UỶ BAN NHÂN DÂN THÀNH PHỐ HỒ CHÍNH MINH

TRƯỜNG ĐẠI HỌC SÀI GÒN



KIỂM THỬ PHẦN MỀM

KHOA : CÔNG NGHỆ THÔNG TIN

**ĐỀ TÀI : KIỂM THỬ WEBSITE FAST FOOD DELIVERY TRÊN MÔ HÌNH AGILE CI/CD**

|  |  |  |
| --- | --- | --- |
| **GIẢNG VIÊN** | **SINH VIÊN THỰC HIỆN** | |
| Ts. Đỗ Như Tài | Nguyễn Thanh Tịnh | : 3122411215 |
|  | Đỗ Nguyễn Nhật Trường | : 3122411230 |
|  | Nguyễn Thị Diệu Linh | : 3122411112 |
|  | Nguyễn Tấn Thành | : 3122411192 |

THÀNH PHỐ HỒ CHÍ MINH, 2025

**MỤC LỤC**

[I. TỔNG QUAN ĐỀ TÀI 6](#_Toc217036904)

[1. Mô tả dự án. 6](#_Toc217036905)

[2. Phương pháp nghiên cứu 6](#_Toc217036906)

[2.1. Các phương pháp kiểm thử 6](#_Toc217036907)

[2.1.1. Kiểm thử tĩnh 6](#_Toc217036908)

[2.1.2. Kiểm thử động 7](#_Toc217036909)

[2.2. Các chiến lược kiểm thử 7](#_Toc217036910)

[2.2.1. Kiểm thử hộp đen 7](#_Toc217036911)

[2.2.2. Kiểm thử hộp trắng 8](#_Toc217036912)

[2.2.3. Kiểm thử hộp xám 8](#_Toc217036913)

[2.3 Các cấp độ kiểm thử 9](#_Toc217036914)

[2.3.1 Kiểm thử đơn vị 9](#_Toc217036915)

[2.3.2. Kiểm thử tích hợp 10](#_Toc217036916)

[2.3.3. Kiểm thử hệ thống 11](#_Toc217036917)

[2.3.4. Kiểm thử chấp nhận 13](#_Toc217036918)

[3. Khảo sát hiện trạng 15](#_Toc217036919)

[3.1. Đối tượng nghiêm cứu 15](#_Toc217036920)

[3.2. Kế hoạch làm việc 15](#_Toc217036921)

[3.3. Tổng quan về phần mềm 15](#_Toc217036922)

[TechStack 16](#_Toc217036923)

[3.4. Business Context 17](#_Toc217036924)

[II. THIẾT KẾ HỆ THỐNG VÀ KIẾN TRÚC 18](#_Toc217036925)

[1. Thiết kế hệ thống 18](#_Toc217036926)

[1.1. Workflow 18](#_Toc217036927)

[1.2. Activity 21](#_Toc217036928)

[1.2.1 Access Control 22](#_Toc217036929)

[1.2.2 Food Management 25](#_Toc217036930)

[1.2.3 Place Order (cart management) 29](#_Toc217036931)

[1.2.4. Menu Catalog Browsing 33](#_Toc217036932)

[1.2.5 Order management 37](#_Toc217036933)

[a) Process Payment 37](#_Toc217036934)

[b) Manage Customer Orders 39](#_Toc217036935)

[1.3. User story 42](#_Toc217036936)

[1.3.1. User Authentication 42](#_Toc217036937)

[1.3.2. Ordering & Cart 42](#_Toc217036938)

[1.3.3. Payment & Checkout 42](#_Toc217036939)

[1.3.4. Delivery Tracking 43](#_Toc217036940)

[1.3.5. Admin Management 43](#_Toc217036941)

[1.4. Conceptual Model 44](#_Toc217036942)

[a) Food System 44](#_Toc217036943)

[b) Cart & OrderItem (khu màu vàng) 44](#_Toc217036944)

[c) User & Role 45](#_Toc217036945)

[d) Order 45](#_Toc217036946)

[e) Payment 45](#_Toc217036947)

[f) Delivery 45](#_Toc217036948)

[1.5. Data model 46](#_Toc217036949)

[1.5.1. ERD 46](#_Toc217036950)

[1.5.2. LOGICAL 47](#_Toc217036951)

[1.6. Data dictionary 48](#_Toc217036952)

[1.6.1. User 48](#_Toc217036953)

[1.6.2. Food 48](#_Toc217036954)

[1.6.3. Order 49](#_Toc217036955)

[2. Thiết kế kiến trúc 50](#_Toc217036956)

[2.1. Sơ đồ khối (Block diagram) 50](#_Toc217036957)

[2.2. Sơ đồ triển khai (Deployment View) 51](#_Toc217036958)

[2.3. Sơ đồ giao tiếp API (Communication view) 52](#_Toc217036959)

[3. Decomposition View 54](#_Toc217036960)

[3.1. Context diagram – C1 54](#_Toc217036961)

[3.2. Container diagram – C2 55](#_Toc217036962)

[3.3. Component Diagram (High-level) 56](#_Toc217036963)

[3.4. Code Diagram-C4 58](#_Toc217036964)

[III. KẾ HOẠCH KIỂM THỬ 65](#_Toc217036965)

[1. Giới thiệu 65](#_Toc217036966)

[1.1 Mục đích 65](#_Toc217036967)

[1.2 Bối cảnh (Background) 65](#_Toc217036968)

[1.3 Phạm vi (Scope) 65](#_Toc217036969)

[1.4 Nhận diện dự án (Project Identification) 66](#_Toc217036970)

[1.5 Tài liệu tham khảo (References) 67](#_Toc217036971)

[2. Chức năng kiểm thử 68](#_Toc217036972)

[2.1 Functionality Testing 68](#_Toc217036973)

[2.1.1 Form Functionality 68](#_Toc217036974)

[2.1.2 Link Testing 68](#_Toc217036975)

[2.1.3 Business workflow 68](#_Toc217036976)

[2.2 Usability 68](#_Toc217036977)

[2.2.1 User Authentication (Xác thực người dùng) 68](#_Toc217036978)

[2.2.1.1. Account Registration – Sign Up (Đăng ký tài khoản) 68](#_Toc217036979)

[2.2.1.2. Error Handling for Existing Email (Xử lý lỗi email đã tồn tại) 69](#_Toc217036980)

[2.2.1.3. User Login (Đăng nhập) 69](#_Toc217036981)

[2.2.1.4. Invalid Credentials Handling (Xử lý thông tin đăng nhập sai) 69](#_Toc217036982)

[2.2.2. Menu Exploration & Selection (Khám phá và lựa chọn thực đơn) 69](#_Toc217036983)

[2.2.2.1. Category Filtering (Lọc theo danh mục) 69](#_Toc217036984)

[2.2.2.2. Category Deselection (Bỏ chọn danh mục) 70](#_Toc217036985)

[2.2.2.3. Add to Cart from Menu (Thêm món vào giỏ hàng) 70](#_Toc217036986)

[2.2.2.4. Quantity Adjustment (Điều chỉnh số lượng món) 70](#_Toc217036987)

[2.2.3. Cart Management (Quản lý giỏ hàng) 70](#_Toc217036988)

[2.2.3.1. View Cart (Xem giỏ hàng) 70](#_Toc217036989)

[2.2.3.2. Remove Item from Cart (Xóa món khỏi giỏ hàng) 70](#_Toc217036990)

[2.2.3.3. Proceed to Checkout (Chuyển sang thanh toán) 70](#_Toc217036991)

[2.2.4. Checkout And Order Placement (Thanh toán và đặt hàng) 71](#_Toc217036992)

[2.2.4.1. Delivery Information Validation (Kiểm tra thông tin giao hàng) 71](#_Toc217036993)

[2.2.4.2. Order Placement (Đặt hàng thành công) 71](#_Toc217036994)

[2.2.4.3. Order History Visibility (Xem lịch sử đơn hàng) 71](#_Toc217036995)

[2.3 Interface Testing (Kiểm thử giao diện & Kết nối) 71](#_Toc217036996)

[2.3.1 Error Handling (Xử lý lỗi tương tác): 71](#_Toc217036997)

[2.3.2 Transaction Interruption (Gián đoạn giao dịch): 72](#_Toc217036998)

[2.3.3 Connection Resilience (Độ ổn định kết nối): 72](#_Toc217036999)

[2.4 Compatibility Testing (Kiểm thử tương thích) 72](#_Toc217037000)

[2.4.1 Browser Compatibility (Tương thích trình duyệt) 72](#_Toc217037001)

[2.4.2 OS Compatibility (Tương thích hệ điều hành) 72](#_Toc217037002)

[2.5 Database Testing (Kiểm thử CSDL) 72](#_Toc217037003)

[2.5.1 Data Integrity (Toàn vẹn dữ liệu) 72](#_Toc217037004)

[2.5.2 Data Consistency (Nhất quán dữ liệu) 72](#_Toc217037005)

[2.6 Performance Testing (Kiểm thử hiệu năng) 72](#_Toc217037006)

[2.6.1 Load Testing (Kiểm thử tải) 72](#_Toc217037007)

[2.6.2 Stress Testing (Kiểm thử chịu đựng) 73](#_Toc217037008)

[3. Chức năng chưa kiểm thử 73](#_Toc217037009)

[3.1. Kiểm thử khả năng mở rộng (Scalability Testing) 73](#_Toc217037010)

[3.2. Kiểm thử bảo mật (Security Testing) 73](#_Toc217037011)

[4. Chiến lược kiểm thử (Test Strategy) 73](#_Toc217037012)

[4.1 Các loại kiểm thử (Testing Types) 73](#_Toc217037013)

[4.1.1 Kiểm thử Chức năng và Khả năng sử dụng (Function and Usability Testing) 73](#_Toc217037014)

[4.1.2 Kiểm thử Giao diện (Interface Testing) 74](#_Toc217037015)

[4.1.3 Kiểm thử Tính tương thích (Compatibility Testing) 75](#_Toc217037016)

[4.1.4 Kiểm thử Cơ sở dữ liệu (Database Testing) 75](#_Toc217037017)

[4.1.5 Kiểm thử Hiệu năng (Performance Testing) 75](#_Toc217037018)

[4.2 Tools 76](#_Toc217037019)

[5. Kiểm thử đơn vị (Unit Testing) 76](#_Toc217037020)

[6. Kiểm thử tích hợp (Integration Testing) 77](#_Toc217037021)

[IV. PHƯƠNG PHÁP KIỂM THỬ 79](#_Toc217037022)

[1. Tổng quan 79](#_Toc217037023)

[2.1. Cấu trúc tổng thể của V-Model 80](#_Toc217037024)

[2.2. Phân tích từng cặp giai đoạn đối xứng 80](#_Toc217037025)

[2.2.1 Business Requirement Specification ↔ Acceptance Testing 80](#_Toc217037026)

[2.2.2 System Requirement Specification ↔ System Testing 80](#_Toc217037027)

[2.2.3. High Level Design ↔ Integration Testing 81](#_Toc217037028)

[2.2.4. Low Level Design ↔ Unit Testing 81](#_Toc217037029)

[2.3. Ý nghĩa của Coding trong V-Model 81](#_Toc217037030)

[3. Phân tích khung nhìn Agile CI/CD 82](#_Toc217037031)

[3.1 Tổng quan về Agile CI/CD 82](#_Toc217037032)

[3.2. Continuous Integration (CI) 82](#_Toc217037033)

[3.3. Continuous Delivery/Deployment (CD) 83](#_Toc217037034)

[3.4. Agile trong CI/CD 83](#_Toc217037035)

[3.5. Triền Khai CI/CD 84](#_Toc217037036)

[4. Static and Dynamic Testing 90](#_Toc217037037)

[4.1. Static Testing 90](#_Toc217037038)

[4.1.1. Khái niệm 90](#_Toc217037039)

[4.1.2. Vai trò của Static Testing trong đồ án 90](#_Toc217037040)

[4.1.3. Các hình thức Static Testing đã áp dụng 90](#_Toc217037041)

[4.1.4. Checklist Review 91](#_Toc217037042)

[4.1.4. Review Test Case (Excel – Manual Test) 91](#_Toc217037043)

[4.1.5. Kết luận 92](#_Toc217037044)

[4.2. Dynamic Testing 92](#_Toc217037045)

[4.2.1. Khái niệm 92](#_Toc217037046)

[4.2.2. White - box Testing 92](#_Toc217037047)

[4.2.3. Black - box Testing 93](#_Toc217037048)

[4.2.3. Gray - box Testing 94](#_Toc217037049)

[5. Kiểm thử tự động (Auto Testing) 95](#_Toc217037050)

[5.1. Khái niệm 95](#_Toc217037051)

[5.2. Các chức năng kiểm thử tự động 95](#_Toc217037052)

[5.2.1. Database Testing 95](#_Toc217037053)

[5.2.1. Performance Testing 97](#_Toc217037054)

[V. KẾT QUẢ KIỂM THỬ 100](#_Toc217037055)

[1. Test Defect 100](#_Toc217037056)

[2. Test Summary 101](#_Toc217037057)

[VI. Kết luận & hướng phát triển 103](#_Toc217037058)

[6.1. Kết luận 103](#_Toc217037059)

[6.2. Hướng phát triển 103](#_Toc217037060)

**DANH MỤC HÌNH ẢNH**

[Figure 1. Giai Đoạn Kiểm Thử 9](#_Toc217036724)

[Figure 2. Workflow của Web Food Delivery 20](#_Toc217036725)

[Figure 3. Usecase tổng của Web Food Delivery 21](#_Toc217036726)

[Figure 4. Conceptual Model 44](#_Toc217036727)

[Figure 5. ERD mức quan niệm 46](#_Toc217036728)

[Figure 6. ERD mức Logic 47](#_Toc217036729)

[Figure 7. Deployment View 51](#_Toc217036730)

[Figure 8. Communication View. 52](#_Toc217036731)

[Figure 9. C1 - Context Diagram 54](#_Toc217036732)

[Figure 10. C2 - Container Diagram 55](#_Toc217036733)

[Figure 11. C3 - Component Diagram 56](#_Toc217036734)

[Figure 12. C4 - Code Diagram 58](#_Toc217036735)

[Figure 13. Khung nhìn V-models 79](#_Toc217036736)

[Figure 14. Agile CI/CD 82](#_Toc217036737)

[Figure 15. Triển khai CI/CD 84](#_Toc217036738)

[Figure 16. Ví dụ về Usability Testing - Test Design 91](#_Toc217036739)

[Figure 17. Ví dụ về Usability Testing - Test Cases – Test Data 92](#_Toc217036740)

[Figure 18. Ví dụ về White - box Testing trong Unit Test 93](#_Toc217036741)

[Figure 19. Ví dụ về Gray - box Testing trong Database Test Cases 94](#_Toc217036742)

[Figure 20. Test Defect 100](#_Toc217036743)

[Figure 21. Biểu đồ tổng hợp các Test 102](#_Toc217036744)

# I. TỔNG QUAN ĐỀ TÀI

## 1. Mô tả dự án.

Trong bối cảnh lĩnh vực F&B luôn luôn đầy rẫy sự cạnh tranh nhiều tập đoàn lớn hay công ty, chuỗi cửa hàng nhà nhỏ ngày càng quan tâm về việc dịch vụ mà họ cung cấp có đủ thoã mãn, đáp ứng được nhu cầu của khách hàng hay không. đứng trước thời đại công nghệ phát triển nhiều doanh nghiệp đã chuyển mình thích ứng với việc áp dụng công nghệ vào lĩnh vực F&B của họ. với sự cạnh tranh ấy thì không ít các website fast food delivery ra đời. thì người dùng họ sẽ yêu cầu và quan tâm đến việc trải nghiệm mượt mà, nhanh chống và không có lỗi đặc biệt là cấc quy trình quan trọng như tìm kiếm giỏ hàng và thanh toán, một lỗi nhỏ thôi cũng có thể khiến doanh nghiệp mất đi nguồn khách hàng không hề nhỏ ngay lập tức.

Để đảm bảo được điều đó các đội ngũ phát triển liên tục cập nhật các tính năng để giữ tính cạnh tranh. điều này tạo ra một áp lực lớn đó là nếu kiểm thử thủ công : sẽ tạo nên một “nút thắc cổ trai”. Việc kiểm thử website sau mỗi lần cập nhật nhỏ là không khả thi, tốn thời gian và dễ sai sót. Nếu triển khai thủ công qua trì build và deploy code mới lên máy chủ phức tạp, rủi ro cao và thường chỉ được thực hiện định kỳ, làm chậm tốc độ ra mắt tính năng.

Chính vì vậy dự án “Kiểm thử website fast food delivery trên mô hình Agile CI/CD” được thành lập.

## 2. Phương pháp nghiên cứu

### 2.1. Các phương pháp kiểm thử

Phương pháp kiểm thử chính ở đây là kiểm thử tĩnh và kiểm thử động

#### 2.1.1. Kiểm thử tĩnh

Là phương pháp thử phần mềm đòi hỏi phải duyệt lại các yêu cầu và các đặc tả bằng tay, thông qua việc sử dụng giấy, bút để kiểm tra logic, lần từng chi tiết mà không cần chạy chương trình. Kiểu kiểm thử này thường được sử dụng bởi chuyên viên thiết kế người mà viết mã lệnh một mình.

Kiểm thử tĩnh cũng có thể được tự động hóa. Nó sẽ thực hiện kiểm tra toàn bộ baogồm các chương trình được phân tích bởi một trình thông dịch hoặc biên dịch mà xácnhận tính hợp lệ về cú pháp của chương trình

#### 2.1.2. Kiểm thử động

Là phương pháp thử phần mềm thông qua việc dùng máy chạy chương trình đểđiều tra trạng thái tác động của chương trình. Đó là kiểm thử dựa trên các ca kiểm thửxác định bằng sự thực hiện của đối tượng kiểm thử hay chạy các chương trình. Kiểm thửđộng kiểm tra cách thức hoạt động động của mã lệnh, tức là kiểm tra sự phản ứng vật lýtừ hệ thống tới các biến luôn thay đổi theo thời gian. Trong kiểm thử động, phần mềmphải thực sự được biên dịch và chạy. Kiểm thử động thực sự bao gồm làm việc với phầnmềm, nhập các giá trị đầu vào và kiểm tra xem liệu đầu ra có như mong muốn hay không.Các phương pháp kiểm thử động gồm có kiểm thử Unit – Unit Tests, Kiểm thử tích hợp –Intergration Tests, Kiểm thử hệ thống – System Tests, và Kiểm thử chấp nhận sản phẩm –Acceptance Tests.

### 2.2. Các chiến lược kiểm thử

Ba trong những chiến lược kiểm thử thông dụng nhất gồm: kiểm thử hộp đen,kiểm thử hộp trắng, kiểm thử hộp xám.

#### 2.2.1. Kiểm thử hộp đen

Kiểm thử hộp đen là một phương pháp kiểm thử phần mềm tập trung vào việc kiểm tra các chức năng bên ngoài của ứng dụng mà không cần biết mã nguồn hoặc cấu trúc bên trong của nó. Người kiểm thử tương tác với hệ thống như một người dùng cuối, cung cấp dữ liệu đầu vào và quan sát đầu ra để xác minh xem phần mềm có hoạt động đúng theo yêu cầu và mong đợi hay không.

* **Ưu nhược điểm**

Kiểm thử hộp đen không có mối liên quan nào tới mã lệnh, và kiểm thử viên chỉ rất đơn giản tâm niệm là: một mã lệnh phải có lỗi. Sử dụng nguyên tắc “hãy đòi hỏi và bạn sẽ được nhận”, những kiểm thử viên hộp đen tìm ra lỗi mà những lập trình viên đã không tìm ra. nhưng, mặt khác, người ta cũng nói kiểm thử hộp đen “giống như là đitrong bóng tối mà không có đèn vậy”, bởi vì kiểm thử viên không biết các phần mềm được kiểm tra thực sự được xây dựng như thế nào. đó là lý do mà có nhiều trường hợp mà một kiểm thử viên hộp đen viết rất nhiều ca kiểm thử để kiểm tra một thứ gì đó mà đáng lẽ có thể chỉ cần kiểm tra bằng 1 ca kiểm thử duy nhất, và/hoặc một số phần của chương trình không được kiểm tra chút nào.

Do vậy, kiểm thử hộp đen có ưu điểm của “một sự đánh giá khách quan”, mặt khác nó lạicó nhược điểm của “thăm dò mù”

#### 2.2.2. Kiểm thử hộp trắng

Là một chiến lược kiểm thử khác, trái ngược hoàn toàn với kiểm thử hộp đen, kiểm thử hộp trắng hay kiểm thử hướng logic cho phép bạn khảo sát cấu trúc bên trong của chương trình. Chiến lược này xuất phát từ dữ liệu kiểm thử bằng sự kiểm thử tính logic của chương trình. Kiểm thử viên sẽ truy cập vào cấu trúc dữ liệu và giải thuật bên trong chương trình (và cả mã lệnh thực hiện chúng)

Các phương pháp kiểm thử hộp trắng

* Kiểm thử giao diện lập trình ứng dụng - API testing (application programming interface): là phương pháp kiểm thử của ứng dụng sử dụng các API công khai và riêng tư
* Bao phủ mã lệnh – Code coverage: tạo các kiểm tra để đáp ứng một số tiêu chuẩnvề bao phủ mã lệnh
* Các phương pháp gán lỗi – Fault injection.
* Các phương pháp kiểm thử hoán chuyển – Mutation testing methods
* Kiểm thử tĩnh – Static testing: kiểm thử hộp trắng bao gồm mọi kiểm thử tĩnh.

Phương pháp kiểm thử hộp trắng cũng có thể được sử dụng để đánh giá sự hoànthành của một bộ kiểm thử mà được tạo cùng với các phương pháp kiểm thử hộp đen.Điều này cho phép các nhóm phần mềm khảo sát các phần của 1 hệ thống ít khi đượckiểm tra và đảm bảo rằng những điểm chức năng quan trọng nhất đã được kiểm tra

#### 2.2.3. Kiểm thử hộp xám

Kiểm thử hộp xám đòi hỏi phải có sự truy cập tới cấu trúc dữ liệu và giải thuật bên trong cho những mục đích thiết kế các ca kiểm thử, nhưng là kiểm thử ở mức người sử dụng hay mức hộp đen. Việc thao tác tới dữ liệu đầu vào và định dạng dữ liệu đầu ra là không rõ ràng, giống như một chiếc “hộp xám”, bởi vì đầu vào và đầu ra rõ ràng là ở bên ngoài “hộp đen” mà chúng ta vẫn gọi về hệ thống được kiểm tra. Sự khác biệt này đặc biệt quan trọng khi quản lý kiểm thử tích hợp - Intergartion testing giữa 2 module mã lệnh được viết bởi hai chuyên viên thiết kế khác nhau, trong đó chỉ giao diện là được đưa ra để kiểm thử. Kiểm thử hộp xám có thể cũng bao gồm cả thiết kế đối chiếu để quyết định, ví dụ, giá trị biên hay thông báo lỗi.

### 2.3 Các cấp độ kiểm thử

Kiểm thử phần mềm gồm có các cấp độ: Kiểm thử đơn vị, Kiểm thử tích hợp, Kiểm thử hệ thống và Kiểm thử chấp nhận sản phẩm.

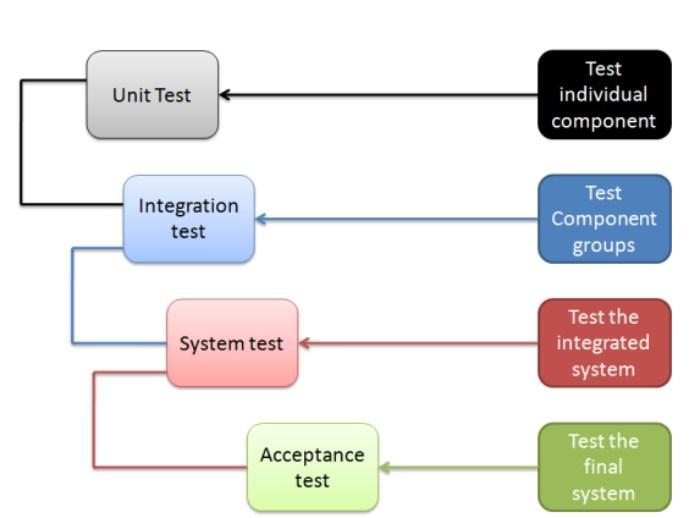


Figure 1. Giai Đoạn Kiểm Thử

#### 2.3.1 Kiểm thử đơn vị

Một đơn vị là một thành phần phần mềm nhỏ nhất mà ta có thể kiểm thử được. Vídụ, các hàm (Function), thủ tục (Procedure), lớp (Class) hay phương thức (Method) đềucó thể được xem là Unit.

Vì Unit được chọn để kiểm tra thường có kích thước nhỏ và chức năng hoạt độngđơn giản, chúng ta không khó khăn gì trong việc tổ chức kiểm thử, ghi nhận và phân tíchkết quả kiểm thử. Nếu phát hiện lỗi, việc xác định nguyên nhân và khắc phục cũng tươngđối dễ dàng vì chỉ khoanh vùng trong một đơn thể Unit đang kiểm tra. Một nguyên lý đúc kết từ thực tiễn: thời gian tốn cho Unit Test sẽ được đền bù bằng việc tiết kiệm rất nhiều thời gian và chi phí cho việc kiểm thử và sửa lỗi ở các mức kiểm thử sau đó.

Unit Test thường do lập trình viên thực hiện. Công đoạn này cần được thực hiệncàng sớm càng tốt trong giai đoạn viết code và xuyên suốt chu kỳ phát triển phần mềm.Thông thường, Unit Test đòi hỏi kiểm thử viên có kiến thức về thiết kế và code củachương trình. Mục đích của Unit Test là bảo đảm thông tin được xử lý và xuất (khỏi Unit)là chính xác, trong mối tương quan với dữ liệu nhập và chức năng của Unit.

Điều này thường đòi hỏi tất cả các nhánh bên trong Unit đều phải được kiểm tra đểphát hiện nhánh phát sinh lỗi. Một nhánh thường là một chuỗi các lệnh được thực thitrong một Unit. Ví dụ: chuỗi các lệnh sau điều kiện If và nằm giữa then ... else là mộtnhánh. Thực tế việc chọn lựa các nhánh để đơn giản hóa việc kiểm thử và quét hết Unitđòi hỏi phải có kỹ thuật, đôi khi phải dùng thuật toán để chọn lựa.

Cùng với các mục kiểm thử khác, Unit Test cũng đòi hỏi phải chuẩn bị trước cácca kiểm thử (Test case) hoặc kịch bản kiểm thử (Test script), trong đó chỉ định rõ dữ liệuđầu vào, các bước thực hiện và dữ liệu đầu ra mong muốn. Các Test case và Test scriptnày nên được giữ lại để tái sử dụng.

#### 2.3.2. Kiểm thử tích hợp

Integration test kết hợp các thành phần của một ứng dụng và kiểm thử như mộtứng dụng đã hoàn thành. Trong khi Unit Test kiểm tra các thành phần và Unit riêng lẻ thìIntgration Test kết hợp chúng lại với nhau và kiểm tra sự giao tiếp giữa chúng.

Hai mục tiêu chính của Integration Test:

* Phát hiện lỗi giao tiếp xảy ra giữa các Unit.
* Tích hợp các Unit đơn lẻ thành các hệ thống nhỏ (Subsystem) và cuối cùng lànguyên hệ thống hoàn chỉnh (System) chuẩn bị cho kiểm thử ở mức hệ thống (SystemTest)

Trong Unit Test, lập trình viên cố gắng phát hiện lỗi liên quan đến chức năng vàcấu trúc nội tại của Unit. Có một số phép kiểm thử đơn giản trên giao tiếp giữa Unit vớicác thành phần liên quan khác, tuy nhiên mọi giao tiếp liên quan đến Unit chỉ thật sựđược kiểm tra đầy đủ khi các Unit tích hợp với nhau trong khi thực hiện Integration Test.

Trừ một số ít ngoại lệ, Integration Test chỉ nên thực hiện trên những Unit đã đượckiểm tra cẩn thận trước đó bằng Unit Test, và tất cả các lỗi mức Unit đã được sửa chữa.Một số người hiểu sai rằng Unit một khi đã qua giai đoạn Unit Test với các giao tiếp giảlập thì không cần phải thực hiện Integration Test nữa. Thực tế việc tích hợp giữa các Unitdẫn đến những tình huống hoàn toàn khác.

Một chiến lược cần quan tâm trong Integration Test là nên tích hợp dần từng Unit.Một Unit tại một thời điểm được tích hợp vào một nhóm các Unit khác đã tích hợp trướcđó và đã hoàn tất các đợt Integration Test trước đó. Lúc này, ta chỉ cần kiểm thử giao tiếpcủa Unit mới thêm vào với hệ thống các Unit đã tích hợp trước đó, điều này sẽ làm chosố lượng can kiểm thử giảm đi rất nhiều, và sai sót sẽ giảm đáng kể.

Có 4 loại kiểm thử trong Integration Test:

Kiểm thử cấu trúc (Structure Test): Tương tự White Box Test, kiểm thử cấu trúc nhằm bảo đảm các thành phần bên trong của một chương trình chạy đúng và chú trọng đến hoạt động của các thành phần cấu trúc nội tại của chương trình chẳng hạn các câu lệnh và nhánh bên trong.

Kiểm thử chức năng (Functional Test): Tương tự Black Box Test, kiểm thử chức năng chỉ chú trọng đến chức năng của chương trình, mà không quan tâm đến cấu trúc bên trong, chỉ khảo sát chức năng của chương trình theo yêu cầu kỹ thuật.

Kiểm thử hiệu năng (Performance Test): Kiểm thử việc vận hành của hệ thống.

Kiểm thử khả năng chịu tải (Stress Test): Kiểm thử các giới hạn của hệ thống.

#### 2.3.3. Kiểm thử hệ thống

Mục đích System Test là kiểm thử thiết kế và toàn bộ hệ thống (sau khi tích hợp) có thỏa mãn yêu cầu đặt ra hay không.

System Test bắt đầu khi tất cả các bộ phận của phần mềm đã được tích hợp thành công. Thông thường loại kiểm thử này tốn rất nhiều công sức và thời gian. Trong nhiều trường hợp, việc kiểm thử đòi hỏi một số thiết bị phụ trợ, phần mềm hoặc phần cứng đặc thù, đặc biệt là các ứng dụng thời gian thực, hệ thống phân bố, hoặc hệ thống nhúng. Ở mức độ hệ thống, người kiểm thử cũng tìm kiếm các lỗi, nhưng trọng tâm là đánh giá về hoạt động, thao tác, sự tin cậy và các yêu cầu khác liên quan đến chất lượng của toàn hệ thống

Điểm khác nhau then chốt giữa Integration Test và System Test là System Test chú trọng các hành vi và lỗi trên toàn hệ thống, còn Integration Test chú trọng sự giao tiếp giữa các đơn thể hoặc đối tượng khi chúng làm việc cùng nhau. Thông thường ta phải thực hiện Unit Test và Integration Test để bảo đảm mọi Unit và sự tương tác giữa chúng hoạt động chính xác trước khi thực hiện System Test.

Sau khi hoàn thành Integration Test, một hệ thống phần mềm đã được hình thành cùng với các thành phần đã được kiểm tra đầy đủ. Tại thời điểm này, lập trình viên hoặc kiểm thử viên bắt đầu kiểm thử phần mềm như một hệ thống hoàn chỉnh. Việc lập kế hoạch cho System Test nên bắt đầu từ giai đoạn hình thành và phân tích các yêu cầu.

System Test kiểm thử cả các hành vi chức năng của phần mềm lẫn các yêu cầu về chất lượng như độ tin cậy, tính tiện lợi khi sử dụng, hiệu năng và bảo mật. Mức kiểm thửnày đặc biệt thích hợp cho việc phát hiện lỗi giao tiếp với phần mềm hoặc phần cứng bên ngoài, chẳng hạn các lỗi "tắc nghẽn" (deadlock) hoặc chiếm dụng bộ nhớ. Sau giai đoạn System Test, phần mềm thường đã sẵn sàng cho khách hàng hoặc người dùng cuối cùng kiểm thử chấp nhận sản phẩm (Acceptance Test) hoặc dùng thử (Alpha/Beta Test).

Đòi hỏi nhiều công sức, thời gian và tính chính xác, khách quan, System Test thường được thực hiện bởi một nhóm kiểm thử viên hoàn toàn độc lập với nhóm phát triển dự án. Bản thân System Test lại gồm nhiều loại kiểm thử khác nhau, phổ biến nhất gồm:

* Kiểm thử chức năng (Functional Test): Bảo đảm các hành vi của hệ thống thỏa mãn đúng yêu cầu thiết kế.
* Kiểm thử hiệu năng (Performance Test): Bảo đảm tối ưu việc phân bổ tài nguyên hệ thống (ví dụ bộ nhớ) nhằm đạt các chỉ tiêu như thời gian xử lý hay đáp ứng câu truy vấn
* Kiểm thử khả năng chịu tải (Stress Test hay Load Test): Bảo đảm hệ thống vận hành đúng dưới áp lực cao (ví dụ nhiều người truy xuất cùng lúc). Stress Test tập trungvào các trạng thái tới hạn, các "điểm chết", các tình huống bất thường như đang giao dịch thì ngắt kết nối (xuất hiện nhiều trong kiểm tra thiết bị như POS, ATM...)
* Kiểm thử cấu hình (Configuration Test).
* Kiểm thử bảo mật (Security Test): Bảo đảm tính toàn vẹn, bảo mật của dữ liệu vàcủa hệ thống
* Kiểm thử khả năng phục hồi (Recovery Test): Bảo đảm hệ thống có khả năng khôi phục trạng thái ổn định trước đó trong tình huống mất tài nguyên hoặc dữ liệu; đặc biệt quan trọng đối với các hệ thống giao dịch như ngân hàng trực tuyến.

Nhìn từ quan điểm người dùng, các cấp độ kiểm thử trên rất quan trọng: Chúng bảo đảm hệ thống đủ khả năng làm việc trong môi trường thực.

Lưu ý là không nhất thiết phải thực hiện tất cả các loại kiểm thử nêu trên. Tùy yêucầu và đặc trưng của từng hệ thống, tuỳ khả năng và thời gian cho phép của dự án, khi lập kế hoạch, người Quản lý dự án sẽ quyết định áp dụng những loại kiểm thử nào.

#### 2.3.4. Kiểm thử chấp nhận

Thông thường, sau giai đoạn System Test là Acceptance Test, được khách hàng thực hiện (hoặc ủy quyền cho một nhóm thứ ba thực hiện). Mục đích của Acceptance Test là để chứng minh phần mềm thỏa mãn tất cả yêu cầu của khách hàng và khách hàng chấp nhận sản phẩm (và trả tiền thanh toán hợp đồng).

Acceptance Test có ý nghĩa hết sức quan trọng, mặc dù trong hầu hết mọi trường hợp, các phép kiểm thử của System Test và Acceptance Test gần như tương tự, nhưng bản chất và cách thức thực hiện lại rất khác biệt.

Đối với những sản phẩm dành bán rộng rãi trên thị trường cho nhiều người sửdụng, thông thường sẽ thông qua hai loại kiểm thử gọi là kiểm thử Alpha – Alpha Test và kiểm thử Beta - Beta Test. Với Alpha Test, người dùng kiểm thử phần mềm ngay tại nơi phát triển phần mềm, lập trình viên sẽ ghi nhận các lỗi hoặc phản hồi, và lên kế hoạch sửa chữa. Với Beta Test, phần mềm sẽ được gửi tới cho người dùng để kiểm thử ngay trong môi trường thực, lỗi hoặc phản hồi cũng sẽ gửi ngược lại cho lập trình viên để sửa chữa.

Thực tế cho thấy, nếu khách hàng không quan tâm và không tham gia vào quá trình phát triển phần mềm thì kết quả Acceptance Test sẽ sai lệch rất lớn, mặc dù phần mềm đã trải qua tất cả các kiểm thử trước đó. Sự sai lệch này liên quan đến việc hiểu sai yêu cầu cũng như sự mong chờ của khách hàng. Ví dụ đôi khi một phần mềm xuất sắc vượt qua các phép kiểm thử về chức năng thực hiện bởi nhóm thực hiện dự án, nhưng khách hàng khi kiểm thử sau cùng vẫn thất vọng vì bố cục màn hình nghèo nàn, thao tác không tự nhiên, không theo tập quán sử dụng của khách hàng,…

Gắn liền với giai đoạn Acceptance Test thường là một nhóm những dịch vụ và tài liệu đi kèm, phổ biến như hướng dẫn cài đặt, sử dụng... Tất cả tài liệu đi kèm phải được cập nhật và kiểm thử chặt chẽ.

## 3. Khảo sát hiện trạng

### 3.1. Đối tượng nghiêm cứu

Đối tượng nghiên cứu cốt lõi của đề tài là quy trình kiểm thử phần mềm và các kỹ thuật kiểm thử tự động (automation testing techniques) được áp dụng trong bối cảnh cụ thể của một dự án phát triển theo mô hình Agile và triển khai bằng đường ống (pipeline) CI/CD. Đề tài không tập trung vào việc xây dựng phần mềm, mà tập trung vào việc xây dựng hệ thống đảm bảo chất lượng cho phần mềm đó.

### 3.2. Kế hoạch làm việc

|  |  |  |
| --- | --- | --- |
| **Giai đoạn** | **Nội dung công việc** | **Thời gian (Tuần)** |
| 1 | Khảo sát, phân tích yêu cầu, xác định mục tiêu, phạm vi. | 1-3 |
| 2 | Nghiên cứu lý thuyết | 4-5 |
| 3 | Xây dựng chiến lược kiểm thử và thiết kế Test Cases. | 6 |
| 4 | Lựa chọn công cụ và thiết lập môi trường (CI/CD pipeline). | 7 |
| 5 | Viết kiểm thử | 8-11 |
| 6 | Tích hợp Test vào pipeline CI/CD, chạy thực nghiệm. | 11-12 |
| 7 | Phân tích kết quả, đánh giá và hoàn thiện báo cáo. | 13-15 |

### 3.3. Tổng quan về phần mềm

Food Delivery Website is a modern full-stack web application built with React.js, MongoDB, Express, Node.js, and Stripe. It provides a seamless platform where customers can browse menus, place orders, and make secure online payments. The application is designed with a responsive and user-friendly interface powered by React.js, while MongoDB, Express, and Node.js ensure a robust and scalable backend. For payments, Stripe is integrated to deliver a smooth and secure checkout experience. This project demonstrates how to combine popular technologies in the MERN stack with third-party services to build a real-world, cloud-ready web application for the food delivery domain.

#### **TechStack**

Frontend

* React.js – Build dynamic, responsive, and interactive user interfaces
* React Router – Client-side routing for smooth navigation
* Axios / Fetch API – Handle HTTP requests to the backend

Backend

* Node.js – JavaScript runtime environment for scalable server-side applications
* Express.js – Lightweight and flexible framework for building RESTful APIs

Database

* MongoDB – NoSQL database for flexible and efficient data storage
* Mongoose – ODM for modeling and interacting with MongoDB

Payment Integration

* Stripe – Secure and reliable online payment gateway

Authentication & Security

* JWT (JSON Web Tokens) – Secure user authentication and session management
* bcrypt – Password hashing for data protection
* CORS & Helmet – Middleware for API securi ty and safe cross-origin requests

Other Tools & Libraries

* Nodemon – Auto-restart server during development
* Dotenv – Manage environment variables
* Git & GitHub – Version control and collaboration

### 3.4. Business Context

Fast Food Delivery Website has the basic business scenario for Menu Catalog, Shopping Cart, Payment Process, Order Management, Delivery Tracking, and Access Control.

With **the menu categories section**, customers can browse items by filtering and sorting by category. On the menu page, she can see the name, description, image, price, and availability of each dish. The Admin can manage the food catalog by adding, updating, or removing items, and uploading images.

With **Shopping Cart**, the Customer can add food items either from the menu list. After adding, the items appear in her shopping cart along with a summary panel showing item cost, delivery fee, discount, subtotal, and total amount. Whenever she updates the cart (add or remove items), the summary panel automatically reflects the changes.

With **Payment Process**, after the Customer clicks checkout, the system validates order data, integrates with **Stripe** to process payment, and then sends confirmation via email. Payment verification ensures secure transactions and updates the order status.

With **Order Management**, the Customer’s confirmed order is stored in the system, and she can view the order details and history. The Admin can manage all customer orders, update order statuses (pending, preparing, delivering, completed), and track operational workflows.

With **Delivery Tracking**, the Customer can follow the progress of her order from preparation to delivery. The Admin or delivery staff can update the current status so the Customer stays informed.

# II. THIẾT KẾ HỆ THỐNG VÀ KIẾN TRÚC

## 1. Thiết kế hệ thống

### 1.1. Workflow

A diagram of a food system

Description automatically generated

A diagram of a process flow

Description automatically generated

A diagram of a process

Description automatically generated

Figure 2. Workflow của Web Food Delivery

### 1.2. Activity

A screenshot of a diagram

Description automatically generated

Figure 3. Usecase tổng của Web Food Delivery

#### 1.2.1 Access Control

|  |  |  |
| --- | --- | --- |
| A diagram of a login and register  AI-generated content may be incorrect. | | |
| Usecase Number | UC01 | |
| Usecase Name | Access Controll | |
| Actor(s) | User(Sysadmin) User(Customer) | |
| Summary | hệ thống cho phép các tác nhân đăng nhập vào để xử dụng các tính năng của nó. nếu chưa có account thì hệ thống cho phép tạo mới tài khoản. | |
| Basic Flow | Actor(s) Actions | System Respone |
| 1. Chọn đăng nhập |  |
|  | 2. Hệ thống hiển thị Form để nhập thông tin tài khoản và mật khẩu (A1) |
| 3. actor nhập vào trường dữ liệu tài khoản và mật khẩu và nhấn button submit (E1) (A2) |  |
|  | 4. Hệ thống kiểm tra thông tin và nếu hợp lệ hệ thống sẽ chuyển hướng actor đến giao diện phù hợp. |
| Alternative Flow | Actor(s) Actions | System Respone |
| A1.1 Actor chọn đăng ký tài khoản |  |
|  | A1.2 Hệ thống hiển thị form đăng ký |
| A1.3 Actor nhập thông tin đăng kí email tài khoản mật khẩu |  |
| A2. Actor thể là SysAdmin/Customer |  |
| Exception Flow | Actor(s) Actions | System Respone |
|  | E1. Không thể xác thực được thông tin tài khoản actor đã nhập  E2. Hiển thị thông báo lỗi nếu không hợp lệ | |
| Pre-condition | None | |
| Asumption | None | |
| Business Rule | None | |
| A screenshot of a computer screen  Description automatically generated | | |
| SEQUENCE HERE | | |

#### 1.2.2 Food Management

|  |  |  |
| --- | --- | --- |
| A white oval with black text  AI-generated content may be incorrect. | | |
| Usecase Number | UC02 | |
| Usecase Name | Manage Food Menu | |
| Actor(s) | User(Sysadmin) | |
| Summary | Cho phép Quản trị viên truy cập vào Bảng điều khiển Quản trị để thực hiện các thao tác quản lý cơ bản đối với danh sách thực đơn/đồ ăn, bao gồm Thêm mới, Cập nhật và Xóa. | |
| Basic Flow | Actor actions | System response |
| 1. SysAdmin được chuyển hướng đến Bảng điều khiển Quản trị. |  |
|  | 2. Hệ thống hiển thị Bảng điều khiển Quản trị (Direct to Admin Panel). |
| 3. SysAdmin chọn chức năng Quản lý Thực đơn (Choose Menu Food List). |  |
|  | 4. Hệ thống hiển thị danh sách thực đơn hiện có |
| 5. SysAdmin không chọn thêm, cập nhật hoặc xóa. |  |
|  | 6. Hệ thống kết thúc phiên quản lý thực đơn. (Kết thúc Use Case) |
| Alternative flow | Actor(S) Action | System Response |
| A1.1. Thêm món mới (ADD) (Chạy từ Bước 4/Basic Flow) | |
| A1.2. SysAdmin chọn chức năng Thêm món mới | |
| A2.1. SysAdmin chọn một món ăn cần cập nhật và chọn chức năng Cập nhật | |
| A2.2. SysAdmin điều chỉnh các thông số cần thiết | |
| A3. SysAdmin chọn món ăn và chọn chức năng Xóa | |
| Exception flow | Actor action | System Response |
| **E1: Lỗi xác thực dữ liệu** (Chạy từ A1.3 hoặc A2.3) | |
| E1.1. SysAdmin nhập thông số không hợp lệ (Ví dụ: Tên món bị bỏ trống, giá âm). |  |
| **E2: Lỗi hệ thống** (Chạy từ A1.4, A2.4, hoặc A3.4) | |
| E2.1. Xảy ra lỗi kết nối cơ sở dữ liệu hoặc lỗi hệ thống khác. | |
| Pre-condition | SysAdmin đã đăng nhập thành công (Authentication). | |
| Post-condition | Danh sách thực đơn được cập nhật thành công (nếu thực hiện Thêm/Cập nhật/Xóa) hoặc SysAdmin kết thúc phiên làm việc. | |
| Business Rule | Chỉ SysAdmin mới có quyền truy cập chức năng này. - Các trường thông số thực đơn phải hợp lệ (Ví dụ: tên món không trùng lặp, giá phải là số dương). | |
| A screenshot of a phone  Description automatically generated | | |
| Sequence Here | | |

#### 1.2.3 Place Order (cart management)

|  |  |  |
| --- | --- | --- |
| A white oval with black text  AI-generated content may be incorrect. | | |
| Usecase Number | UC03s | |
| Usecase Name | Place Order | |
| Actor(s) | User(Customer) | |
| Summary | Cho phép Khách hàng đã đăng nhập duyệt qua các món ăn, thêm chúng vào giỏ hàng, xem lại giỏ hàng, điều chỉnh số lượng/xóa món, và cuối cùng tiến hành thanh toán. | |
| Basic flow | Actor action | System response |
|  | 1. actor thực hiện đăng nhập xác thực đăng nhập |  |
|  |  | 2. Hệ thống hiển thị Trang Khách hàng (Direct to customer site). |
|  | 3. Customer duyệt qua danh mục món ăn (Menu Catalog Browsing). |  |
|  | 4. Customer chọn và thêm món ăn vào giỏ hàng (Add to cart). |  |
|  |  | 5. Hệ thống tự động tính toán lại tổng tiền giỏ hàng (Auto ReCalculating). |
|  | 6. Customer chọn **Xem giỏ hàng** (View cart...). |  |
|  |  |  |
|  |  | 7. Hệ thống hiển thị danh sách các món trong giỏ hàng, số lượng, giá và Tổng kết đơn hàng (Calculate summary). |
|  | 8. Customer chọn Tiến hành Thanh toán (Process Checkout). |  |
| Alternative flow | Actor action | System flow |
|  | A1: Customer chọn Xóa bớt món (DELETE LESS). (chạy từ bước 8 basic flow) | |
|  | A2 : Customer chọn Thêm số lượng (ADD MORE) cho một món đã có. (chạy từ bước 8 basic flow) | |
| Exception flow |  | |
| Pre-condition | Customer đã đăng nhập thành công (Authentication). | |
| Post-condition | Đơn hàng của Customer được tạo và chuyển sang trạng thái chờ xử lý (hoặc đã thanh toán). | |
| Business Rule | Số lượng món ăn phải là số nguyên dương. - Giỏ hàng phải có ít nhất một món để tiến hành thanh toán. | |
| A screenshot of a phone  Description automatically generated | | |
| Sequence here | | |

#### 1.2.4. Menu Catalog Browsing

|  |  |  |
| --- | --- | --- |
| A white oval with black text  AI-generated content may be incorrect. | | |
| Usecase Number | UC04 | |
| Usecase Name | Menu Catalog Browsing | |
| Actor(s) | User(Customer) | |
| Summary | Cho phép Khách hàng xem danh sách các món ăn có sẵn, tùy chọn xem chi tiết từng món và thêm món đó vào giỏ hàng. | |
| Basic flow | Actor Acions | System Response |
|  | 1. Customer chọn chức năng **Duyệt Menu** (Browsing Menu). |  |
|  |  | 2. Hệ thống lấy dữ liệu menu từ cơ sở dữ liệu (Fetch Data from DB). |
|  |  | 3. Hệ thống hiển thị **Tất cả danh sách món ăn** (Show All List Item Food). |
|  | 4. Customer chọn nút **"Thêm vào Giỏ hàng"** trực tiếp từ danh sách món ăn. |  |
|  |  | 5. Hệ thống thêm món vào giỏ hàng của Customer. (Kết thúc Use Case) |
| Alternative flow | Actor action | System response |
|  | **A1: Xem chi tiết món ăn** (Chạy từ Bước 4/Basic Flow) | |
|  | A1.1. Customer chọn **Xem chi tiết** (View detail) của một món ăn. |  |
|  | A1.2. Customer quyết định mua món đó và chọn **"Thêm vào Giỏ hàng"** từ trang chi tiết. |  |
|  | **A2. Quay lại Menu** (Chạy từ A1.2) | |
|  | A2.1. Customer không thêm món vào giỏ và chọn **Quay lại Menu**. |  |
| Exception flow |  |  |
| Pre-condition | Customer đang truy cập trang web/ứng dụng và đã chọn chức năng xem menu. | |
| Post-condition | Món ăn đã được thêm thành công vào giỏ hàng của Customer. | |
| Business Rule | Chỉ có những món ăn còn khả dụng mới được hiển thị. - Việc thêm món sẽ kích hoạt việc tính toán lại tổng tiền giỏ hàng | |
| A screenshot of a cell phone  Description automatically generated | | |
| Sequence here | | |

#### 1.2.5 Order management

##### a) Process Payment

|  |  |  |
| --- | --- | --- |
| A diagram of a process payment  AI-generated content may be incorrect. | | |
| Usecase Number | UC05 | |
| Usecase Name | Process Payment | |
| Actor(s) | User(Customer) ; External\_Service(STRIPE) | |
| Summary | Cho phép Khách hàng tiến hành thanh toán cho đơn hàng của mình thông qua dịch vụ thanh toán bên ngoài (Stripe), sau đó hệ thống xác nhận và cập nhật trạng thái đơn hàng. | |
| Basic flow | Actor Action | System Response |
|  | 1. thực hiện đăng nhập authentication |  |
|  |  | 2. hệ thống chuyển hướng đến trang dành cho khách hàng. |
|  | 3. Customer hoàn tất quản lý giỏ hàng (Cart Management) và chọn **Thanh toán**. |  |
|  |  |  |
|  | 4. Stripe xử lý thanh toán (A1) |  |
|  |  |  |
|  |  | 5. Hệ thống thực hiện xác minh giao dịch bảo mật (payment verification secure transaction). |
|  |  | 6. Hệ thống cập nhật trạng thái thanh toán là **"Đã thanh toán"** (update payment status). |
|  |  | 7. Hệ thống hiển thị thông báo **"Thanh toán thành công"** (show success). |
|  | 8. Customer chọn **Xem chi tiết đơn hàng** |  |
|  | 9. Customer chọn **Theo dõi trạng thái giao hàng** (Tracking delivery status). |  |
| Alternative flow | A1. Đơn hàng có thể xử lý không thành công ? hệ thống quay lại bước 3 basic flow | |
| Exception flow |  | |
| Pre-condition | Customer đã đăng nhập và có các món hàng trong giỏ (Cart Management) | |
| Post-condition | Trạng thái thanh toán của đơn hàng được cập nhật thành công (Paid) và thông tin được lưu. | |

##### b) Manage Customer Orders

|  |  |  |
| --- | --- | --- |
| A white oval with black text  AI-generated content may be incorrect. | | |
| Usecase Number | UC06 | |
| Usecase Name | Manage Customer Orders | |
| Actor(s) | User(SysAdmin) | |
| Summary | Cho phép Quản trị viên xem tất cả các đơn hàng, cập nhật trạng thái xử lý đơn hàng và thông tin theo dõi giao hàng. | |
| Basic flow | Actor action | System response |
|  | 1. Thực hiện xác thực đăng nhập |  |
|  |  | 2. Chuyển hướng actor đến giao diện tương thích với vai trò |
|  | 3. SysAdmin chọn **Xem tất cả đơn hàng khách hàng** (view all customer order). |  |
|  | 4. SysAdmin chọn một đơn hàng và chọn **Cập nhật trạng thái** (update status order). |  |
|  | 5. SysAdmin chọn **Cập nhật theo dõi giao hàng** (update delivery tracking). |  |
|  |  | 6. Hệ thống cập nhật thông tin theo dõi giao hàng thành công. (Kết thúc Use Case) |
| Pre-condition | SysAdmin đã đăng nhập thành công và được chuyển đến Bảng điều khiển Quản trị (Show Admin Panel). | |
| Post-condition | Trạng thái đơn hàng và thông tin theo dõi giao hàng được cập nhật thành công. | |
| Sequence here | | |

A screenshot of a black screen

Description automatically generated

### 1.3. User story

#### 1.3.1. User Authentication

Register

* As a new customer
* I want to sign up with my name, email, and password
* So that I can securely access the food delivery platform.

Login/Logout

* As a returning customer
* I want to log in/out and manage my profile
* So that I can track my orders and re-order easily.

#### 1.3.2. Ordering & Cart

Browse the food items

* As a customer
* I want to browse the fast food menu with images, categories, and prices
* So that I can choose what I want to eat.

Shopping cart management

* As a customer
* I want to add, update, or remove food items from my cart
* So that I can review my order before checkout.

Payment process

* As a customer
* I want to see the subtotal, shipping fee, and total amount in the cart
* So that I know exactly how much I need to pay.

#### 1.3.3. Payment & Checkout

Thanh toán trực tuyến với Stripe

* As a customer
* I want to pay securely using credit/debit card via Stripe
* So that I can complete my order without cash.

Nhận xác nhận đơn hàng

* As a customer
* I want to receive a payment confirmation and order summary
* So that I know my order has been placed successfully.

#### 1.3.4. Delivery Tracking

Theo dõi tiến trình giao hàng

* As a customer
* I want to see real-time updates of my order status (placed, processing, on the way, delivered)
* So that I know when to expect my food.

Cập nhật trạng thái đơn hàng

* As a delivery staff
* I want to update delivery progress (e.g., picked up, on the way, delivered)
* So that customers are informed in real-time.

#### 1.3.5. Admin Management

Quản lý menu

* As an admin
* I want to add, edit, or delete food items (with name, price, category, image)
* So that the menu is always updated.

Quản lý đơn hàng

* As an admin
* I want to view, update, and manage customer orders
* So that I can ensure smooth delivery operations.

Quản lý người dùng

* As an admin
* I want to manage registered users and their activity
* So that I can maintain a secure system.

### 1.4. Conceptual Model

A diagram of a customer

AI-generated content may be incorrect.

Figure 4. Conceptual Model

#### a) Food System

* Food: chứa thông tin món ăn.
* SysAdmin (quản trị viên): có thể thực hiện các thao tác như CRUD (thêm/sửa/xóa món) và gán hàng tồn kho (Assign Inventory).
* Customer: có thể lọc (Filter) và xem món ăn.

#### b) Cart & OrderItem (khu màu vàng)

* Customer sở hữu Cart (giỏ hàng).
* Cart chứa nhiều OrderItem (các món cụ thể mà khách thêm vào giỏ).
* Mối quan hệ:
  + Một Cart có thể có nhiều OrderItem.
  + Một Customer chỉ sở hữu 1 Cart.

#### c) User & Role

* User là người dùng hệ thống (có thể là khách hàng hoặc admin).
* Mỗi User có thể có 1 hoặc nhiều Role (vai trò).

#### d) Order

* Customer có thể đặt một đơn hàng (Order).
* Mỗi Order thuộc về 1 Customer.
* Mối quan hệ là 1-1: một khách có thể đặt nhiều đơn, nhưng mỗi đơn chỉ thuộc về một khách.

#### e) Payment

* Customer sử dụng Stripe (cổng thanh toán) để thanh toán (Payment).
* Payment liên kết với:
  + PaymentStatus (trạng thái thanh toán)
  + Cart Info (thông tin giỏ hàng)
* Một khách có thể có nhiều lần thanh toán khác nhau.

#### f) Delivery

* Delivery thể hiện quá trình giao hàng.
* Customer có thể theo dõi (follow) quá trình giao hàng.
* SysAdmin có thể cập nhật (update) trạng thái giao hàng.
* DeliveryStatus là một Enum, có các trạng thái:
  + Food Processing
  + Out for Delivery
  + Delivered
  + Unlocked

### 1.5. Data model

#### 1.5.1. ERD

A diagram of a computer program

AI-generated content may be incorrect.

Figure 5. ERD mức quan niệm

Sơ đồ này mô tả ba thực thể chính: User (Người dùng), Food (Món ăn), và Order (Đơn hàng).

1. User (Người dùng): Được xác định bằng ID, có các thuộc tính như Name, Email, và Password. Nó cũng chứa thuộc tính đa giá trị cartData (dữ liệu giỏ hàng) bao gồm FoodID và Quantity.
2. Food (Món ăn): Được xác định bằng FoodID, có các thuộc tính như name, Price, và category.
3. Order (Đơn hàng): Được xác định bằng OrderID, có các thuộc tính như status, amount, address, và items.

**Mối Quan hệ:**

* Place: Thể hiện mối quan hệ User đặt Order (Một người dùng đặt nhiều đơn hàng).
* Contains: Thể hiện mối quan hệ Order chứa Food (Nhiều món ăn có trong nhiều đơn hàng, với thuộc tính Quantity đi kèm).

#### 1.5.2. LOGICAL

A black screen with white text

AI-generated content may be incorrect.

Figure 6. ERD mức Logic

Mô hình này là một Sơ đồ Thực thể – Quan hệ (ERD) đang ở giai đoạn sơ khai, tập trung vào mô hình quan hệ chuẩn (Relational Model).  
Nó xác định 4 thực thể chính: USER, ORDER, FOOD, và ORDER\_ITEM.

ORDER\_ITEM đóng vai trò là thực thể liên kết (Junction Entity), giúp thiết lập mối quan hệ Nhiều–Nhiều (N:M) giữa ORDER và FOOD  
(Một đơn hàng có nhiều món, một món có thể nằm trong nhiều đơn hàng).

Hiện tại, mối quan hệ giữa các thực thể chưa được vẽ (chưa có hình thoi nối),  
nhưng các mối quan hệ logic sẽ là:

* USER đặt ORDER → *(1:N)*
* ORDER có các ORDER\_ITEM → *(1:N)*
* FOOD được tham chiếu bởi các ORDER\_ITEM → *(1:N)*

# 1.6. Data dictionary

#### 1.6.1. User

| **Trường (Field)** | **Mô tả** | **Kiểu dữ liệu Mongoose** | **Yêu cầu (Required)** | **Ràng buộc/Giá trị mặc định** |
| --- | --- | --- | --- | --- |
| \_id | Khóa chính (Primary Key) | ObjectId | Có | Tự động tạo bởi MongoDB |
| name | Tên đầy đủ của người dùng | String | Có | - |
| email | Địa chỉ email của người dùng | String | Có | Unique (Duy nhất) |
| password | Mã băm của mật khẩu | String | Có | - |
| cartData | Dữ liệu giỏ hàng đang hoạt động | Object | Không | Default: {} (Đối tượng rỗng) |

#### 1.6.2. Food

| **Trường (Field)** | **Mô tả** | **Kiểu dữ liệu Mongoose** | **Yêu cầu (Required)** | **Ràng buộc/Giá trị mặc định** |
| --- | --- | --- | --- | --- |
| \_id | Khóa chính (Primary Key) | ObjectId | Có | Tự động tạo bởi MongoDB |
| userId | ID của người dùng đặt hàng | String | Có | Tham chiếu đến User.\_id |
| items | Danh sách các món ăn đã đặt | Array | Có | Nhúng dữ liệu: Mỗi phần tử là một đối tượng chứa foodId, quantity, v.v. |
| amount | Tổng số tiền của đơn hàng | Number | Có | - |
| address | Địa chỉ giao hàng | Object | Có | Chứa các trường như street, city, zipcode, v.v. |
| status | Trạng thái xử lý đơn hàng | String | Không | Default: "Food Processing" |
| date | Thời gian đặt hàng | Date | Không | Default: Date.now() (Thời gian hiện tại) |
| payment | Trạng thái thanh toán | Boolean | Không | Default: false (Chưa thanh toán) |

#### 1.6.3. Order

| **Trường (Field)** | **Mô tả** | **Kiểu dữ liệu Mongoose** | **Yêu cầu (Required)** | **Ràng buộc/Giá trị mặc định** |
| --- | --- | --- | --- | --- |
| \_id | Khóa chính (Primary Key) | ObjectId | Có | Tự động tạo bởi MongoDB |
| userId | ID của người dùng đặt hàng | String | Có | Tham chiếu đến User.\_id |
| items | Danh sách các món ăn đã đặt | Array | Có | Nhúng dữ liệu: Mỗi phần tử là một đối tượng chứa foodId, quantity, v.v. |
| amount | Tổng số tiền của đơn hàng | Number | Có | - |
| address | Địa chỉ giao hàng | Object | Có | Chứa các trường như street, city, zipcode, v.v. |
| status | Trạng thái xử lý đơn hàng | String | Không | Default: "Food Processing" |
| date | Thời gian đặt hàng | Date | Không | Default: Date.now() (Thời gian hiện tại) |
| payment | Trạng thái thanh toán | Boolean | Không | Default: false (Chưa thanh toán) |

## 2. Thiết kế kiến trúc

### 2.1. Sơ đồ khối (Block diagram)

A diagram of a software model

Description automatically generated with medium confidence

Figure- 3: Block diagram for fast food delivery website

Tại đây chúng tôi sử dụng ReactJS để xử lý frontend áp dụng khái niệm SPA (single pages Application) được đề cập trong Firgure-1, khi lần đầu truy cập trình duyệt tải index.html sau đó mọi điều hướng nội bộ xử lý bằng Client-Side-Routing và SPA gọi API Backend để lấy giao diện và render lên giao diện (UI).

Trong figure-3 ta thấy được ở phía backend ta xử dụng kiến trúc phân lớp dạng logic ngang (Kiến trúc phân lớp dưới dạng logic ngang là gì đã đề cập trước đó). Ở phần này trong backend request đi qua luồng ngang *Routes/API ->  Controllers ->   Models ->  Database.* Khi nhìn sơ đồ này ta ta chỉ thấy kiến trúc theo tầng kỹ thuật chứ không phân biệt rõ ràng chức năng nào nằm ở đâu. Trong khi đó kiến trúc logic dọc thì ngược lại (chúng tôi sẽ không đi sâu vào điều này ở đây).

### 2.2. Sơ đồ triển khai (Deployment View)

A screenshot of a computer

Description automatically generated

Figure 7. Deployment View

Ở đây chúng tôi triển khai theo mô hình CI/CD kết hợp với Docker và Kubernetes để đảm bảo tính tự động hoá, khả năng mở rộng về sau và dễ dàng quản lý. Quy trình được mô hình hoá qua.  figure-4. Giao đoạn phát triển (Development) chúng tôi sử dụng Visual Studio Code để phát triển website kết hợp với ngôn ngữ NodeJS. Mã nguồn được quản lý hệ thống Git và đẩy (push) code lên Github Repository. Khi có sự thay đổi về mã nguồn Github Actions sẽ kích hoạt Pineline, công việc của pineline là kiểm tra và chạy test cho mã nguồn, xây dựng Docker image từ file Dockerfile đã định nghĩa, đẩy (push) image này lên Docker Hub (hoặc GitHub Container Registry). Kubernetes Cluster sẽ lấy (pull) Docker image mới nhất từ Container Registry.

### 2.3. Sơ đồ giao tiếp API (Communication view)

A screenshot of a computer

AI-generated content may be incorrect.

Figure 8. Communication View.

*Figure 8. Communication View* cung cấp cái nhìn tổng quan toàn diện về các endpoint RESTful của Food Delivery API, hệ thống hóa chức năng của ứng dụng thành bốn miền chính tương ứng với các quy trình nghiệp vụ cốt lõi của hệ thống. Miền Giỏ hàng (Shopping Cart) cung cấp ba endpoint POST cho phép khách hàng quản lý trạng thái giỏ hàng của mình: /api/cart/get để truy xuất các món hiện có trong giỏ, /api/cart/add để thêm món ăn vào giỏ, và /api/cart/remove để xóa món ăn khỏi giỏ, tạo nên các thao tác cơ bản giúp khách hàng tùy chỉnh đơn hàng trước khi thanh toán. Miền Thực đơn (Food Menu) triển khai giao diện CRUD đầy đủ gồm /api/food/add cho quản trị viên thêm món ăn mới, /api/food/remove để xóa món ăn, và /api/food/list là endpoint GET cho phép cả khách hàng và quản trị viên lấy danh sách thực đơn, hỗ trợ lọc và sắp xếp linh hoạt. Miền Xác thực Người dùng (User Authentication) quản lý danh tính thông qua /api/user/register để đăng ký tài khoản mới và /api/user/login để xác thực người dùng hiện có, cả hai đều sử dụng xác thực dựa trên JWT kết hợp mã hóa mật khẩu bằng bcrypt nhằm đảm bảo an toàn truy cập. Miền Quản lý Đơn hàng (Order Management) điều phối toàn bộ vòng đời đơn hàng thông qua năm endpoint: /api/order/place để khởi tạo đơn hàng và xử lý thanh toán, /api/order/verify để xác nhận hoàn tất thanh toán, /api/order/userorders để truy xuất lịch sử đặt hàng của khách, /api/order/status để cập nhật trạng thái giao hàng, và /api/order/list để cung cấp cho quản trị viên cái nhìn tổng thể về tất cả đơn hàng phục vụ công tác quản lý vận hành. Cấu trúc API này thể hiện kiến trúc RESTful được tổ chức rõ ràng, phân tách hợp lý giữa các chức năng, sử dụng phương thức HTTP phù hợp (POST cho các thao tác thay đổi trạng thái, GET cho truy xuất dữ liệu), đồng thời cung cấp đầy đủ endpoint cần thiết để hỗ trợ cả chức năng hướng khách hàng và nghiệp vụ quản trị, tạo nên một giao diện nhất quán và hoàn chỉnh cho toàn bộ quy trình kinh doanh giao đồ ăn.

### 3. Decomposition View

### 3.1. Context diagram – C1

A diagram of a customer service

AI-generated content may be incorrect.

Figure 9. C1 - Context Diagram

Sơ đồ ngữ cảnh như Figure 9 thể hiện tất cả thực thể bên ngoài có thể tương tác với hệ thống bao gồm Customers, SysAdmin, hệ thống bên ngoài ở đây là Payment gateway. Toàn bộ hệ thống thể hiện như một quy trình duy nhất. Sơ đồ này mô tả trung tâm là Food Delivery System. Figure-6  mô tả yêu cầu của một hệ thống phân phối thức ăn nhanh. Cho phép khánh hàng đặt hàng trực tuyến đồng thời giá cả và thông tin của món hàng được tải lên giúp dễ dàng duyệt hàng và người quản trị hệ thống có thể quản lý hàng hoá, và các nghiệp vụ cơ bản (CRUD).

### 3.2. Container diagram – C2

Sơ đồ container cho ta thấy hình dạng tổng thể của kiến trúc hệ thống và các công nghệ chính được lựa chọn, cách chúng được sử dụng và cách các container giao tiếp với nhau. Đây là sơ đồ tập trung cao về công nghệ.

A diagram of a customer

AI-generated content may be incorrect.

Figure 10. C2 - Container Diagram

Dựa vào Figure 10 nó chỉ ra hệ thống Fast Food Delivery có 4 container tạo thành

* Web application (thuộc frontend)
* Single page application (thuộc frontend)
* Food delivery API (thuộc backend)
* Database (thuộc backend)

Web application là một ứng dụng web ReactJS bao gồm nội dung tạo nên SPA (Single page application). SPA chạy trong trình duyệt của khánh hàng. Cung cấp đầy đủ tính năng đặt hàng trực tuyến

Web application sử dụng API/HTTP được cung cấp bởi úng dụng của NodeJS + Express, API lấy thông tin từ database sau đó tạo payment bằng Stripe SDK (NodeJS). Stripe Payment Gateway cung cấp hạ tầng xử lý thanh toán thẻ và callback về backend API đễ xác nhận. Database lưu lại trạng thái.

### 3.3. Component Diagram (High-level)

A computer screen shot of a diagram

Description automatically generated

Figure 11. C3 - Component Diagram

Chúng tôi để đảm bảo khả năng bảo trì, mở rộng và phân trách nhiệm rõ ràng. Hệ thống backend của chúng tôi thể hiện kiến trúc phân tần cổ điển (hay còn gọi là MVC) được điều chỉnh cho việc hướng đến API.

Hãy nhìn vào Figure 11, ứng dụng được thế kế tuân theo 4 layer :

* API layer hay còn gọi là tầng Routes đóng vai trò là cổng API của ứng dụng tầng này được chia thành các file route riêng biệt. Mỗi file định nghĩa các HTTP Endpoint cho một thực thể nghiệp vụ cụ thể trong ứng dụng. Tầng routes chịu trách nhiệm ánh xạ (mapping) các yêu cầu HTTP đến các trình xử lý tương ứng. Khi một ứng dụng client gửi yêu cầu đến backend, nó sẽ đi qua tầng này đầu tiên — đóng vai trò như người điều phối lưu lượng, hướng từng yêu cầu đến đúng đường dẫn.
  + Ví dụ :
    - Một yêu cầu **POST** để tạo món ăn mới được foodRoute.js tiếp nhận.
    - Một yêu cầu **GET** để lấy giỏ hàng người dùng được cartRoute.js xử lý.
* Tầng Middleware đây là tầng xử lý các mối quan tâm xuyên suốt. thì ở tầng này giữa routes và controllers là middleware layer, được đại diện bởi file auth.js, middleware là lớp trung gian có thể kiểm tra, chỉnh sửa hoặc từ chối yêu cầu trước khi đến logic của controller. Nó xử lý các vấn đề bảo mật như :
  + Kiểm tra xem yêu cầu có chứa thông tin xác thực hợp lệ không.
  + Xác minh token cho phiên đăng nhập.
  + Đảm bảo chỉ người dùng được ủy quyền mới có thể truy cập tài nguyên bảo vệ

Middleware này đóng vai trò người gác cổng ngăn chặn truy cập trái phép vào các thao tác nhạy cảm. Bằng cách tập trung logic bảo mật trong middleware, ứng dụng tránh việc lặp lại mã xác thực ở nhiều controller. Khi một route cần xác thực, nó chỉ cần khai báo sử dụng middleware auth trước khi xử lý. Điều này tạo ra sự tách biệt rõ ràng giữa bảo mật và nghiệp vụ.

* Tầng controller – điều phối nghiệp vụ. Tầng này phản chiếu cấu trúc của routes, giúp duy trì tính nhất quán trong toàn ứng dụng. Controllers đóng vai trò điều phối viên nhận yêu cầu đã được xác thực, thực thi nghiệp vụ, tương tác với tầng dữ liệu, rồi định dạng phản hồi trả về client. Tầng controller cũng là ranh giới chuyển đổi, chuyển đổi các khái niệm HTTP (tham số, header, mã trạng thái) thành nghiệp vụ thuần túy, rồi ngược lại - từ kết quả nghiệp vụ sang phản hồi HTTP.
* Tầng models, ở tầng nền của ứng dụng là thư mục models, định nghĩa cấu trúc dữ liệu và cung cấp lớp trừu tượng hóa truy cập cơ sở dữ liệu. Mỗi model thường đại diện cho một bảng trong cơ sở dữ liệu, và bao gồm toàn bộ logic truy cập dữ liệu. Nhiệm vụ chính cho tầng này là định nghĩa schema dữ liệu.

### 3.4. Code Diagram-C4

A screenshot of a computer screen

Description automatically generated

Figure 12. C4 - Code Diagram

Sơ đồ C4 Code thể hiện ở figure- 9 đại bản mô tả hoàn chỉnh về cách mà hệ thống backend đặt đồ ăn được cấu trúc ở mức chi tiết nhất. Không giống như các sơ đồ trừu tượng chỉ thể hiện mối quan hệ khái niệm, sơ đồ này cho thấy chi tiết triển khai thực tế: các route cụ thể, method signatures, thuộc tính dữ liệu, và mạng lưới phụ thuộc phức tạp gắn kết toàn hệ thống.

Sơ đồ mô tả kiến trúc đa tầng kinh điển với sự tách biệt rõ ràng về trách nhiệm. Ở lớp ngoài cùng là các route (CartRoute, OrderRoute, FoodRoute, UserRoute) giao diện công khai của ứng dụng. Các route này được bảo vệ bởi middleware xác thực (AuthMiddleware). Dưới là các controller (CartController, OrderController, FoodController, UserController) chịu trách nhiệm điều phối logic nghiệp vụ. Ở tầng dữ liệu, bao gồm các model (CartModel, OrderModel, FoodModel, UserModel) tương tác với cơ sở dữ liệu thông qua DBConfig. Cuối cùng là AppConfig nơi chứa cấu hình toàn cục cho toàn ứng dụng.

* The Entry Points: Route Components
  + CartRoute: Thành phần CartRoute định nghĩa giao diện HTTP cho chức năng giỏ hàng. Cấu trúc của nó bao gồm bốn điểm cuối riêng biệt, mỗi điểm cuối tương ứng với một thao tác giỏ hàng cơ bản. Điểm cuối post("/get") truy xuất trạng thái giỏ hàng hiện tại của người dùng, cho phép giao diện người dùng hiển thị nội dung giỏ hàng. Điểm cuối post("/add") thêm các mặt hàng vào giỏ hàng, chấp nhận mã định danh sản phẩm và số lượng. Điểm cuối post("/update/:id") sửa đổi các mặt hàng hiện có trong giỏ hàng, thường thay đổi số lượng. Điểm cuối delete("/data") xóa hoàn toàn các mặt hàng khỏi giỏ hàng. Tất cả các tuyến này đều được đóng gói với Middleware.use(auth), cho biết rằng xác thực là bắt buộc đối với các thao tác giỏ hàng. Điều này hoàn toàn hợp lý về mặt bảo mật—giỏ hàng là tài nguyên dành riêng cho người dùng và chỉ chủ sở hữu mới có thể truy cập. Việc sử dụng xác thực nhất quán trên tất cả các điểm cuối giỏ hàng thể hiện một thiết kế bảo mật chu đáo.
  + OrderRoute: Thành phần OrderRoute xử lý đường dẫn quan trọng từ giỏ hàng đến khi hoàn tất đơn hàng. Nó hiển thị năm điểm cuối bao phủ toàn bộ vòng đời đơn hàng. Điểm cuối post("/place") khởi tạo việc tạo đơn hàng, chuyển đổi giỏ hàng thành một bản ghi đơn hàng cố định. Điểm cuối post("/verify") có thể xử lý việc xác minh thanh toán hoặc xác nhận đơn hàng. Điểm cuối post("/userorders") truy xuất lịch sử đơn hàng của người dùng. Điểm cuối get("/list") là hàm để liệt kê tất cả các đơn hàng. Điểm cuối post("/status") cho phép theo dõi đơn hàng bằng cách cập nhật hoặc truy vấn trạng thái đơn hàng. Giống như CartRoute, tất cả các điểm cuối đều được bảo vệ bởi phần mềm trung gian xác thực. Các đơn hàng chứa thông tin tài chính và cá nhân nhạy cảm, khiến việc xác thực trở nên không thể thương lượng. Sự đa dạng của các điểm cuối phản ánh sự phức tạp của việc quản lý đơn hàng—không chỉ là tạo đơn hàng mà còn là xác minh, theo dõi và xem lại lịch sử đơn hàng.
  + FoodRoute: Thành phần FoodRoute cung cấp một giao diện CRUD hoàn chỉnh cho danh mục thực phẩm. Bốn điểm cuối tuân thủ chặt chẽ các quy ước RESTful. Điểm cuối post("/add") tạo các mục thực phẩm mới, chấp nhận các chi tiết như tên, mô tả, giá cả và hình ảnh. Điểm cuối post("/list") truy xuất danh mục thực phẩm, có thể bao gồm lọc và phân trang. Điểm cuối post("/remove") xóa các mục thực phẩm khỏi danh mục. Điểm cuối post("/update") sửa đổi các mục thực phẩm hiện có. Điều thú vị là các tuyến đường này cũng sử dụng phần mềm trung gian xác thực. Trong hệ thống sản xuất, chúng ta có thể mong đợi các thao tác đọc (/list) được công khai trong khi các thao tác ghi (/add, /remove, /update) yêu cầu quyền quản trị viên. Phần mềm trung gian xác thực ở đây có thể phân biệt giữa người dùng thông thường và quản trị viên dựa trên vai trò người dùng.
  + UserRoute: Thành phần UserRoute quản lý tài khoản người dùng và xác thực. Nó định nghĩa bốn điểm cuối chính. Điểm cuối post("/login") xác thực người dùng, xác thực thông tin đăng nhập và cấp mã thông báo. Điểm cuối post("/register") tạo tài khoản người dùng mới. Điểm cuối post("/token") có thể làm mới mã thông báo xác thực hoặc xác thực mã thông báo hiện có. Cuối cùng, Middleware.use(auth) bảo vệ toàn bộ tuyến đường, mặc dù trên thực tế, các điểm cuối đăng nhập và đăng ký nên được truy cập công khai. Mâu thuẫn rõ ràng này—bảo vệ đăng nhập/đăng ký bằng xác thực—có thể chỉ ra rằng phần mềm trung gian được áp dụng có chọn lọc ở cấp phương thức thay vì cấp tuyến đường, hoặc một số điểm cuối nhất định bị loại trừ rõ ràng khỏi các kiểm tra xác thực.
* The Security Layer: AuthMiddleware

Thành phần AuthMiddleware là thành phần bảo vệ ứng dụng, triển khai xác thực dựa trên token. Cấu trúc của nó tiết lộ hoạt động bên trong của xác thực JWT (JSON Web Token).

Phương thức +authMiddleware(req,res,next) là điểm vào, tuân theo các quy ước của middleware Express.js. Phương thức này nhận yêu cầu, phản hồi và hàm next, cho phép nó cho phép, sửa đổi hoặc từ chối các yêu cầu. Phương thức +verify(String Token) xác thực chữ ký và ngày hết hạn của token. Phương thức +decode(token Token) trích xuất thông tin người dùng đã được mã hóa từ token.

Middleware này có thể kiểm tra tiêu đề Authorization của các yêu cầu đến, trích xuất token người mang, xác minh chữ ký mật mã của nó bằng khóa bí mật (từ AppConfig), kiểm tra dấu thời gian hết hạn của nó, giải mã payload để trích xuất thông tin người dùng và đính kèm thông tin này vào đối tượng yêu cầu để sử dụng ở hạ nguồn.

Bằng cách tập trung xác thực trong middleware, ứng dụng đảm bảo việc thực thi bảo mật nhất quán. Các tuyến không triển khai logic xác thực riêng của chúng; họ chỉ đơn giản tuyên bố rằng cần phải xác thực và phần mềm trung gian sẽ xử lý mọi thứ.

* The Orchestration Layer: Controllers
  + **CartController**: CartController điều phối các hoạt động giỏ hàng, kết nối các tuyến đường và mô hình. Các phương thức công khai của nó tương ứng với các điểm cuối tuyến đường: +addToCart(req,res) xử lý việc thêm sản phẩm, +removeFromCart(req,res) xử lý việc xóa sản phẩm, và +getListofCart(req,res) truy xuất nội dung giỏ hàng. Bộ điều khiển tương tác với CartModel thông qua mối quan hệ "use", gọi các phương thức mô hình để thực hiện các hoạt động cơ sở dữ liệu. Khi người dùng thêm một sản phẩm, bộ điều khiển sẽ xác thực yêu cầu (đảm bảo sản phẩm tồn tại và số lượng là số dương), gọi CartModel để duy trì thay đổi và trả về phản hồi đã được định dạng cho máy khách. Bộ điều khiển này có thể triển khai các quy tắc kinh doanh vượt ra ngoài các hoạt động CRUD đơn giản. Ví dụ: khi thêm một sản phẩm, nó có thể kiểm tra xem sản phẩm đã tồn tại trong giỏ hàng hay chưa và tăng số lượng thay vì tạo một mục trùng lặp. Khi tính tổng, nó có thể áp dụng các khoản giảm giá hoặc mã khuyến mại.
  + **OrderController**: OrderController quản lý logic nghiệp vụ phức tạp nhất trong hệ thống. Các phương thức public của nó xử lý toàn bộ vòng đời đơn hàng: +placeOrder(req,res) tạo đơn hàng, +verifyOrder(req,res) xác nhận đơn hàng, +userOrders(req,res) truy xuất lịch sử, +listOrders(req,res) cung cấp chế độ xem quản trị và +updateStatus(req,res) theo dõi tiến trình đơn hàng. Mối quan hệ "Sử dụng" với OrderModel biểu thị các tương tác cơ sở dữ liệu, nhưng bộ điều khiển có thể phối hợp nhiều mô hình. Việc đặt hàng yêu cầu đọc CartModel, tạo OrderModel, có thể cập nhật FoodModel (cho hàng tồn kho) và xóa giỏ hàng. Sự phối hợp đa mô hình này chính xác là những gì bộ điều khiển được thiết kế để xử lý. Quy trình đặt hàng có thể bao gồm một số bước: xác thực giỏ hàng có hàng, xác minh thông tin giao hàng đã đầy đủ, tính toán tổng số tiền cuối cùng (bao gồm thuế và phí), xử lý thanh toán thông qua cổng bên ngoài, tạo bản ghi đơn hàng một cách nguyên tử và gửi thông báo xác nhận. Bộ điều khiển điều phối tất cả những điều này, đảm bảo rằng nếu bất kỳ bước nào bị lỗi, toàn bộ hoạt động sẽ được khôi phục.
  + **FoodController**: FoodController quản lý danh mục sản phẩm bằng ba phương thức cốt lõi: +addFood(req,res) tạo sản phẩm, +listFood(req,res) truy xuất danh mục và +removeFood(req,res) xóa sản phẩm. Mối quan hệ "Sử dụng" với FoodModel hiển thị quyền truy cập dữ liệu, nhưng controller bổ sung logic nghiệp vụ lên trên. Việc thêm sản phẩm có thể bao gồm việc tải lên và xử lý hình ảnh, xác thực giá cả là số dương, đảm bảo tên là duy nhất và đặt giá trị mặc định cho các trường tùy chọn. Việc liệt kê sản phẩm có thể thực hiện phân trang, lọc theo danh mục, sắp xếp theo giá hoặc mức độ phổ biến, và loại trừ các sản phẩm không có sẵn. Việc xóa sản phẩm cần được cân nhắc kỹ lưỡng. Controller nên kiểm tra xem có đơn hàng đang hoạt động nào tham chiếu đến sản phẩm đó hay không. Nếu có, controller có thể ngăn việc xóa hoặc thực hiện xóa mềm (đánh dấu là không có sẵn thay vì xóa bản ghi) để bảo toàn tính toàn vẹn của lịch sử đơn hàng.
  + UserController: UserController xử lý tài khoản người dùng và xác thực bằng hai phương thức chính: +loginUser(req,res) xác thực người dùng và +registerUser(req,res) tạo tài khoản. Quá trình đăng nhập bao gồm việc xác thực email và mật khẩu đã được cung cấp, truy vấn UserModel để tìm người dùng qua email, so sánh mật khẩu đã cung cấp với mã băm đã lưu trữ (sử dụng bcrypt hoặc tương tự), tạo mã thông báo JWT chứa thông tin người dùng và thời gian hết hạn, và trả lại mã thông báo cho máy khách để sử dụng trong các yêu cầu tiếp theo. Việc đăng ký yêu cầu xác thực định dạng và tính duy nhất của email, đảm bảo đáp ứng các yêu cầu về độ mạnh của mật khẩu, băm mật khẩu trước khi lưu trữ, tạo bản ghi người dùng trong cơ sở dữ liệu và có thể gửi email chào mừng hoặc liên kết xác minh. Mối quan hệ "Sử dụng" với UserModel biểu thị các hoạt động cơ sở dữ liệu này, nhưng bộ điều khiển triển khai các quy tắc nghiệp vụ và biện pháp bảo mật bảo vệ tài khoản người dùng.
* The Configuration Layer: System Foundation
  + AppConfig: Thành phần AppConfig tập trung cấu hình trên toàn ứng dụng. Các phương thức của nó bao gồm +post: Number (xác định cổng máy chủ), +env: String (xác định môi trường như phát triển hoặc sản xuất), và có thể có các thiết lập khác không được hiển thị.
  + DBConfig: Thành phần DBConfig quản lý kết nối cơ sở dữ liệu bằng một phương thức hiển thị duy nhất: +connectDB(). Phương thức này thiết lập kết nối cơ sở dữ liệu, xử lý việc gộp kết nối, logic thử lại và xử lý lỗi.
* The Data Layer: Models and Persistence
  + OrderModel: OrderModel biểu diễn các giao dịch mua đã hoàn thành. mô hình đơn hàng phản ánh sự phức tạp của việc quản lý đơn hàng. Tính năng theo dõi trạng thái cho phép các tính năng như "theo dõi đơn hàng của tôi". Cờ thanh toán phân biệt giữa đơn hàng đã đặt và đã thanh toán. Mảng items lưu giữ ảnh chụp nhanh các sản phẩm tại thời điểm mua hàng, bảo vệ bản ghi đơn hàng khỏi các thay đổi trong danh mục sau này.
  + FoodModel: FoodModel định nghĩa cấu trúc sản phẩm. Mối quan hệ "Tham chiếu" từ OrderModel đến FoodModel chỉ ra rằng giỏ hàng lưu trữ các tham chiếu đến các mặt hàng thực phẩm thay vì sao chép dữ liệu thực phẩm.
  + UserModel: lưu trữ tài khoản người dùng với các thuộc tính bao gồm +name: String, +email: String, +password: String (đã băm) và +cartData: Object (có thể là giỏ hàng hoặc tham chiếu được nhúng).

# III. KẾ HOẠCH KIỂM THỬ

## 1. Giới thiệu

### 1.1 Mục đích

Tài liệu Test Plan này dành cho dự án Web Food Delivery nhằm hỗ trợ các mục tiêu sau:

* Xác định thông tin tổng quan của dự án và các thành phần phần mềm cần được kiểm thử.
* Liệt kê các yêu cầu cần kiểm thử (Requirements for Test) ở mức độ tổng quát.
* Đề xuất và mô tả các chiến lược kiểm thử sẽ được áp dụng trong dự án.
* Xác định nguồn lực cần thiết và ước lượng công sức kiểm thử.
* Liệt kê các sản phẩm bàn giao của hoạt động kiểm thử (test deliverables).

### 1.2 Bối cảnh (Background)

Hệ thống được phát triển là một ứng dụng Web Food Delivery, cho phép người dùng:

* Duyệt và tìm kiếm món ăn.
* Thực hiện các thao tác đặt món, quản lý giỏ hàng và thanh toán trực tuyến.
* Theo dõi trạng thái đơn hàng theo thời gian thực.

Hệ thống tích hợp với các dịch vụ bên ngoài như cổng thanh toán trực tuyến, hệ thống quản lý đơn hàng, và cơ sở dữ liệu lưu trữ thông tin người dùng và đơn hàng.

Sản phẩm được định hướng triển khai trên nền tảng web, hoạt động trên các trình duyệt phổ biến và được xây dựng như một sản phẩm độc lập phục vụ người dùng cuối.

### 1.3 Phạm vi (Scope)

|  |  |  |
| --- | --- | --- |
|  | Nhóm chức năng | Nội dung kiểm thử |
| 1 | Danh mục sản phẩm | Kiểm thử việc hiển thị danh sách sản phẩm, xem chi tiết sản phẩm. |
| 2 | Giỏ hàng | Kiểm tra quá trình thêm, chỉnh sửa, xóa sản phẩm trong giỏ hàng, tính toán tổng tiền và cập nhật dữ liệu đúng thời gian thực. |
| 3 | Quy trình thanh toán | - Kiểm tra quy trình xác nhận đơn hàng, xử lý thanh toán, và phản hồi kết quả giao dịch. |
| 4 | Quản lý đơn hàng | Kiểm thử khả năng hiển thị, xác nhận, cập nhật và thay đổi trạng thái đơn hàng bởi quản trị viên |
| 5 | Quản lý sản phẩm | Kiểm tra các thao tác của admin: thêm mới, chỉnh sửa, xóa và cập nhật sản phẩm trong cơ sở dữ liệu |
| 6 | Cơ sở dữ liệu | Kiểm thử tính toàn vẹn dữ liệu khi thêm/xóa/sửa bản ghi, đảm bảo dữ liệu đồng nhất giữa các thao tác frontend và backend. |

## ****1.4 Nhận diện dự án (Project Identification)****

Bảng dưới đây xác định các tài liệu và mức độ sẵn có được sử dụng để xây dựng **Test Plan**:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Tài liệu (kèm phiên bản / ngày)** | **Đã tạo hoặc sẵn có** | **Đã nhận hoặc đã rà soát** | **Tác giả / Nguồn** | **Ghi chú** |
| Tài liệu Đặc tả Yêu cầu (Requirements Specification) | Có | Có |  |  |
| Tài liệu Đặc tả Chức năng (Functional Specification) | Có | Có |  |  |
| Báo cáo Use Case (Use-Case Reports) | Không | Không |  |  |
| Kế hoạch Dự án (Project Plan) | Có | Có |  |  |
| Tài liệu Đặc tả Thiết kế (Design Specifications) | Có | Có |  |  |
| Nguyên mẫu (Prototype) | Có | Có |  |  |
| Tài liệu Hướng dẫn Người dùng (User’s Manuals) | Không | Không |  |  |
| Mô hình hoặc Luồng nghiệp vụ (Business Model or Flow) | Có | Có |  |  |
| Mô hình hoặc Luồng dữ liệu (Data Model or Flow) | Có | Có |  |  |
| Chức năng và Quy tắc nghiệp vụ (Business Functions and Rules) | Có | Có |  |  |
| Đánh giá Rủi ro Dự án hoặc Nghiệp vụ (Project or Business Risk Assessment) | Không | Không |  |  |

### ****1.5 Tài liệu tham khảo (References)****

Bảng dưới đây liệt kê các tài liệu bên ngoài được tham chiếu:

|  |  |  |
| --- | --- | --- |
| **STT** | **Tài liệu** | **Mô tả** |
| 1 | Tài liệu Đặc tả Yêu cầu Phần mềm dự án ABC (Software Requirements Specification), Phiên bản 1.0.3.3, Thứ Hai, 25/08/2010 | Tài liệu SRS mô tả đầy đủ hành vi bên ngoài của ứng dụng hoặc hệ thống con được xác định. Tài liệu cũng trình bày các yêu cầu phi chức năng, các ràng buộc thiết kế và các yếu tố khác cần thiết nhằm cung cấp một mô tả toàn diện và đầy đủ về các yêu cầu phần mềm. |
| 2 | Kế hoạch Phát triển Phần mềm dự án ABC (Software Development Plan), Phiên bản 1.0.5, ngày 05/09/2010 | Mục đích của Kế hoạch Phát triển Phần mềm là xác định các nhiệm vụ của dự án ABC (sau đây gọi là IM hoặc dự án) sẽ được triển khai. Tài liệu tập trung vào các năng lực mà các bên liên quan mong muốn, đặc biệt là người dùng mục tiêu, cũng như lý do cho các yêu cầu đó. |
| 3 |  |  |

## 2. Chức năng kiểm thử

### 2.1 Functionality Testing

#### 2.1.1 Form Functionality

* Kiểm tra tất cả các form hoạt động đúng chức năng.
* Xác minh các trường dữ liệu có validation phù hợp.
* Kiểm tra thông báo lỗi hiển thị khi nhập thiếu hoặc sai dữ liệu.
* Đảm bảo dữ liệu được gửi và lưu thành công sau khi submit.
* Kiểm tra các chức năng thêm, sửa, xem và xóa dữ liệu trong form.

#### 2.1.2 Link Testing

* Kiểm tra tất cả các liên kết hoạt động đúng và không bị lỗi.
* Xác minh các liên kết nội bộ và liên kết ngoài (API Stripe) hoạt động bình thường.
* Đảm bảo không tồn tại trang không thể truy cập từ hệ thống.

#### 2.1.3 Business workflow

* Kiểm tra luồng quy trình trọn vẹn (End-to-End) từ khi người dùng đăng nhập, tìm kiếm món ăn, thêm vào giỏ hàng đến khi thanh toán thành công.
* Xác minh tính nhất quán của dữ liệu (sản phẩm, số lượng, tổng tiền) khi người dùng di chuyển qua lại giữa các trang trong quá trình đặt hàng.
* Kiểm tra khả năng xử lý của hệ thống khi quy trình bị gián đoạn (ví dụ: người dùng hủy thanh toán giữa chừng hoặc quay lại trang trước) để đảm bảo không sinh ra lỗi logic
* Xác minh hệ thống điều hướng đúng trang đích (Redirect) sau khi hoàn thành mỗi bước nghiệp vụ (ví dụ: Đăng nhập xong về Trang chủ, Thanh toán xong về Trang xác nhận đơn).

## 2.2 Usability

#### 2.2.1 User Authentication (Xác thực người dùng)

##### 2.2.1.1. Account Registration – Sign Up (Đăng ký tài khoản)

* Kiểm tra rằng hệ thống cho phép người dùng tạo tài khoản mới một cách dễ dàng thông qua nút “Sign In” → “Create a new account”.
* Kiểm tra rằng hệ thống chấp nhận thông tin hợp lệ (tên, email, mật khẩu) và tự động đăng nhập hoặc chuyển hướng người dùng về trang chủ sau khi đăng ký thành công.
* Kiểm tra rằng hệ thống không hiển thị lỗi kỹ thuật khi đăng ký thành công.

##### 2.2.1.2. Error Handling for Existing Email (Xử lý lỗi email đã tồn tại)

* Kiểm tra rằng khi người dùng nhập email đã được đăng ký, hệ thống hiển thị thông báo lỗi rõ ràng và dễ hiểu (ví dụ: *“Email already exists”*).
* Kiểm tra rằng người dùng không bị mất dữ liệu đã nhập và vẫn ở trong giao diện đăng ký để chỉnh sửa thông tin.

##### 2.2.1.3. User Login (Đăng nhập)

* Kiểm tra rằng người dùng có thể đăng nhập thành công với thông tin hợp lệ.
* Kiểm tra rằng sau khi đăng nhập, nút “Sign In” được thay thế bằng biểu tượng hồ sơ người dùng hoặc tùy chọn đăng xuất, thể hiện trạng thái đăng nhập rõ ràng.

##### 2.2.1.4. Invalid Credentials Handling (Xử lý thông tin đăng nhập sai)

* Kiểm tra rằng khi người dùng nhập email hoặc mật khẩu không hợp lệ, hệ thống hiển thị thông báo lỗi thân thiện (ví dụ: *“Invalid credentials”*).
* Kiểm tra rằng người dùng vẫn được giữ ở giao diện đăng nhập để thử lại, không bị chuyển trang ngoài ý muốn.

#### 2.2.2. Menu Exploration & Selection (Khám phá và lựa chọn thực đơn)

##### 2.2.2.1. Category Filtering (Lọc theo danh mục)

* Kiểm tra rằng khi người dùng chọn một danh mục món ăn (ví dụ: *Salad*), hệ thống chỉ hiển thị các món thuộc danh mục đó.
* Kiểm tra rằng kết quả lọc được cập nhật ngay lập tức, không cần tải lại trang.

##### 2.2.2.2. Category Deselection (Bỏ chọn danh mục)

* Kiểm tra rằng người dùng có thể bỏ chọn danh mục đã chọn (hoặc chọn “All”) để hiển thị lại toàn bộ món ăn.
* Kiểm tra rằng giao diện phản hồi rõ ràng, tránh gây nhầm lẫn về trạng thái lọc.

##### 2.2.2.3. Add to Cart from Menu (Thêm món vào giỏ hàng)

* Kiểm tra rằng khi người dùng nhấn nút **“+”** trên thẻ món ăn, hệ thống thêm món vào giỏ thành công.
* Kiểm tra rằng bộ đếm số lượng hiển thị ngay trên thẻ món và/hoặc biểu tượng giỏ hàng được cập nhật.

##### 2.2.2.4. Quantity Adjustment (Điều chỉnh số lượng món)

* Kiểm tra rằng người dùng có thể tăng hoặc giảm số lượng món bằng các nút **“+”** và **“-”**.
* Kiểm tra rằng số lượng thay đổi được phản hồi ngay lập tức, không gây trễ hoặc sai lệch.

#### 2.2.3. Cart Management (Quản lý giỏ hàng)

##### 2.2.3.1. View Cart (Xem giỏ hàng)

* Kiểm tra rằng khi nhấn vào biểu tượng giỏ hàng, hệ thống hiển thị đầy đủ danh sách món đã chọn.
* Kiểm tra rằng thông tin về số lượng, giá từng món và tổng tiền được hiển thị chính xác, dễ đọc.

##### 2.2.3.2. Remove Item from Cart (Xóa món khỏi giỏ hàng)

* Kiểm tra rằng người dùng có thể xóa món ăn khỏi giỏ hàng thông qua nút **“Remove”** hoặc biểu tượng **“x”**.
* Kiểm tra rằng tổng tiền được cập nhật chính xác ngay sau khi xóa món.

##### 2.2.3.3. Proceed to Checkout (Chuyển sang thanh toán)

* Kiểm tra rằng nút **“PROCEED TO CHECKOUT”** dễ nhìn và dễ thao tác.
* Kiểm tra rằng người dùng được chuyển hướng đúng sang trang **“Place Order”**.

#### 2.2.4. Checkout And Order Placement (Thanh toán và đặt hàng)

##### 2.2.4.1. Delivery Information Validation (Kiểm tra thông tin giao hàng)

* Kiểm tra rằng hệ thống yêu cầu người dùng nhập đầy đủ các trường bắt buộc (tên, địa chỉ,, …).
* Kiểm tra rằng khi bỏ trống trường bắt buộc, hệ thống hiển thị thông báo hoặc đánh dấu lỗi trực quan, ngăn người dùng tiếp tục.

##### 2.2.4.2. Order Placement (Đặt hàng thành công)

* Kiểm tra rằng khi người dùng nhập đầy đủ thông tin và nhấn **“PROCEED TO PAYMENT”**, hệ thống xử lý đơn hàng thành công.
* Kiểm tra rằng người dùng được chuyển hướng tới trang xác nhận đơn hàng hoặc cổng thanh toán mô phỏng.

##### 2.2.4.3. Order History Visibility (Xem lịch sử đơn hàng)

* Kiểm tra rằng sau khi đặt hàng, đơn hàng mới xuất hiện trong mục **“My Orders”** hoặc **“Orders”** của hồ sơ người dùng.
* Kiểm tra rằng trạng thái đơn hàng (ví dụ: *“Food Processing”*) được hiển thị rõ ràng và dễ hiểu.

### 2.3 Interface Testing (Kiểm thử giao diện & Kết nối)

#### 2.3.1 Error Handling (Xử lý lỗi tương tác):

* Kiểm tra cơ chế bắt lỗi (Try-Catch) khi giao tiếp với MongoDB Atlas.
* Đảm bảo mọi thông báo lỗi từ Database (như Connection Timeout, Authentication Failed) đều được ứng dụng xử lý và hiển thị thông báo thân thiện cho người dùng (User-friendly message).

#### 2.3.2 Transaction Interruption (Gián đoạn giao dịch):

* Kiểm tra tính toàn vẹn dữ liệu khi người dùng hủy thao tác giữa chừng (ví dụ: tắt trình duyệt khi đang lưu dữ liệu). Đảm bảo không sinh ra dữ liệu rác hoặc dữ liệu trùng lặp trong Database.

#### 2.3.3 Connection Resilience (Độ ổn định kết nối):

* Kiểm tra hành vi của ứng dụng khi kết nối mạng bị ngắt đột ngột (Connection Reset). Ứng dụng cần có cơ chế thử kết nối lại (Retry mechanism) hoặc thông báo rõ ràng thay vì treo hệ thống.

### 2.4 Compatibility Testing (Kiểm thử tương thích)

#### 2.4.1 Browser Compatibility (Tương thích trình duyệt)

* Kiểm tra rằng giao diện website hiển thị đúng bố cục, không bị vỡ font hay sai lệch vị trí trên các trình duyệt phổ biến phiên bản mới nhất (Chrome, Safari, Edge).

#### 2.4.2 OS Compatibility (Tương thích hệ điều hành)

* Kiểm tra sự ổn định của ứng dụng web trên các hệ điều hành khác nhau (Windows, macOS).

### 2.5 Database Testing (Kiểm thử CSDL)

#### 2.5.1 Data Integrity (Toàn vẹn dữ liệu)

* Đảm bảo thông tin đơn hàng được lưu trữ đầy đủ các trường dữ liệu quan trọng (Mã đơn, ID khách hàng, Danh sách món, Tổng tiền) mà không bị mất mát.

#### 2.5.2 Data Consistency (Nhất quán dữ liệu)

* Đảm bảo rằng khi cập nhật/thêm/sửa/xóa thông tin món ăn, trang chủ sẽ hiển thị đúng thông tin vừa cập nhật

### 2.6 Performance Testing (Kiểm thử hiệu năng)

#### 2.6.1 Load Testing (Kiểm thử tải)

* Kiểm tra rằng thời gian tải trang chủ (Home Page) và trang admin (Admin Page) không vượt quá 3 giây trong điều kiện mạng tiêu chuẩn
* Kiểm tra khả năng phản hồi của hệ thống khi có nhiều người dùng (ví dụ: 500 users) cùng thực hiện hành động "Thêm vào giỏ hàng" đồng thời.

#### 2.6.2 Stress Testing (Kiểm thử chịu đựng)

* Kiểm tra hành vi của hệ thống khi lượng truy cập vượt quá giới hạn thiết kế (ví dụ: giờ cao điểm) để đảm bảo hệ thống có cơ chế xử lý lỗi an toàn thay vì bị sập hoàn toàn (Crash).

## 3. Chức năng chưa kiểm thử

### ****3.1. Kiểm thử khả năng mở rộng (Scalability Testing)****

* Khả năng mở rộng theo chiều ngang (scale-out)
* Hiệu năng khi triển khai nhiều instance backend
* Phân phối tải khi có load balancer
* Hiệu năng khi số lượng người dùng tăng liên tục trong thời gian dài (soak testing)

### ****3.2. Kiểm thử bảo mật (Security Testing)****

Hệ thống hiện **chưa được kiểm thử chuyên sâu** các khía cạnh bảo mật, bao gồm:

* SQL Injection / NoSQL Injection
* Cross-Site Scripting (XSS)
* Cross-Site Request Forgery (CSRF)
* Brute-force attack đối với chức năng Login
* Kiểm tra phân quyền chi tiết (Role-based access control) giữa User và Admin
* Kiểm thử lộ thông tin nhạy cảm trong response (JWT, stack trace, error log)

## ****4. Chiến lược kiểm thử (Test Strategy)****

### ****4.1 Các loại kiểm thử (Testing Types)****

#### **4.1.1 Kiểm thử Chức năng và Khả năng sử dụng (Function and Usability Testing)**

|  |  |
| --- | --- |
| **Nội dung** | **Mô tả** |
| **Mục tiêu kiểm thử (Test Objective)** | Đảm bảo các chức năng của đối tượng kiểm thử hoạt động đúng, bao gồm điều hướng, nhập dữ liệu, xử lý và truy xuất dữ liệu. |
| **Kỹ thuật (Technique)** | Thực hiện từng use case, luồng use case hoặc chức năng với dữ liệu hợp lệ và không hợp lệ để xác minh các nội dung sau:  • Kết quả mong đợi xuất hiện khi sử dụng dữ liệu hợp lệ.  • Các thông báo lỗi hoặc cảnh báo phù hợp được hiển thị khi sử dụng dữ liệu không hợp lệ. |
| **Tiêu chí hoàn thành (Completion Criteria)** | • Tất cả các ca kiểm thử đã được lập kế hoạch đều đã được thực hiện.  • Tất cả các lỗi (defects) đã được xác định đều đã được khắc phục. |
| **Lưu ý đặc biệt (Special Considerations)** | Xác định hoặc mô tả các yếu tố hoặc vấn đề (bên trong hoặc bên ngoài) ảnh hưởng đến việc triển khai và thực hiện kiểm thử chức năng. |

#### **4.1.2 Kiểm thử Giao diện (Interface Testing)**

|  |  |
| --- | --- |
| **Nội dung** | **Mô tả** |
| **Mục tiêu kiểm thử (Test Objective)** | Đảm bảo việc giao tiếp giữa các thành phần của hệ thống (Frontend – Backend – Database – Dịch vụ bên ngoài) diễn ra ổn định, dữ liệu được truyền và xử lý chính xác, đồng thời các lỗi giao tiếp được xử lý phù hợp. |
| **Kỹ thuật (Technique)** | Thực hiện kiểm thử các tình huống giao tiếp giữa ứng dụng và Database/API, bao gồm các trường hợp kết nối thành công và thất bại; mô phỏng lỗi kết nối, timeout hoặc gián đoạn giao dịch để đánh giá khả năng xử lý lỗi của hệ thống. |
| **Tiêu chí hoàn thành (Completion Criteria)** | • Các luồng giao tiếp chính hoạt động ổn định.  • Không phát sinh lỗi nghiêm trọng gây mất dữ liệu hoặc treo hệ thống. |
| **Lưu ý đặc biệt (Special Considerations)** | Phụ thuộc vào độ ổn định của mạng, Database và các dịch vụ bên thứ ba; một số lỗi khó tái hiện trong môi trường kiểm thử. |

#### **4.1.3 Kiểm thử Tính tương thích (Compatibility Testing)**

|  |  |
| --- | --- |
| **Nội dung** | **Mô tả** |
| **Mục tiêu kiểm thử (Test Objective)** | Đảm bảo hệ thống Web Food Delivery hoạt động ổn định và hiển thị đúng trên các trình duyệt và hệ điều hành khác nhau. |
| **Kỹ thuật (Technique)** | Thực hiện kiểm thử thủ công trên các trình duyệt phổ biến (Chrome, Safari, Edge) và các hệ điều hành khác nhau (Windows, macOS) với cùng một bộ chức năng. |
| **Tiêu chí hoàn thành (Completion Criteria)** | • Giao diện hiển thị nhất quán. • Không xảy ra lỗi chức năng do khác biệt môi trường. |
| **Lưu ý đặc biệt (Special Considerations)** | Sự khác biệt về phiên bản trình duyệt và độ phân giải màn hình có thể ảnh hưởng đến kết quả hiển thị. |

#### **4.1.4 Kiểm thử Cơ sở dữ liệu (Database Testing)**

|  |  |
| --- | --- |
| **Nội dung** | **Mô tả** |
| **Mục tiêu kiểm thử (Test Objective)** | Đảm bảo dữ liệu được lưu trữ, cập nhật và truy xuất trong cơ sở dữ liệu một cách chính xác, đầy đủ và nhất quán. |
| **Kỹ thuật (Technique)** | Thực hiện kiểm thử dựa trên các thao tác nghiệp vụ từ giao diện và đối chiếu trực tiếp dữ liệu trong Database; kiểm tra các truy vấn đọc/ghi dữ liệu. |
| **Tiêu chí hoàn thành (Completion Criteria)** | • Không xảy ra mất mát hoặc sai lệch dữ liệu. • Dữ liệu hiển thị trên giao diện khớp với dữ liệu trong Database. |
| **Lưu ý đặc biệt (Special Considerations)** | Cần quyền truy cập Database để kiểm tra; dữ liệu test phải được tách biệt với dữ liệu thật. |

#### **4.1.5 Kiểm thử Hiệu năng (Performance Testing)**

|  |  |
| --- | --- |
| **Nội dung** | **Mô tả** |
| **Mục tiêu kiểm thử (Test Objective)** | Đánh giá khả năng đáp ứng và độ ổn định của hệ thống khi chịu tải người dùng đồng thời. |
| **Kỹ thuật (Technique)** | Thực hiện Load Testing và Stress Testing bằng cách mô phỏng nhiều người dùng truy cập và thao tác đồng thời trên hệ thống. |
| **Tiêu chí hoàn thành (Completion Criteria)** | • Thời gian phản hồi nằm trong ngưỡng cho phép. • Hệ thống không bị sập khi tải cao. |
| **Lưu ý đặc biệt (Special Considerations)** | Kết quả phụ thuộc vào cấu hình server và môi trường kiểm thử. |

### 

### 4.2 Tools

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Tool** | **Vendor/In-house** | **Version** |
| Performance testing | JMeter |  |  |
| Database testing | Python script + Selenium |  |  |

## 5. Kiểm thử đơn vị (Unit Testing)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **Scenario** | **Test Description** | **Test Data** | **Expected result** |
| **UTC-01** | Kiểm tra thiếu thông tin bắt buộc | Gọi hàm lưu order không có số điện thoại | phone: "" | Trả về lỗi 400: "Missing info", không gọi Stripe. |
| **UTC-02** | Xác thực định dạng Email | Nhập email sai cú pháp | email: "tester.com" | Trả về lỗi 400: "Invalid email format". |
| **UTC-03** | Kiểm tra hạn mức thanh toán | Nhập số tiền thanh toán quá thấp | amount: 1000 | Trả về lỗi 400: "Amount too low". |
| **UTC-04** | Giả lập lỗi từ Stripe (Mocking) | Mô phỏng Stripe trả về lỗi kết nối | Stripe: Network Error | Hàm catch được lỗi, trả về 500: "Payment System Error". |

|  |  |
| --- | --- |
| **Nội dung** | **Mô tả** |
| **Mục tiêu (Test Objective)** | Đảm bảo các hàm logic xử lý dữ liệu đầu vào và các ràng buộc nghiệp vụ hoạt động chính xác. |
| **Kỹ thuật (Technique)** | Sử dụng **Mocking** để giả lập các service bên thứ 3 (Stripe) và Database nhằm cô lập logic hàm. |
| **Tiêu chí hoàn thành** | 100% các hàm quan trọng được test. Tỷ lệ bao phủ (Coverage) đạt trên 80%. |

## 6. Kiểm thử tích hợp (Integration Testing)

|  |  |  |  |
| --- | --- | --- | --- |
| **Bước thực hiện** | **Mong đợi (UI/Flow)** | **Integration Check (API/DB)** | **Test Case Name** |
| 1. Điền thông tin thanh toán & nhấn Đặt hàng | Giao diện hiển thị trạng thái "Processing" | API: Nhận Request, xác thực Token và gửi yêu cầu sang Stripe. | **ITC-01: Thanh toán và Tạo đơn hàng** |
| 2. Nhận kết quả từ Stripe | Chuyển hướng sang trang "Order Success" | Stripe: Trả về session\_id. API nhận mã thành công (201). |  |
| 3. Kiểm tra lưu trữ Database | Hiển thị đơn hàng mới trong mục "My Orders" | DB: Tạo bản ghi mới trong bảng Orders với trạng thái isPaid: true. |  |
| 4. Kiểm tra trạng thái Giỏ hàng | Giỏ hàng của người dùng được làm trống | DB Sync: Bản ghi giỏ hàng tương ứng trong bảng Carts bị xóa bỏ. |  |

|  |  |
| --- | --- |
| **Nội dung** | **Mô tả** |
| **Mục tiêu (Test Objective)** | Xác thực luồng dữ liệu đi xuyên suốt qua nhiều module và đảm bảo tính nhất quán dữ liệu giữa API và DB. |
| **Kỹ thuật (Technique)** | Thực hiện kiểm thử đầu cuối (End-to-End) trên môi trường Staging/Dev có kết nối Database thật. |
| **Lưu ý đặc biệt** | Cần kiểm tra kỹ cơ chế mã hóa (Hash) mật khẩu và tính toàn vẹn của Token JWT trong quá trình truyền tin. |

# IV. PHƯƠNG PHÁP KIỂM THỬ

## 1. Tổng quan

Kiểm thử phần mềm là một giai đoạn không thể thiếu trong vòng đời phát triển hệ thống, nhằm đảm bảo sản phẩm phần mềm đáp ứng đúng yêu cầu đặt ra, hoạt động ổn định và đạt chất lượng mong muốn trước khi đưa vào sử dụng. Việc lựa chọn và áp dụng phương pháp kiểm thử phù hợp giúp phát hiện sớm các lỗi tiềm ẩn, giảm thiểu chi phí sửa lỗi và nâng cao độ tin cậy của hệ thống.

Trong đồ án này, các phương pháp kiểm thử được xây dựng dựa trên đặc điểm của hệ thống, yêu cầu nghiệp vụ và phạm vi triển khai. Mỗi phương pháp kiểm thử đóng vai trò khác nhau trong việc đánh giá chất lượng phần mềm, từ việc xác minh tính đúng đắn của 2. Phân tích khung nhìn V-Models

Hình minh họa thể hiện mô hình V-Model, một mô hình phát triển phần mềm mở rộng từ mô hình thác nước (Waterfall), trong đó hoạt động phát triển (Verification) và hoạt động kiểm thử (Validation) được liên kết chặt chẽ với nhau theo từng giai đoạn tương ứng.

A diagram of a software development process

AI-generated content may be incorrect.

Figure 13. Khung nhìn V-models

### 2.1. Cấu trúc tổng thể của V-Model

Mô hình có dạng chữ V, bao gồm hai nhánh chính:

* Nhánh trái – Verification (Xác minh)  
  Đại diện cho vòng đời của lập trình viên (Developer’s life cycle), tập trung vào việc phân tích yêu cầu và thiết kế hệ thống.
* Nhánh phải – Validation (Thẩm định)  
  Đại diện cho vòng đời của kiểm thử viên (Tester’s life cycle), tập trung vào việc kiểm tra và xác nhận sản phẩm phần mềm thông qua các mức kiểm thử khác nhau.

Điểm đáy của chữ V là giai đoạn Coding, nơi mã nguồn được hiện thực dựa trên các thiết kế đã xác định.

### 2.2. Phân tích từng cặp giai đoạn đối xứng

Một đặc điểm quan trọng của V-Model là mỗi giai đoạn phát triển đều có một giai đoạn kiểm thử tương ứng, và test case được thiết kế song song từ sớm, không chờ đến khi code xong.

#### 2.2.1 Business Requirement Specification ↔ Acceptance Testing

* Business Requirement Specification  
  Xác định các yêu cầu nghiệp vụ ở mức cao từ phía khách hàng/người dùng.
* Acceptance Testing  
  Kiểm tra xem hệ thống cuối cùng có đáp ứng đúng các yêu cầu nghiệp vụ ban đầu hay không. (có các file Business Workflow)

Xác nhận phần mềm **phù hợp với mong đợi người dùng.**

#### 2.2.2 System Requirement Specification ↔ System Testing

* **System Requirement Specification**

Mô tả chi tiết các yêu cầu chức năng và phi chức năng của toàn bộ hệ thống.

* **System Testing**

Kiểm thử hệ thống như một khối hoàn chỉnh, đánh giá toàn bộ chức năng và hành vi của hệ thống. (có các file đặc tả như Test Desgin, Test Excution, Test case)

Mục tiêu: đảm bảo hệ thống hoạt động đúng theo đặc tả kỹ thuật.

#### 2.2.3. High Level Design ↔ Integration Testing

* **High Level Design (HLD)**  
  Thiết kế kiến trúc tổng thể, các module chính và cách chúng tương tác với nhau.
* **Integration Testing**  
  Kiểm tra việc tích hợp giữa các module, đảm bảo các thành phần giao tiếp đúng và không phát sinh lỗi. (có các file về Integration Test)

Mục tiêu: phát hiện lỗi ở giao diện và luồng dữ liệu giữa các module.

#### 2.2.4. Low Level Design ↔ Unit Testing

* **Low Level Design (LLD)**  
  Thiết kế chi tiết từng module, hàm, lớp và cấu trúc dữ liệu.
* **Unit Testing**Kiểm thử từng đơn vị nhỏ nhất của phần mềm (function, method, class).

Mục tiêu: đảm bảo từng thành phần hoạt động chính xác một cách độc lập. (có các file về Unit Test)

### 2.3. Ý nghĩa của Coding trong V-Model

Giai đoạn Coding nằm ở đáy mô hình, là nơi:

* Mã nguồn được hiện thực dựa trên React và Node JS
* Sau khi code xong, các mức kiểm thử được thực hiện theo thứ tự từ Unit → Integration → System → Acceptance

## 3. Phân tích khung nhìn Agile CI/CD

### 3.1 Tổng quan về Agile CI/CD

Agile CI/CD là cách tiếp cận phát triển phần mềm hiện đại, kết hợp giữa phương pháp Agile và quy trình tích hợp – triển khai liên tục (Continuous Integration / Continuous Delivery). Mục tiêu chính của Agile CI/CD là rút ngắn chu kỳ phát triển, tăng khả năng phản hồi với thay đổi yêu cầu và đảm bảo chất lượng phần mềm thông qua việc kiểm thử và triển khai tự động.

Khác với các mô hình phát triển tuyến tính, Agile CI/CD nhấn mạnh vào việc phát triển lặp – gia tăng (iterative & incremental), trong đó sản phẩm được xây dựng và cải tiến liên tục qua nhiều vòng lặp ngắn (iteration/sprint).

A diagram of a software development process

AI-generated content may be incorrect.

Figure 14. Agile CI/CD

### 3.2. Continuous Integration (CI)

CI (Continuous Integration) được hiểu theo nghĩa “tích hợp liên tục”. Nó đòi hỏi các thành viên trong đội ngũ phát triển ứng dụng phải tích hợp công việc với nhau một cách liên tục, thường xuyên. Cần ít nhất 1 tích hợp mỗi ngày. Điều này nhằm giúp phát hiện các vấn đề phát sinh một cách nhanh nhất.

Việc sử dụng các CI sẽ giúp cho các nhà phát triển thúc đẩy quá trình triển khai, phát triển phần mềm nhanh hơn và hiệu quả hơn. Quy trình làm việc của CI diễn ra qua các bước như sau:

* Các Developer sẽ đưa mã cam kết (Commit code) của nhà phát triển lên kho (Repository)
* Sau đó, CI Server thực hiện giám sát trên kho và kiểm tra tổng thể xem có vấn đề gì xảy ra không để có phương án xử lý kịp thời.
* Trong trường hợp có vấn đề cần xử lý, CI server sẽ code mới ngay tại kho Repository rồi chuyển sang build, chạy các unit và integration test. Tiếp đến CI server sẽ tạo và gửi các phản hồi lại cho từng thành viên trong team. Quá trình làm việc này diễn ra liên tục và xảy ra bất cứ thời điểm nào trong ngày.

### 3.3. Continuous Delivery/Deployment (CD)

CD (Continuous Deployment) có nghĩa là triển khai liên tục. So với CI thì CD thực hiện các nhiệm vụ cao cấp hơn. CD thực hiện việc kiểm tra tất cả những thay đổi về code được build và code trong môi trường kiểm thử.

Nó cho phép các Coder tự động hóa phần mềm testing và kiểm tra qua nhiều thước đo trước khi triển khai. Rất nhiều bài testing cần thực hiện như: UI testing, integration testing, API testing,...

CD sẽ sử dụng Deployment Pipeline để chia nhỏ quy trình chuyển giao thành các giai đoạn nhỏ khác nhau. Từng giai đoạn sẽ có mục tiêu riêng, giúp xác minh chất lượng của các tính năng để tránh được các lỗi phát sinh gây ảnh hưởng tới quá trình trải nghiệm của người dùng.

### ****3.4. Agile trong CI/CD****

Trong Agile CI/CD, quy trình phát triển và kiểm thử được tích hợp chặt chẽ theo chu trình lặp như sau:

1. Lập kế hoạch (Plan)  
   Xác định yêu cầu dưới dạng user story, ưu tiên theo giá trị nghiệp vụ.
2. Phát triển (Code)  
   Lập trình viên triển khai chức năng theo sprint ngắn, thường xuyên cập nhật mã nguồn.
3. Tích hợp liên tục (Continuous Integration)  
   Mỗi lần thay đổi mã nguồn đều được tích hợp vào hệ thống chung, kèm theo việc build và kiểm tra tự động.
4. Kiểm thử liên tục (Continuous Testing)  
   Các bài kiểm thử được thực thi thường xuyên nhằm phát hiện lỗi sớm trong quá trình phát triển.
5. Triển khai liên tục (Continuous Delivery/Deployment)  
   Phiên bản phần mềm có thể được triển khai nhanh chóng đến môi trường kiểm thử hoặc môi trường sử dụng thực tế.

Chu trình này được lặp lại liên tục cho đến khi sản phẩm hoàn thiện.

### 3.5. Triền Khai CI/CD

CI/CD cách hoạt động :

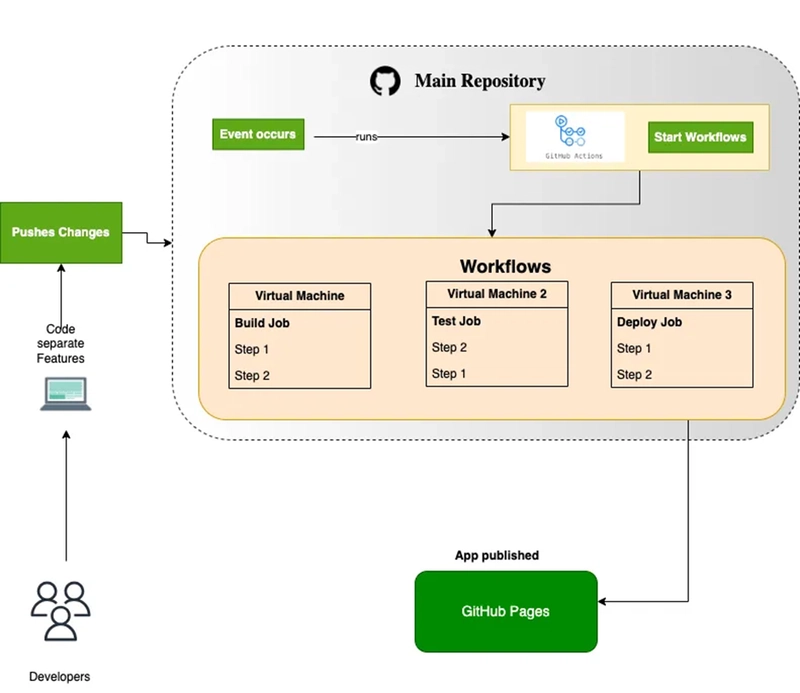


Figure 15. Triển khai CI/CD

Là một nền tảng CI/CD phổ biến, GitHub Actions cho phép các nhà phát triển tự động hóa quy trình xây dựng, kiểm thử và triển khai ứng dụng trực tiếp trong kho lưu trữ GitHub của họ.

Cốt lõi của GitHub Actions là các quy trình công việc, các quy trình tự động thực thi một hoặc nhiều tác vụ được xác định trước.

Các quy trình làm việc có thể được tùy chỉnh để phù hợp với các nhu cầu cụ thể, chẳng hạn như:

* Chạy thử nghiệm cho mỗi yêu cầu kéo (pull request).
* Tự động triển khai mã đã hợp nhất lên môi trường sản xuất

Các quy trình này được định nghĩa trong một tệp YAML được lưu trữ trong kho lưu trữ của bạn, đảm bảo rằng việc tự động hóa được tích hợp liền mạch vào quy trình phát triển của bạn.

- *Buóc 1* : Tạo dự án Node.js của bạn

A screenshot of a computer

Description automatically generated

Trong hướng dẫn này, tôi đã tạo sẵn một ứng dụng Node.js. Bạn có thể tự tạo một ứng dụng hoặc sử dụng một dự án hiện có.

- *Bước 2*: Tạo kho lưu trữ GitHub

Khởi tạo kho lưu trữ Git trong dự án của bạn, Đẩy dự án của bạn lên GitHub (nếu bạn chưa làm)

- *Bước 3*: Thiết lập các biến bí mật trong GitHub

1. Hãy truy cập vào kho lưu trữ GitHub của bạn.
2. Nhấp vào tab Cài đặt.
3. Trong thanh bên trái, điều hướng đến Bí mật và biến → Hành động.
4. Nhấp vào "Thêm bí mật kho lưu trữ mới".
5. Nhập tên bí mật.
6. Dán giá trị bí mật (ví dụ: khóa riêng SSH, mã thông báo API) vào ô tương ứng.
7. Nhấp vào Thêm bí mật.

A screenshot of a computer

Description automatically generated

- *Bước 4*: Tạo quy trình làm việc GitHub Actions

Bây giờ, hãy định nghĩa quy trình CI/CD của bạn trong tệp đó

A screen shot of a computer program

Description automatically generated

Nếu bạn muốn thêm bước đẩy bản dựng lên Docker Hub, bạn có thể tạo một tác vụ riêng biệt như ví dụ bên dưới. Tác vụ này sẽ xử lý việc xây dựng ảnh Docker và đẩy nó lên kho lưu trữ Docker Hub của bạn….

A screen shot of a computer

Description automatically generated

*- Bước 5* : chuẩn bị đường chạy

Khi làm việc với GitHub Actions, runner là môi trường nơi các quy trình công việc của bạn được thực thi. Có 2 loại runner: Các Runner được lưu trữ trên GitHub và Người chạy tự túc

* Các Runner do GitHub

cung cấp và quản lý. Bạn không cần thiết lập hay bảo trì bất cứ thứ gì, GitHub sẽ tự động tạo ra một môi trường mới, an toàn và biệt lập cho mỗi tác vụ trong quy trình làm việc của bạn. Chúng rất phù hợp với hầu hết các trường hợp sử dụng CI/CD và có thể sử dụng với nỗ lực tối thiểu

Bạn có thể lựa chọn từ các môi trường như :

* Ubuntu-lastest
* windows-latest
* macos-latest
* **Các Runner cá nhân**

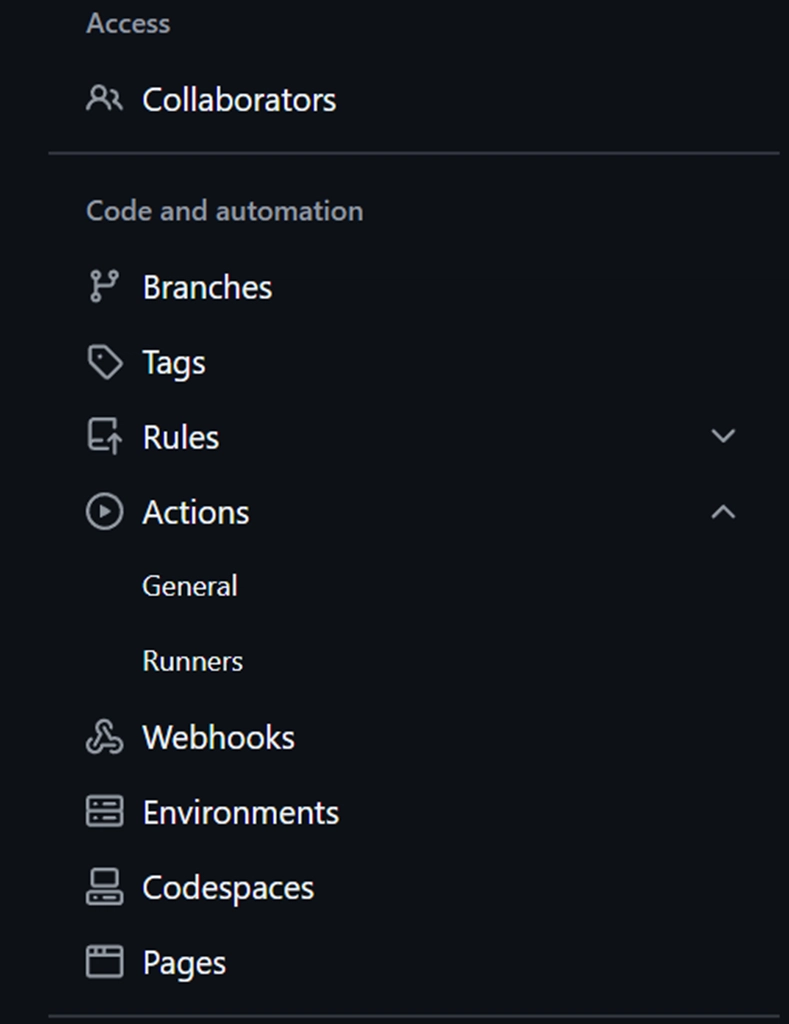
Lưu trữ là những máy ảo do bạn tự quản lý, giống như VPS cá nhân, máy chủ cục bộ hoặc máy ảo đám mây, và bạn đăng ký chúng với kho lưu trữ hoặc tổ chức GitHub của mình. Sau khi cấu hình xong, GitHub có thể chuyển các tác vụ quy trình công việc sang cơ sở hạ tầng riêng của bạn.

Điều này cho phép bạn kiểm soát hoàn toàn môi trường, nghĩa là bạn có thể cài đặt trước các thư viện phụ thuộc, phân bổ thêm bộ nhớ và tinh chỉnh nó theo đúng nhu cầu của mình. Chúng đặc biệt hữu ích trong các môi trường riêng tư hoặc hạn chế, nơi các máy chủ ảo được lưu trữ công khai trên GitHub không phù hợp.

Chúng cũng cung cấp khả năng thực thi nhanh hơn cho các bản dựng lặp lại vì môi trường được duy trì liên tục, không giống như các runner được lưu trữ trên GitHub tạo ra các máy ảo mới mỗi lần. Nhưng hãy nhớ rằng, bạn cũng chịu trách nhiệm về bảo mật, bảo trì, thời gian hoạt động và khả năng mở rộng của runner, tất cả đều đòi hỏi thiết lập và quản lý liên tục nhiều hơn một chút.

Bây giờ, chúng ta sẽ xem cách thiết lập trình chạy tự lưu trữ.

1. Hãy truy cập vào kho lưu trữ GitHub của bạn.
2. Nhấp vào tab Cài đặt.
3. Trong thanh bên trái, chọn Hành động → Người chạy



Nhấp vào nút New self-hosted runner



Chọn hình người chạy, Sao chép và dán các lệnh thiết lập do GitHub cung cấp vào cửa sổ dòng lệnh trên máy chủ của bạn để đăng ký runner với kho lưu trữ hoặc tổ chức của bạn.

*- Bước 6*: Kiểm tra quy trình CI/CD của bạn

Sau khi mọi công đoạn chuẩn bị đã hoàn tất, bạn có thể bắt đầu kiểm tra quy trình tự động hóa bằng cách thực hiện một thay đổi nhỏ trong dự án Node.js và đẩy (push) mã nguồn lên nhánh chính trên GitHub. Ngay sau đó, hãy truy cập vào tab Actions trong kho lưu trữ để trực tiếp theo dõi và quan sát quá trình thực thi của workflow. Nếu các bước thiết lập đều chính xác, GitHub Actions sẽ tự động thiết lập kết nối SSH tới máy chủ VPS của bạn, từ đó triển khai phiên bản mã nguồn mới nhất một cách nhanh chóng và hiệu quả mà không cần can thiệp thủ công.

## 4. Static and Dynamic Testing

### 4.1. Static Testing

#### 4.1.1. Khái niệm

Static Testing (kiểm thử tĩnh) là phương pháp kiểm thử không thực thi phần mềm, mà tập trung vào việc xem xét, phân tích và đánh giá các artefact trong quá trình phát triển như tài liệu yêu cầu, thiết kế, test plan, test case và mã nguồn.

Mục tiêu chính của Static Testing là phát hiện lỗi sớm ngay từ giai đoạn chuẩn bị, trước khi hệ thống được chạy thực tế, từ đó giảm chi phí và rủi ro trong các giai đoạn kiểm thử động sau này.

#### 4.1.2. Vai trò của Static Testing trong đồ án

Trong phạm vi đồ án, Static Testing được sử dụng chủ yếu để:

* Xác minh tính đầy đủ, rõ ràng và nhất quán của tài liệu kiểm thử
* Đảm bảo các test case được thiết kế phù hợp với yêu cầu
* Phát hiện sớm các lỗi logic, thiếu sót hoặc trùng lặp trong quá trình thiết kế test

Static Testing giúp đảm bảo rằng khi bước sang giai đoạn Dynamic Testing, các test case và quy trình kiểm thử đã đạt chất lượng cần thiết.

#### 4.1.3. Các hình thức Static Testing đã áp dụng

Trong quá trình xây dựng Test Plan và Test Design, việc kiểm tra được thực hiện bằng cách đối chiếu nội dung với yêu cầu hệ thống nhằm đảm bảo:

* Các loại kiểm thử (Functional, Interface, Database, Performance) đều được đề cập
* Phạm vi kiểm thử không bị thiếu hoặc trùng lặp
* Cấu trúc Test Plan tuân theo đúng template chuẩn

Ví dụ:

Tại Chương III, phần *2.2.1.1. Account Registration – Sign Up (Đăng ký tài khoản),* trong chương này đã mô tả rõ những việc mà phần *Account Registration* cần làm để Kiểm thử Usability. Ảnh dưới đây là mô tả của Test Design về chức năng trên

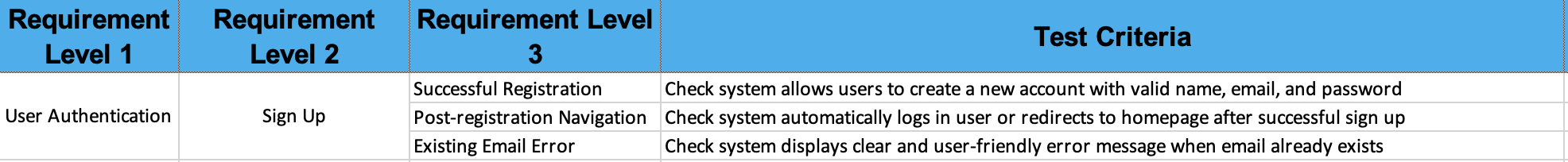


Figure 16. Ví dụ về Usability Testing - Test Design

Ảnh đã mô tả đủ các chức năng cần Test đầy đủ tại file Test Design cho việc thiết về *Sign Up* của Usability Testing. Được tách ra các phần rõ rệt (Require 1 -> 2 -> 3) để dễ dàng đọc hiểu cần làm gì trong đó. Test Criteria cũng được viết chi tiết để việc sinh ra Test Case dễ dàng hơn mà không bị thiếu sót.

#### 4.1.4. Checklist Review

Static Testing được thực hiện thông qua checklist để đảm bảo các thành phần quan trọng không bị bỏ sót.

Ví dụ:

……..

#### 4.1.4. Review Test Case (Excel – Manual Test)

Các test case được thiết kế trên Excel cũng được kiểm tra tĩnh trước khi thực thi nhằm đảm bảo:

* Test Step rõ ràng, logic
* Expected Result khớp với requirement
* Test Data được mô tả đầy đủ
* Không trùng lặp test case

Ví dụ:

A screenshot of a blue and white text

AI-generated content may be incorrect.

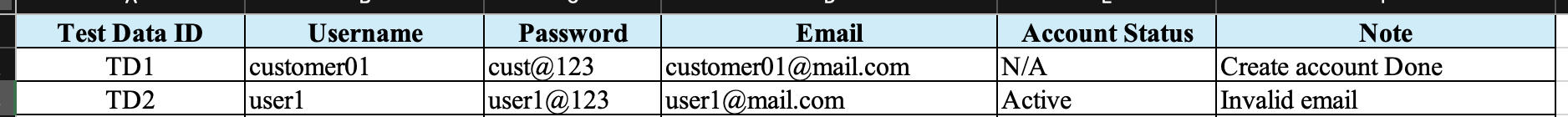


Figure 17. Ví dụ về Usability Testing - Test Cases – Test Data

Ảnh trên là Test Case được sinh ra từ Test Design ở trên, cho thấy Test Cases này đã phủ để các chức năng cần kiểm thử trong Test Design. Ở mỗi Test Case đều có Test Data, nơi lưu trữ các thông tin để có thể kiểm thử. Và thật may, các Test Case này đều Pass hết

#### 4.1.5. Kết luận

Static Testing đóng vai trò quan trọng trong đồ án thông qua việc rà soát và phân tích các tài liệu kiểm thử như Test Plan, Test Design và Test Case. Việc áp dụng các kỹ thuật Static Testing giúp đảm bảo nền tảng kiểm thử vững chắc trước khi tiến hành kiểm thử động, góp phần nâng cao hiệu quả và chất lượng của toàn bộ quy trình kiểm thử phần mềm.

### 4.2. Dynamic Testing

#### 4.2.1. Khái niệm

Dynamic Testing (kiểm thử động) là phương pháp kiểm thử trong đó phần mềm được thực thi thực tế nhằm đánh giá hành vi của hệ thống thông qua việc so sánh kết quả thực tế với kết quả mong đợi được mô tả trong test case. Khác với kiểm thử tĩnh, Dynamic Testing chỉ được thực hiện khi hệ thống đã có khả năng chạy và tương tác.

Trong đồ án này, Dynamic Testing được áp dụng để xác nhận rằng hệ thống Fast Food Delivery hoạt động đúng theo các yêu cầu chức năng và phi chức năng đã đề ra.

#### 4.2.2. White - box Testing

White - box testing được áp dụng ở mức kiểm thử đơn vị nhằm kiểm tra các nhánh xử lý và logic bên trong của hệ thống backend. Các test case được thiết kế dựa trên luồng thực thi của mã nguồn, bao gồm các trường hợp thành công, thất bại và xử lý ngoại lệ. Các kịch bản kiểm thử vẫn đảm bảo bao phủ các nhánh logic quan trọng của hệ thống.



Figure 18. Ví dụ về White - box Testing trong Unit Test

Nhìn vào ảnh ta có thể thấy, đây là Unit Test được kiểm thử theo White – Box Testing. Nó được kiểm thử theo *Logic Code : stripe.create throws Error; orderModel.save throws Error*. Đây là việc kiểm thử trực tiếp dựa trên luồng xử lý trong code backend, không phải góc nhìn user.

Có *Branch / Catch / Path coverage.* Cột *Coverage Type* ghi rõ: *Statement, Branch (True / False), Branch (Catch block), Path.* Đây là thuật ngữ được dùng trong White-box testing

Có mock hành vi nội bộ: *Mock: stripe.create throws Error; Mock: orderModel.save throws Error.*

#### 4.2.3. Black - box Testing

Dynamic Black-box Testing là phương pháp kiểm thử động trong đó người kiểm thử không quan tâm đến cấu trúc bên trong của hệ thống, mà chỉ tập trung vào đầu vào và đầu ra dựa trên các test case đã thiết kế. Hệ thống được xem như một “hộp đen”, Tester không biết bên trong xử lý thế nào và Chỉ quan tâm hệ thống làm đúng hay sai

Trong đồ án, Dynamic Black-box Testing được sử dụng chủ yếu cho các chức năng hướng người dùng, bao gồm:

* Kiểm thử chức năng (Funcional Testing)
* Kiểm thử tương thích (Compatibility Testing)
* Kiểm thử khả dụng (Usability Testing)

Việc kiểm thử được thực hiện bằng cách thao tác trực tiếp trên giao diện web và so sánh kết quả hiển thị với expected result trong test case.

Ví dụ:

A screenshot of a blue and white text

AI-generated content may be incorrect.

Tại ảnh này, việc kiểm thử dựa trực tiếp vào việc chúng ta lấy dữ liệu ở TestData, mang lên Website để kiểm thử, so sánh với kết quả dự đoán. Nếu đúng thì kết quả dự đoán cho ra True, ngược lại là Fail. Việc kiểm thử này không liên qua tới code, database của hệ thống.

#### 4.2.3. Gray - box Testing

Gray-box Testing là phương pháp kiểm thử trong đó người kiểm thử có hiểu biết MỘT PHẦN về hệ thống bên trong (kiến trúc, database, API, luồng xử lý), nhưng KHÔNG test trực tiếp từng dòng code như White-box. Gray-box = Black-box + kiến thức backend



Figure 19. Ví dụ về Gray - box Testing trong Database Test Cases

Ảnh 19 đã là 1 ví dụ về việc áp dụng Gray – box Testing trong việc thực thi Test Case. Việc kiểm thử dữ liệu của đơn đặt hàng trong DB sau khi việc đặt được hoàn thành có chính xác với dữ liệu vừa được dùng để đặt hàng hay không. Hay như việc kiểm thử xem CustomerID có được lưu trữ chính xác không (ID người đặt hàng phải khớp với đơn hàng vừa đặt)

## 5. Kiểm thử tự động (Auto Testing)

### 5.1. Khái niệm

Kiểm thử tự động (Automation Testing) là phương pháp kiểm thử trong đó các test case được thực hiện tự động bằng công cụ hoặc chương trình thay vì thao tác thủ công. Tester viết kịch bản kiểm thử (test script) để mô phỏng hành vi của người dùng hoặc kiểm tra phản hồi của hệ thống, sau đó công cụ sẽ tự động thực thi các bước kiểm thử và ghi nhận kết quả.

Khác với kiểm thử thủ công, kiểm thử tự động giúp giảm công sức lặp lại, tăng tốc độ kiểm thử và đảm bảo tính nhất quán của kết quả.

Kiểm thử tự động không áp dụng cho mọi trường hợp, mà phù hợp nhất với các tình huống sau:

### 5.2. Các chức năng kiểm thử tự động

#### 5.2.1. Database Testing

Trong đồ án, kiểm thử tự động được áp dụng thông qua việc xây dựng các kịch bản kiểm thử cơ sở dữ liệu bằng script. Các script này được sử dụng để tự động kiểm tra việc lưu trữ, cập nhật và truy xuất dữ liệu trong MongoDB, đồng thời so sánh kết quả thực tế với kết quả mong đợi. Hình thức kiểm thử này giúp tăng độ chính xác và giảm thời gian kiểm thử so với kiểm thử thủ công.

Dưới đây là 1 vài script để tự động kiểm tra dữ liệu trong database. Script được xây dựng dựa trên test case đã thiết kế trước.

def test\_tc\_dc\_01\_update\_db\_reflects\_on\_ui(db\_collection, driver):  
 *"""TC 01: Cập nhật DB -> Kiểm tra hiển thị trên UI"""* updated\_data = {  
 "name": "Gà nướng",  
 "description": "Updated via AutoTest",  
 "price": 45  
 }  
  
 # 1. Update DB  
 db\_collection.update\_one(  
 {"\_id": CONFIG["TEST\_ID\_UPDATE"]},  
 {"$set": updated\_data}  
 )  
  
 # 2. Verify UI  
 driver.get(CONFIG["UI\_URL"])  
 try:  
 # Tìm item theo tên  
 item\_name = driver.find\_element(  
 By.XPATH,  
 f"//\*[contains(@class, 'food-item-name-rating')]//p[contains(text(), '{updated\_data['name']}')]"  
 )  
  
 # Tìm giá tiền tương ứng  
 card = item\_name.find\_element(By.XPATH, "./ancestor::\*[contains(@class, 'food-item')]")  
 price\_text = card.find\_element(By.CLASS\_NAME, "food-item-price").text  
  
 assert str(updated\_data["price"]) in price\_text  
 print("TC 01 Passed")  
  
 except Exception as e:  
 pytest.fail(f"TC 01 Failed: {e}")

Script này mô tả việc Xác nhận thông tin sau khi update sản phẩm trong DB thì bên Website phải hiển thị đúng thông tin đã được cập nhật.

Để có thể kiểm thử tự động, Script được kết hợp với Selenium. Script được viết trên nền code python kết hợp với Selenium package.

* Trước tiên dùng Script để truy cập vào DB, cập nhật thông tin sản phẩm.
* Dùng Selenium để truy trích xuất xem thông tin đã hiển thị đúng hay chưa.

#### 5.2.1. Performance Testing

Khác với Database Testing, Performance Testing dùng công cụ để kiểm thử tự động. Công cụ được sử dụng là JMeter.

Apache JMeter là một công cụ mã nguồn mở được phát triển bởi Apache Software Foundation, dùng để kiểm thử hiệu năng (Performance Testing) và kiểm thử tải (Load/Stress Testing) của các ứng dụng, đặc biệt là ứng dụng web và dịch vụ API.

JMeter cho phép mô phỏng nhiều người dùng ảo (virtual users) gửi request đồng thời đến hệ thống nhằm đo lường các chỉ số như thời gian phản hồi, thông lượng và tỷ lệ lỗi.

Trong kiểm thử phần mềm, JMeter được sử dụng để:

* Đánh giá khả năng chịu tải của hệ thống
* Phát hiện điểm nghẽn hiệu năng
* Kiểm tra độ ổn định khi hệ thống hoạt động trong thời gian dài
* Đảm bảo hệ thống vẫn phản hồi đúng dưới tải cao

JMeter cho phép cấu hình các kịch bản kiểm thử linh hoạt thông qua Thread Group, Sampler, Timer, Listener và Assertion.

Ở đồ án này, JMeter được sử dụng thực hiện Load Testing với Stress Testing trong kiểm thử Performance. Giúp đánh giá khả năng ổn định của hệ thống trong điều kiện tải cao.

Dưới đây là cấu hình của 1 Test Case trong JMeter

*Test case này mô tả việc 500 người dùng thêm sản phẩm vào giỏ hàng.*

- Thread Group: được cấu hình với 500 người dùng ảo, ramp-up trong 600 giây nhằm mô phỏng lưu lượng truy cập tăng dần. Kịch bản kiểm thử được chạy liên tục trong 900 giây để đánh giá khả năng chịu tải và độ ổn định của hệ thống. Tùy chọn giữ nguyên người dùng qua các vòng lặp được bật nhằm mô phỏng hành vi người dùng thực tế.

A screenshot of a computer program

AI-generated content may be incorrect.

- HTTP Request trong JMeter được cấu hình sử dụng giao thức HTTPS để gửi yêu cầu trực tiếp đến backend đã được triển khai trên nền tảng Render. Endpoint /api/cart/add được sử dụng để mô phỏng hành vi người dùng thêm sản phẩm vào giỏ hàng. Các tham số request được gửi dưới dạng JSON và kèm theo header xác thực nhằm đảm bảo phản ánh đúng môi trường vận hành thực tế của hệ thống.

A screenshot of a computer

AI-generated content may be incorrect.

- Header Manager: được sử dụng để thiết lập header **Content-Type: application/json** cho các HTTP request trong JMeter, đảm bảo dữ liệu gửi lên server được định dạng đúng theo chuẩn JSON và backend có thể xử lý chính xác nội dung request.

A screen shot of a computer

AI-generated content may be incorrect.

- Summary Report: hiển thị kết quả tổng hợp của quá trình kiểm thử hiệu năng, bao gồm số lượng request đã gửi, thời gian phản hồi trung bình, giá trị nhỏ nhất và lớn nhất, độ lệch chuẩn, tỷ lệ lỗi và thông lượng của hệ thống. Kết quả cho thấy hệ thống xử lý thành công 500 request với thời gian phản hồi trung bình khoảng 297 ms và tỷ lệ lỗi bằng 0%, đáp ứng tốt yêu cầu kiểm thử tải đồng thời.

A screenshot of a computer

AI-generated content may be incorrect.

# V. KẾT QUẢ KIỂM THỬ

## 1. Test Defect

Test Defect (lỗi phần mềm) là sự sai lệch giữa kết quả thực tế (Actual Result) của hệ thống so với kết quả mong đợi (Expected Result) được mô tả trong tài liệu yêu cầu (Requirement), Use Case hoặc Test Case.

Một defect được ghi nhận khi:

* Hệ thống không thực hiện đúng chức năng đã được yêu cầu, hoặc
* Hệ thống xử lý sai dữ liệu đầu vào, hoặc
* Hệ thống không xử lý đầy đủ các tình huống bất thường (exception, timeout, lỗi kết nối…), hoặc
* Hệ thống vi phạm các yêu cầu phi chức năng như bảo mật, hiệu năng, độ ổn định.

Dưới đây là Bảng mô tả các Test Defect của Đồ án:

A screenshot of a computer

AI-generated content may be incorrect.

Figure 20. Test Defect

Ảnh trên đã mô tả 4 lỗi của hệ thống trong quá trình kiểm thử, đã làm rõ mức độ ưu tiên và độ nghiêm trọng cần phải xử lý khi hệ thống gặp lỗi.

Mô tả 1 Test Defect:

Trong quá trình kiểm thử chức năng **đăng ký người dùng**, nhóm kiểm thử phát hiện một lỗi nghiêm trọng liên quan đến **xác thực dữ liệu đầu vào ở phía backend**.

Cụ thể, khi thực hiện gửi trực tiếp yêu cầu API đến endpoint /api/user/register với định dạng email không hợp lệ (ví dụ: không chứa ký tự “@” hoặc tên miền), hệ thống vẫn chấp nhận dữ liệu và tạo tài khoản người dùng thành công.

Kết quả thực tế cho thấy backend không thực hiện đầy đủ việc kiểm tra định dạng email, mà chỉ phụ thuộc vào validation phía frontend. Điều này cho phép người dùng hoặc kẻ tấn công bỏ qua giao diện người dùng và gửi yêu cầu trực tiếp đến server để tạo dữ liệu không hợp lệ trong hệ thống.

Trong khi đó, kết quả mong đợi là hệ thống phải từ chối yêu cầu đăng ký, trả về thông báo lỗi rõ ràng (ví dụ: *“Please enter a valid email”*) và không tạo tài khoản mới.

Defect này được đánh giá:

* Severity: Medium – vì ảnh hưởng đến tính toàn vẹn dữ liệu người dùng
* Priority: Medium – cần được sửa trong giai đoạn hoàn thiện chức năng xác thực

Lỗi cho thấy hệ thống chưa đảm bảo nguyên tắc validation hai lớp (frontend & backend), tiềm ẩn rủi ro về dữ liệu và bảo mật trong môi trường thực tế.

*Kết luận:*

Test Defect đóng vai trò quan trọng trong việc đánh giá chất lượng phần mềm. Thông qua việc ghi nhận và phân tích defect, nhóm phát triển có thể phát hiện sớm các vấn đề tiềm ẩn, đặc biệt là các lỗi không dễ nhận thấy thông qua kiểm thử giao diện.

## 2. Test Summary

Trong giai đoạn kiểm thử, hệ thống được đánh giá thông qua nhiều loại kiểm thử khác nhau bao gồm: Functional Testing, Interface Testing, Database Testing, Compatibility Testing, Usability Testing và Performance Testing.  
Mục tiêu của giai đoạn này là xác minh hệ thống hoạt động đúng theo yêu cầu, ổn định và đáp ứng các tiêu chí phi chức năng đã đề ra.

Dưới đây là bảng tổng hợp các Test đã làm và kết quả.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Type (Sheet)** | **Passed** | **Failed** | **N/A** | **Total** |
| Interface | 0 | 4 | 1 | 35 |
| Functional | 0 | 0 | 1 | 30 |
| Compatibility | 12 | 0 | 0 | 12 |
| Database | 8 | 0 | 0 | 8 |
| Usability | 13 | 0 | 0 | 13 |
| Performance | 6 | 0 | 0 | 6 |
| Total | 39 | 4 | 2 | 104 |

Biểu đồ dưới đây thể hiện tỷ lệ kết quả kiểm thử của toàn bộ hệ thống. Phần lớn các test case đạt trạng thái Passed, cho thấy hệ thống hoạt động ổn định. Một số test case Failed và N/A chủ yếu xuất hiện trong kiểm thử Interface do phụ thuộc vào môi trường và tích hợp hệ thống bên ngoài.

Figure 21. Biểu đồ tổng hợp các Test

# VI. Kết luận & hướng phát triển

## 6.1. Kết luận

Trong đồ án này, nhóm đã nghiên cứu, thiết kế và triển khai hệ thống kiểm thử phần mềm cho ứng dụng web theo quy trình kiểm thử chuẩn, bao gồm các giai đoạn: phân tích yêu cầu, lập kế hoạch kiểm thử, thiết kế test case, chuẩn bị môi trường, thực thi kiểm thử, báo cáo kết quả và quản lý lỗi.

Hệ thống kiểm thử được xây dựng với nhiều loại kiểm thử khác nhau như Functionality Testing, Interface Testing, Database Testing, Performance Testing và Compatibility Testing, giúp đánh giá toàn diện chất lượng của ứng dụng. Các test case được thiết kế rõ ràng, có test data và tiêu chí đánh giá cụ thể, đảm bảo khả năng tái sử dụng và dễ dàng theo dõi kết quả kiểm thử.

Kết quả thực nghiệm cho thấy hệ thống hoạt động ổn định trong các kịch bản kiểm thử đã xây dựng. Phần lớn các chức năng chính của hệ thống đáp ứng đúng yêu cầu ban đầu, các lỗi phát sinh đã được ghi nhận và quản lý thông qua công cụ theo dõi lỗi, góp phần nâng cao độ tin cậy và chất lượng của phần mềm.

Thông qua đồ án, nhóm đã củng cố kiến thức về quy trình kiểm thử phần mềm thực tế, nâng cao kỹ năng phân tích yêu cầu, thiết kế test case và làm việc với các công cụ hỗ trợ kiểm thử. Đây là nền tảng quan trọng cho việc áp dụng kiểm thử phần mềm trong các dự án thực tế sau này.

## 6.2. Hướng phát triển

Mặc dù đã đạt được các mục tiêu đề ra, hệ thống vẫn còn một số hạn chế và có thể được mở rộng trong tương lai theo các hướng sau:

Tự động hóa kiểm thử: Mở rộng việc áp dụng các công cụ kiểm thử tự động như Selenium, JMeter để giảm thời gian và công sức cho kiểm thử thủ công, đặc biệt với các test case lặp lại.

Mở rộng phạm vi kiểm thử: Bổ sung thêm các loại kiểm thử nâng cao như Security Testing, Usability Testing và Accessibility Testing nhằm đánh giá toàn diện hơn chất lượng phần mềm.

Quản lý test và defect chuyên nghiệp hơn: Áp dụng các công cụ quản lý kiểm thử và lỗi chuyên dụng (Test Management Tool) để nâng cao hiệu quả theo dõi và báo cáo.

Mở rộng môi trường kiểm thử: Kiểm thử trên nhiều nền tảng, trình duyệt và thiết bị khác nhau nhằm đảm bảo tính tương thích và ổn định của hệ thống.

Những hướng phát triển trên sẽ giúp hệ thống kiểm thử ngày càng hoàn thiện, đáp ứng tốt hơn yêu cầu thực tế và tiệm cận với các quy trình kiểm thử chuyên nghiệp trong doanh nghiệp.