 

Graduation Project Proposal

[ Project Name:Talabat]

Submitted By:

Shahd Ahmed Arafa

Rahma Nasr

Mustafa Amr Ibrahim

Youssef Mohamed Zanaty

Ahmed Khaled

**Acknowledgement**

I would like to express my deepest gratitude to the Ministry of Communications and Information Technology (MCIT) and the Digital Egypt Builders Initiative (DEBI) for providing me with the valuable opportunity to enhance my skills and knowledge through their esteemed program. This project would not have been possible without their continuous support and guidance.

I am also immensely grateful to my mentor, Mr. Mohamed Mahmoud, for his dedicated supervision, valuable insights, and constructive feedback throughout the course of this project. His expertise and encouragement were instrumental in achieving the desired results and enhancing the quality of this work.

Furthermore, I would like to acknowledge everyone who contributed directly or indirectly to the successful completion of this project. Their assistance and encouragement are sincerely appreciated.

Abstract

This project aims to evaluate and enhance the performance of the Talabat application, a leading food delivery platform in the Middle East. As online food delivery services continue to grow in popularity, it is essential to assess how effectively Talabat meets user needs in terms of convenience, user satisfaction, and overall functionality. The study involves analyzing various aspects of the application, including its strengths, weaknesses, and areas requiring improvement.

Through a comprehensive review of existing studies and critical analysis, this project identifies several challenges faced by Talabat, such as occasional delivery delays, high fees, and limited customer support. To address these issues, proposed improvements include implementing machine learning algorithms to predict peak demand periods, enhancing customer service through AI-driven solutions, and providing better incentives to attract more users.

The findings of this project will serve as a foundation for future research and practical implementations aimed at improving the efficiency and user experience of food delivery applications like Talabat.

Table of Contents

[1. Project Planning & Management 5](#_Toc193324955)

[Project Proposal 5](#_Toc193324956)

[Exceptional Feature: Order Splitting Option 5](#_Toc193324957)

[Project Plan 5](#_Toc193324958)

[Task Assignment & Roles 5](#_Toc193324959)

[Risk Assessment & Mitigation Plan 5](#_Toc193324960)

[KPIs (Key Performance Indicators) 6](#_Toc193324961)

[The success of the project will be measured using the following KPIs: 1. Response Time: Ensuring efficient loading times for the application. 2. System Uptime: Maintaining a high availability rate (e.g., 99.9%). 3. User Adoption Rate: Tracking the number of users actively using the application. 4. Customer Satisfaction: Gathering feedback to improve user experience continuously continuously Literature Review - Talabat Application 6](#_Toc193324962)

[2. Literature Review - Talabat Application 6](#_Toc193324963)

[Feedback & Evaluation 6](#_Toc193324964)

[Suggested Improvements 6](#_Toc193324965)

[Final Grading Criteria 6](#_Toc193324966)

3. Requirements Gathering………………………………………………………………………………………………………………………6

• Stakeholder Analysis – Identifying key stakeholders and their needs……………………………………………………6

• User Stories & Use Cases – Scenarios illustrating how users interact with the system………………………….6

• Functional Requirements – List of features and functionalities…………………………………………………………….6

• Non-functional Requirements – Performance, security, usability, and reliability criteria……………………..6

4. System Analysis & Design……………………………………………………………………………………………………………………7

1.Problem Statement & Objectives – Define the problem being solved and project goals………………………7

• Use Case Diagram & Descriptions – Identify system actors and interactions……………………………………….7

• Functional & Non-Functional Requirements – Clearly state system capabilities and constraints…………7

• Software Architecture – High-level design outlining system components, interactions, and architecture style (e.g., MVC, Microservices)………………………………………………………………………………………………………………7

2. Database Design & Data Modeling………………………………………………………………………………………………………7

• ER Diagram (Entity-Relationship Diagram) – A well-defined ERD showcasing database structure and relationships…………………………………………………………………………………………………………………………………………8

• Logical & Physical Schema – Tables, attributes, keys, and normalization considerations……………………8

3. Data Flow & System Behavior…………………………………………………………………………………………………………..8

• DFD (Data Flow Diagram) – Context-level and detailed levels showing how data moves through the system…………………………………………………………………………………………………………………………………………………8

• Sequence Diagrams – Process flow representation of key interactions between components…………8

• Activity Diagram – Visualizing the workflow of processes or user actions within the system……………8

• State Diagram – Represents different states of an object and how it transitions between them……..8

• Class Diagram – Defines the structure of the system by showing classes, attributes, methods, and relationships………………………………………………………………………………………………………………………………………..8

4. UI/UX Design & Prototyping ……………………………………………………………………………………………………………9

• Wireframes & Mockups – Screens and visual representations of the user interface…………………………9

• UI/UX Guidelines – Design principles, color schemes, typography, and accessibility considerations..9

5. System Deployment & Integration………………………………………………………………………………………………….9

• Technology Stack – Backend, frontend, and database technologies………………………………………………..9

• Deployment Diagram – Describes how software components are distributed across hardware……..9

• Component Diagram – Shows high-level system components and their dependencies…………………..9

6. Additional Deliverables (if applicable)……………………………………………………………………………………………9

• API Documentation – If the system includes APIs, provide documentation for endpoints and usage.9

• Testing & Validation – Unit tests, integration tests, and user acceptance testing plan…………………..9

• Deployment Strategy – Hosting environment, deployment pipelines, and scaling considerations….9

# 1. Project Planning & Management

## Project Proposal

This project aims to develop a food delivery application similar to Talabat, providing users with an efficient and user-friendly platform to order food and groceries from various restaurants and stores. The main objectives are to enhance user experience, improve delivery efficiency, and ensure system reliability. The scope of the project includes designing a web application, implementing backend APIs, developing a user-friendly interface, and integrating payment systems.

### Exceptional Feature: Order Splitting Option

This application offers a unique feature called the 'Order Splitting Option'. This feature allows users to split a single order across multiple restaurants or stores at the same time, providing greater convenience and flexibility.  
  
How it works:  
- Users can select various items from different restaurants and add them to a single cart.  
- Upon checkout, the order is automatically split and sent to the respective restaurants simultaneously.  
- The application displays each order’s status separately for easy tracking.  
  
Benefits:  
- Saves time by allowing mixed orders from multiple restaurants.  
- Provides flexibility for users with diverse preferences.  
- Encourages users to rely more on the application for their various needs.

## Project Plan

The project plan involves the following components:  
1. Timeline: A Gantt chart will be used to visualize the project's progress, including each phase from planning to deployment.  
2. Milestones: Key deliverables include project proposal submission, prototype development, testing, and final deployment.  
3. Deliverables: Functional application, documentation, user guides, and deployment files.  
4. Resource Allocation: Distribution of tasks and resources among team members to ensure efficient execution.

## Task Assignment & Roles

The project team consists of developers, designers, and testers, each responsible for specific tasks:  
1. Developers: Writing clean, maintainable code and implementing backend APIs.  
2. Designers: Creating an intuitive and user-friendly interface.  
3. Testers: Ensuring functionality through rigorous testing and providing feedback for improvements.

## Risk Assessment & Mitigation Plan

Potential risks and their mitigation strategies include:  
1. Technical Issues: Utilizing version control systems (e.g., Git) to avoid data loss and ensure code integrity.  
2. Delays in Development: Implementing agile methodologies to adapt to changes and maintain progress.  
3. Security Vulnerabilities: Incorporating secure coding practices and conducting regular vulnerability assessments.

## KPIs (Key Performance Indicators)

## The success of the project will be measured using the following KPIs: 1. Response Time: Ensuring efficient loading times for the application. 2. System Uptime: Maintaining a high availability rate (e.g., 99.9%). 3. User Adoption Rate: Tracking the number of users actively using the application. 4. Customer Satisfaction: Gathering feedback to improve user experience continuously continuously Literature Review - Talabat Application

# Literature Review - Talabat Application

## Feedback & Evaluation

The project demonstrates a comprehensive understanding of how food delivery applications function. The literature review is well-organized, providing clear insights into the strengths and weaknesses of Talabat. However, further research on improving delivery efficiency and customer service is recommended.

## Suggested Improvements

1. Implementing machine learning algorithms to predict peak demand periods and optimize delivery routes.  
2. Enhancing customer support systems through chatbots and AI-driven solutions.  
3. Providing better incentives and discounts to attract more users.

## Final Grading Criteria

The evaluation of the project will be based on the following criteria:  
1. Documentation (30%) - Clarity, organization, and quality of the literature review.  
2. Implementation (40%) - Practical application of concepts and technologies.  
3. Testing (20%) - Effectiveness of testing and validation of the application.  
4. Presentation (10%) - Professionalism and quality of the presentation.

1. Requirements Gathering

**1. Introduction**

**1.1 Purpose**

This document outlines the functional and non-functional requirements for an online food delivery platform that connects customers with restaurants and delivery partners.

**1.2 Scope**

The platform allows customers to browse restaurants, place orders, and track deliveries in real time. It provides a web and mobile-friendly interface for users, restaurant partners, and delivery personnel.

**1.3 Stakeholders**

* **Customers** – Users who browse menus and place orders.
* **Restaurants** – Partners who list their menus and fulfill orders.
* **Delivery Riders** – Individuals responsible for delivering orders.
* **Admin Team** – Manages users, orders, and platform settings.

**2. Functional Requirements**

**2.1 User Registration & Authentication**

* Customers, restaurants, and delivery partners can register via email, phone number, or social media.
* Secure login using OTP or password authentication.
* Role-based access control (Customers, Restaurants, Delivery Riders, Admin).

**2.2 Customer Features**

* Browse restaurants and filter by cuisine, location, ratings, and offers.
* View restaurant menus with images, prices, and item descriptions.
* Add items to the cart, customize orders, and apply promo codes.
* Choose a delivery address using Google Maps integration.
* Track the order status (Pending, Preparing, Out for Delivery, Delivered).
* Save favorite restaurants and order history.
* Secure payment options (Credit/Debit Cards, Wallets, Cash on Delivery).
* Rate and review restaurants and delivery experience.

**2.3 Restaurant Partner Features**

* Create and manage restaurant profiles.
* Add and update menus, pricing, and availability.
* Receive and accept/reject orders.
* Manage order preparation time.
* View sales reports and earnings.

**2.4 Delivery Partner Features**

* Accept or decline delivery requests.
* Navigate using Google Maps for optimal routes.
* Update order status (Picked Up, Out for Delivery, Delivered).
* View earnings and delivery history.

**2.5 Admin Panel**

* Manage users (Customers, Restaurants, Delivery Partners).
* Approve restaurant registrations.
* Monitor order statuses and issue refunds.
* Generate analytics and sales reports.
* Handle complaints and support tickets.

**3. Non-Functional Requirements**

**3.1 Performance & Scalability**

* The system should handle high traffic loads during peak hours.
* Optimize database queries for fast response times.

**3.2 Security**

* Implement data encryption for user credentials and payments.
* Protect against SQL Injection and Cross-Site Scripting (XSS).
* Enforce role-based access controls.

**3.3 Usability & Accessibility**

* Ensure a user-friendly and responsive design.
* Support accessibility features for differently-abled users.

**3.4 Availability & Reliability**

* Maintain 99.9% uptime with server redundancy.
* Implement auto-retry mechanisms for failed transactions.

**4. Technical Requirements**

**4.1 Technology Stack**

* **Frontend:** React.js / Vue.js (Web), Flutter / React Native (Mobile)
* **Backend:** .NET Core / ASP.NET Web API
* **Database: Microsoft SQL Server**

**4.2 Architecture**

* **Microservices-based architecture** for scalability.
* **RESTful API** for communication between frontend and backend.
* **Entity Framework Core** for database interaction.
* **JWT-based authentication** for secure user management.
* **SignalR for real-time updates** (order tracking and notifications).

**5. Conclusion**

This document defines the core requirements for building an online food delivery platform using **.NET**. Future iterations may include AI-based recommendations, loyalty programs, and enhanced delivery tracking features.

**4.system Analysis & Design**

1. **Problem Statement:**

Online food delivery faces issues like delays, high fees, and poor customer support. Restaurants struggle with order management, and delivery efficiency needs improvement.

**Objectives:**

* Optimize order processing and delivery.
* Use AI for demand prediction and customer support.
* Improve user experience and payment security.

### Use Case Descriptions:

1. **Place Order:** Customer selects food, system processes it.
2. **Process Order:** Restaurant prepares and updates status.
3. **Assign Delivery:** System assigns nearest driver.
4. **Deliver Order:** Driver picks up and delivers.
5. **Process Payment:** Customer pays via multiple methods.
6. **Customer Support:** AI chatbot assists with issues.

## 3. Functional & Non-Functional Requirements:

**Functional:**

* Browse menus, place orders.
* Restaurants manage orders.
* AI chatbot for support.
* Secure payments.

**Non-Functional:**

* Scalable system for peak hours.
* Fast order processing (<3s).
* Secure transactions.
* Responsive UI.

## 4. Software Architecture:

**Architecture: MVC (Model-View-Controller)**

* **Frontend:** Mobile/Web UI.
* **Backend:** Order processing, AI, authentication.
* **Database:** Stores users, orders, restaurants.
* **APIs:** Payments, Google Maps.

**System Flow:**

1. Customers place orders.
2. Backend notifies restaurants.
3. Orders assigned to drivers.
4. Payments processed securely.
5. AI chatbot handles support.

2. Database Design & Data Modeling:

• ER Diagram (Entity-Relationship Diagram) – A well-defined ERD showcasing database structure and relationships.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

• Logical & Physical Schema – Tables, attributes, keys, and normalization considerations:

Logical Schema for the Database:

**User Table** (id, full\_name, email, password, phone\_number, address, user\_type, created\_at) – id is the Primary Key.

**Normalization Considerations:**

* Email and phone number must be unique for each user.
* Address can be split into city, street, and zip\_code to avoid redundancy.

**Restaurant Table** (restaurant\_id, restaurant\_name, description, phone\_number , address, rating, owner\_id, created\_at) – restaurant\_id is the Primary Key, owner\_id is a Foreign Key referencing User(id).

**Normalization Considerations:**

* Avoid storing duplicate data like phone\_number; it can be in a separate table.

**Menu\_Item Table** (item\_id, name, description, price, category, restaurant\_id, available) – item\_id is the Primary Key, restaurant\_id is a Foreign Key referencing Restaurant(restaurant\_id).

**Normalization Considerations:**

* category can be moved to a separate table if there are multiple recurring categories.

**Order Table** (order\_id, user\_id, restaurant\_id, total\_price, status, created\_at) – order\_id is the Primary Key, user\_id is a Foreign Key referencing User(id), restaurant\_id is a Foreign Key referencing Restaurant(restaurant\_id).

**Normalization Considerations:**

* Avoid storing duplicate data like restaurant\_name inside the order table

**Order\_Details Table** (order\_detail\_id, order\_id, item\_id, quantity, subtotal) – order\_detail\_id is the Primary Key, order\_id is a Foreign Key referencing Order(order\_id), item\_id is a Foreign Key referencing Menu\_Item(item\_id).

**Normalization Considerations:**

* subtotal = quantity \* price should be calculated dynamically instead of being stored.

**Payment Table** (payment\_id, order\_id, payment\_method, transaction\_id, payment\_status, created\_at) – payment\_id is the Primary Key, order\_id is a Foreign Key referencing Order(order\_id).

Normalization Considerations:

* payment\_method can be moved to a separate table if multiple payment methods exist.

**Delivery Table** (delivery\_id, order\_id, delivery\_person\_id, status, estimated\_time) delivery\_id is the Primary Key, order\_id is a Foreign Key referencing Order(order\_id), delivery\_person\_id is a Foreign Key referencing User(id).

Data Flow Diagrams (DFD)

**Context- and Detalied Level DFD:**

**A diagram of a software application

AI-generated content may be incorrect.**

**Sequence Diagram: Order Process:**

A diagram of a food delivery

AI-generated content may be incorrect.

**Activity Diagram - Order Workflow:**

**A diagram of a restaurant

AI-generated content may be incorrect.**

**State Diagram – Order Lifecycle:**

**A screen shot of a phone

AI-generated content may be incorrect.**

**Class Diagram – System Structure:**

A diagram of a restaurant

AI-generated content may be incorrect.

**Technology Stack – System Deployment & Integration**

**1. Backend Technologies**

* **Framework:** .NET Core Web API
* **Programming Language:** C#
* **Architecture Pattern:** RESTful API, MVC (Model-View-Controller)
* **Authentication & Authorization:** JWT (JSON Web Token), OAuth 2.0
* **Data Handling:** Entity Framework Core (EF Core)
* **Caching:** Redis or In-Memory Caching
* **Logging & Monitoring:** Serilog, Application Insights
* **API Documentation:** Swagger (OpenAPI)

**2. Frontend Technologies**

* **Framework/Library:** React.js or Angular
* **Markup & Styling:** HTML5, CSS3, Bootstrap, Tailwind CSS
* **State Management:** Redux (if React), NgRx (if Angular)
* **Client-side Routing:** React Router / Angular Router
* **API Communication:** Axios / Fetch API

**3. Database Technologies**

* **Database Management System:** SQL Server
* **ORM (Object-Relational Mapping):** Entity Framework Core
* **Database Design:** Normalized relational schema
* **Stored Procedures & Triggers:** SQL Server T-SQL
* **Replication & Backup:** SQL Server Replication, Automated Backups

**4. Deployment & Hosting**

* **Cloud Provider:** Microsoft Azure / AWS
* **Containerization:** Docker
* **Orchestration:** Kubernetes / Azure Kubernetes Service (AKS)
* **Web Server:** IIS (Internet Information Services)
* **CI/CD Pipeline:** GitHub Actions, Azure DevOps, Jenkins
* **Load Balancing:** Nginx / Azure Load Balancer

**5. Integration & Third-party Services**

* **Payment Gateway:** Stripe, PayPal, or local gateways
* **Push Notifications:** Firebase Cloud Messaging (FCM)
* **Geolocation Services:** Google Maps API
* **Email & SMS Services:** SendGrid, Twilio
* **Social Login:** Google, Facebook OAuth

1. **Implementation (Source Code & Execution)**

## 1. Source Code

### 1.1 Structured & Well-Commented Code

The project's source code follows clean coding principles with proper documentation. Each module, function, and class is well-commented to enhance readability and maintainability.

**1.2 Coding Standards & Naming Conventions**

* Code formatting adheres to industry standards (e.g., consistent indentation, spacing, and structure).
* Meaningful variable and function names are used to improve code clarity.
* Follows .NET and JavaScript best practices for naming conventions.

**1.3 Modular Code & Reusability**

* The application is structured into reusable components and functions.
* Business logic is separated from the presentation layer.
* Common functionalities are encapsulated in helper classes and services.

**1.4 Security & Error Handling**

* Input validation is enforced to prevent security vulnerabilities such as SQL injection and cross-site scripting (XSS).
* Exception handling is implemented to ensure system stability and reliability.
* Authentication and authorization mechanisms are integrated to secure the system.

## 2. Version Control & Collaboration

**2.1 Version Control Repository**

* The project is hosted on [GitHub/GitLab/Bitbucket] with a structured repository.
* Repository link: [Insert Repository Link]

**2.2 Branching Strategy**

* Follows GitFlow strategy:
  + main branch for stable production releases.
  + develop branch for ongoing development.
  + Feature branches (feature/\*) for new functionalities.
  + Bugfix branches (bugfix/\*) for fixing issues.

**2.3 Commit History & Documentation**

* Commit messages are meaningful and follow a consistent format.
* Pull requests include detailed descriptions of changes and related issues.

**2.4 CI/CD Integration (if applicable)**

* Automated build and testing pipelines are configured.
* Deployment process includes staging and production environments.

## 3. Deployment & Execution

**3.1 README File**

The project includes a comprehensive README.md file containing:

* Installation steps
* System requirements (hardware/software dependencies)
* Configuration instructions
* Execution guide (running the project locally or accessing a deployed version)
* API documentation (if applicable)

**3.2 Executable Files & Deployment Link**

* The system is packaged into deployable artifacts.
* Executable versions (e.g., .exe, .jar, .apk) are available if applicable.
* Deployed web application link: [Insert Deployment Link]

# Testing & Quality Assurance

**1. Test Plan & Test Cases**

**1.1 Test Plan**

The testing plan for the Food Ordering System aims to ensure that the application functions correctly, meets user requirements, and maintains high-quality standards.

**Objectives:**

* Verify that all functionalities work as expected.
* Ensure the system is secure, stable, and scalable.
* Identify and resolve any usability or performance issues.

**Testing Scope:**

* Functional Testing
* UI/UX Testing
* Security Testing
* Performance Testing
* Integration Testing

**Tools Used:**

* Manual testing using test cases
* Automated testing using [Selenium, NUnit, or any applicable framework]

|  |
| --- |
| 1.2 Test Cases |
| |  |  |  |  | | --- | --- | --- | --- | | **Test Case ID** | **Test Scenario** | **Expected Result** | **Status** | | TC001 | User Registration with valid data | Account successfully created | Pending | | TC002 | User Registration with invalid email | Error message displayed | Pending | | TC003 | Login with correct credentials | User logged in successfully | Pending | | TC004 | Login with incorrect password | Error message displayed | Pending | | TC005 | Place an order with valid items | Order placed successfully | Pending | | TC006 | Place an order without selecting items | Error message displayed | Pending | | TC007 | Payment processing with valid card | Payment successful | Pending | | TC008 | Payment processing with invalid card | Payment declined | Pending | |

**2. Automated Testing (if applicable)**

If automated testing is implemented, the following scripts and frameworks are used:

* **Selenium** for UI testing.
* **NUnit/XUnit** for backend unit testing.
* **Postman/Newman** for API testing.

## 3. Bug Reports

A structured bug tracking system is used to log, track, and resolve issues. The following template is used for reporting bugs:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Bug ID | Description | Steps to Reproduce | |  |  | | --- | --- | | Expected Result |  | | Actual Result | Status |
| BUG001 | |  |  | | --- | --- | | Login failure with valid credentials |  | | 1. Open login page 2. Enter valid credentials 3. Click login | User should log in successfully | Error message appears | Open |
| BUG002 | Payment system crashes on checkout | 1. Add items to cart 2. Proceed to checkout 3. Enter payment details 4. Click submit | Payment should be processed | System crashes | In Progress |

Bug reports are maintained in a **tracking system (e.g., Jira, Trello, GitHub Issues)** to ensure timely resolution.