2 giờ sáng** (cron: '0 2 \* \* \*').
  + **workflow\_dispatch:** Cho phép kích hoạt workflow này **thủ công** qua giao diện GitHub.
  + **extended-tests:** Chạy tất cả các kiểm thử (test:coverage và integration) và **Tạo báo cáo kiểm thử** (test-report.md) sau đó **Upload** các báo cáo này thành **Artifacts** để dễ dàng truy cập.
  + **Mục đích:** Đảm bảo chất lượng code liên tục và có báo cáo định kỳ, không chỉ dựa vào các thay đổi code tức thời.

**3. Docker Configuration for CI/CD (Cấu hình Docker cho CI/CD)**

Cấu hình cho việc đóng gói ứng dụng.

* **3.1 Multi-stage Dockerfile Optimization (BackEnd/Dockerfile):** Dockerfile này sử dụng **Multi-stage build** để tối ưu hóa kích thước ảnh và tăng tốc độ build/kiểm thử.
  + **base -> deps:** Cài đặt chỉ các dependencies cần thiết cho **sản xuất** (production).
  + **dev:** Dành cho phát triển cục bộ (chạy npm run dev).
  + **test:** Dành cho môi trường kiểm thử (chạy npm run test:ci). Đây là target được sử dụng trong docker-compose.test.yml.
  + **production:** Ảnh cuối cùng, rất nhỏ gọn (chỉ chứa dependencies cần thiết) và sử dụng **người dùng không phải root** (nodejs) để tăng cường bảo mật. Nó cũng có **HEALTHCHECK**.
  + **Mục đích:** Tạo ra các ảnh Docker nhỏ, an toàn và có thể tái sử dụng, phân tách rõ ràng giữa môi trường dev, test và production.
* **3.2 Docker Compose for CI Testing (docker-compose.test.yml):** Cấu hình để chạy kiểm thử bằng Docker Compose **cục bộ** hoặc trong môi trường CI khác.
  + **mongodb-test:** Thiết lập một DB MongoDB tạm thời cho kiểm thử, sử dụng tmpfs (hệ thống tệp tạm thời trong RAM) để **tăng tốc độ I/O** cho DB trong kiểm thử.
  + **backend-test & frontend-test:** Định nghĩa các dịch vụ kiểm thử, sử dụng target: test từ Dockerfile, trỏ đến DB kiểm thử và mount thư mục coverage để lấy kết quả kiểm thử ra ngoài container.
  + **Mục đích:** Cung cấp môi trường kiểm thử cô lập, nhất quán, dễ dàng tái tạo.

**4. GitHub Repository Setup (Thiết lập Kho lưu trữ GitHub)**

Các cấu hình cần thiết trên giao diện GitHub.

* **4.1 Repository Secrets Configuration (Cấu hình Secrets):** Hướng dẫn thêm các biến môi trường nhạy cảm (như mật khẩu DB, khóa JWT, token triển khai, v.v.) vào **GitHub Secrets**.
  + **Mục đích:** Giữ bí mật nhạy cảm khỏi bị công khai trong code hoặc logs của workflow.
* **4.2 Branch Protection Rules (Quy tắc Bảo vệ Nhánh):** Thiết lập các điều kiện **bắt buộc** phải hoàn thành trước khi code có thể được hợp nhất vào nhánh main.
  + **Required status checks:** Bắt buộc tất cả các job CI (code-quality, backend-tests, frontend-tests, build-images) phải **Pass** (thành công).
  + **Mục đích:** Đảm bảo chỉ có code đã được kiểm thử, đạt chất lượng và đã xây dựng ảnh Docker thành công mới được phép vào nhánh chính, ngăn ngừa lỗi triển khai.

**5. Local Development Workflow (Quy trình Phát triển Cục bộ)**

Giúp các nhà phát triển duy trì chất lượng code trước khi đẩy lên Git.

* **5.1 Pre-commit Hooks Setup (husky & lint-staged):** Cấu hình Git hooks để tự động chạy kiểm tra chất lượng và định dạng code **trước khi commit**.
  + **Mục đích:** Ngăn ngừa code kém chất lượng (ví dụ: lỗi lint, sai định dạng) được commit ngay từ đầu, giảm tải cho CI Pipeline.
* **5.2 Development Scripts (package.json):** Định nghĩa các script tiện lợi (dev, test, lint, docker:test, v.v.) trong package.json gốc để dễ dàng chạy các tác vụ từ terminal.
  + **Mục đích:** Đơn giản hóa các lệnh phức tạp và đảm bảo tất cả nhà phát triển sử dụng cùng một cách để chạy các tác vụ.

**6. Quality Gates and Metrics (Cổng Chất lượng và Số liệu)**

Thiết lập các ngưỡng để đánh giá chất lượng.

* **6.1 Coverage Requirements (jest.config.js):** Định nghĩa **ngưỡng bảo hiểm code** (coverageThreshold) bắt buộc phải đạt được. Nếu tổng thể hoặc các thư mục cụ thể không đạt ngưỡng này (ví dụ: branches: 80), kiểm thử sẽ thất bại.
  + **Mục đích:** Đảm bảo mức độ kiểm thử đủ cao và ngăn chặn việc hợp nhất code chưa được kiểm thử.
* **6.2 Performance Budgets (FrontEnd/vite.config.js):** Cấu hình để tách các gói code lớn (vendor, router, redux) và đặt giới hạn cảnh báo kích thước gói (chunkSizeWarningLimit).
  + **Mục đích:** Quản lý và kiểm soát kích thước bundle của FrontEnd, ngăn ngừa hiệu suất tải trang chậm.

**8. Deployment Pipeline (Pipeline Triển khai)**

Ví dụ triển khai cụ thể cho môi trường Staging.

* **8.1 Staging Deployment (deploy-staging.yml):** Workflow chạy khi push lên nhánh develop hoặc kích hoạt thủ công.
  + Sử dụng môi trường **staging** của GitHub Actions để quản lý biến môi trường.
  + Sử dụng công cụ **Railway CLI** để triển khai các dịch vụ **backend** và **frontend**.
  + Thực hiện **Health Checks** (kiểm tra trạng thái) và **Smoke Tests** (kiểm thử cơ bản) sau khi triển khai để xác nhận ứng dụng đã hoạt động ổn định.
  + **Mục đích:** Tự động hóa việc triển khai lên môi trường staging và xác minh tính khả dụng cơ bản.

**✅ Tính Đúng Đắn của Việc Triển khai Theo Hướng dẫn**

**Có, việc triển khai theo hướng dẫn này là một cách tiếp cận rất đúng đắn và chuyên nghiệp để thiết lập CI/CD.**

**Lý do Hướng dẫn này là Tốt:**

1. **Tính Toàn diện:** Nó bao gồm toàn bộ chu trình DevOps từ kiểm tra chất lượng code (lint, audit), kiểm thử nhiều cấp độ (unit, integration), xây dựng ảnh (docker build), đến triển khai (deploy-staging) và đo lường chất lượng (coverage thresholds).
2. **Best Practices (Thực tiễn Tốt nhất):**
   * **Tích hợp sớm:** Sử dụng pre-commit hooks (Husky) để bắt lỗi ngay từ local.
   * **Môi trường nhất quán:** Sử dụng **Docker** và **Multi-stage build** giúp đảm bảo rằng những gì chạy trên CI/CD và Production là chính xác như nhau.
   * **An toàn:** Sử dụng GitHub Secrets, người dùng không phải root trong Docker, và kiểm tra bảo mật (npm audit).
   * **Khả năng song song:** Thiết lập các job GitHub Actions độc lập để chạy song song (ví dụ: code-quality, backend-tests, frontend-tests), giúp **tăng tốc** CI.
   * **Cổng chất lượng (Quality Gates):** Áp dụng Branch Protection Rules và Coverage Thresholds để **ngăn chặn** code lỗi hoặc chưa được kiểm thử hợp nhất vào nhánh chính.
3. **Chi tiết Kỹ thuật:** Các cấu hình như việc sử dụng **services** và **MongoDB Docker** trong backend-tests job, tmpfs trong docker-compose.test.yml, và metadata-action của Docker là những kỹ thuật tiên tiến, cho thấy độ hiểu biết sâu sắc về CI/CD và Docker.