*PRODUCT DESIGN Specification*

*PIZZA Connection*

*DESIGN Document*

*VERSION 2.0 | 12/05/2023*

*APPROVED BY SAEED BAKHSHAN*

*KANTA ISLAM | Matthew kROL | Mehedi Zihad | William Esparza*

VERSION HISTORY

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Version # | Implemented By | Revision Date | Approved By | Approval Date | Reason |
| 1.0 | Kanta Islam  Matthew Krol  Mehedi Zihad  William Esparza | 10/18/2023 |  |  | First Draft |
| 1.1 | 10/21/2023 |  |  | Feedback |
| 2.0 | 12/05/23 |  |  | Final |

[1. Introduction 3](#_Toc433579082)

[1.1 PURPOSE OF THE PRODUCT DESIGN SPECIFICATION Document 3](#_Toc343333524)

[2.1. GENERAL OVERVIEW 4](#_Toc1543542162)

[2.3. ASSUMPTIONS & CONSTRAINT 7](#_Toc198324435)

[3. Architecture Design 10](#_Toc1972840891)

[3.1. HARDWARE ARCHITECTURE 10](#_Toc1724673919)

[3.2. SOFTWARE ARCHITECTURE 11](#_Toc1646035065)

[3.3. SECURITY ARCHITECTURE 12](#_Toc1013518016)

[3.4. COMMUNICATION ARCHITECTURE 12](#_Toc1680549192)

[4. System Design 13](#_Toc90860141)

[4.1. USE-CASE 13](#_Toc108448650)

[4.1.1. USE\_CASE-1 13](#_Toc655172531)

[4.1.2. USE\_CASE-2 14](#_Toc2022906688)

[4.1.3. USE\_CASE-3 15](#_Toc1131449154)

[4.1.4. USE\_CASE-4 17](#_Toc779635285)

[4.1.5. USE\_CASE-5 18](#_Toc1657293554)

[4.1.6. USE\_CASE-6 20](#_Toc897030001)

[4.1.7. USE\_CASE-7 21](#_Toc1238405160)

[23](#_Toc636162646)

[4.1.8. USE\_CASE-8 23](#_Toc770365834)

[4.1.9. USE\_CASE-9 24](#_Toc472589199)

[4.1.10. USE\_CASE-10 25](#_Toc581213856)

[4.1.11. USE\_CASE-11 26](#_Toc21033739)

[4.1.12. USE\_CASE-12 27](#_Toc601679225)

[4.1.13. USE\_CASE-13 28](#_Toc353473522)

[4.1.14. USE\_CASE-14 29](#_Toc44480948)

[4.1.15. USE\_CASE-15 30](#_Toc30430501)

[4.1.16. USE\_CASE-16 30](#_Toc455972117)

[4.1.17. USE\_CASE-17 31](#_Toc1759055140)

[4.1.18. USE\_CASE-18 33](#_Toc649229388)

[4.1.19. USE\_CASE-19 34](#_Toc888233016)

[4.2. SEQUENCE DIAGRAM 36](#_Toc1823305933)

[4.3. DATA FLOW DIAGRAM 46](#_Toc1594567358)

[4.4. DATABASE DESIGN 47](#_Toc1377855258)

[4.5. APPLICATION PROGRAM INTERFACES 47](#_Toc1545949037)

[4.6. USER INTERFACE DESIGN 48](#_Toc285707655)

[5. PRODUCT DESIGN SPECIFICATION APPROVAL 51](#_Toc1443422632)

[Appendix A: References 52](#_Toc687624961)

[Appendix B: Key Terms 52](#_Toc1368185639)

# Introduction

## 1.1 PURPOSE OF THE PRODUCT DESIGN SPECIFICATION Document

The product design specification document tracks the necessary information required to effectively define architecture and system design to provide guidance for development. The primary purpose is to offer clear and detailed guidance to the development team. It serves as a reference point for architects, designers, and developers, helping them understand the overall architecture and design goals.

**Architecture definition:** It defines the architecture of the system or product to be developed. This includes the high-level structure, components, modules, and their interactions, ensuring that the development team follows a coherent and well-planned approach.

**System design:** It outlines the detailed system design, specifying how various components will function together to meet the project's objectives, including data flows, algorithms, and technical specifications.

**Project planning:** The product design specification document is typically created during the project's planning phase. Its intended audience is the project manager, aiding in project planning by providing a clear roadmap for development, enabling teams to estimate timelines, distribute resources, and identify potential risks.

**Communication tool:** It serves as a communication tool within the project team and development team. Portions of it can be used by project managers, architecture, and developers to ensure everyone is on the same page regarding design and architecture decisions.

Client/user involvement: Portions of the document, such as the user interface (UI) section, may occasionally be shared with clients and users when their input and approval into the UI are needed. This helps align the product design with user expectations and requirements.

Documentation and traceability: It provides a structured record of design decisions and requirements valuable for auditing, quality assurance, and maintaining a traceable history of design changes.

**Risk mitigation:** By specifying design details and potential challenges, the document can help identify and mitigate risks early in the development process, reducing the likelihood of costly design changes later.

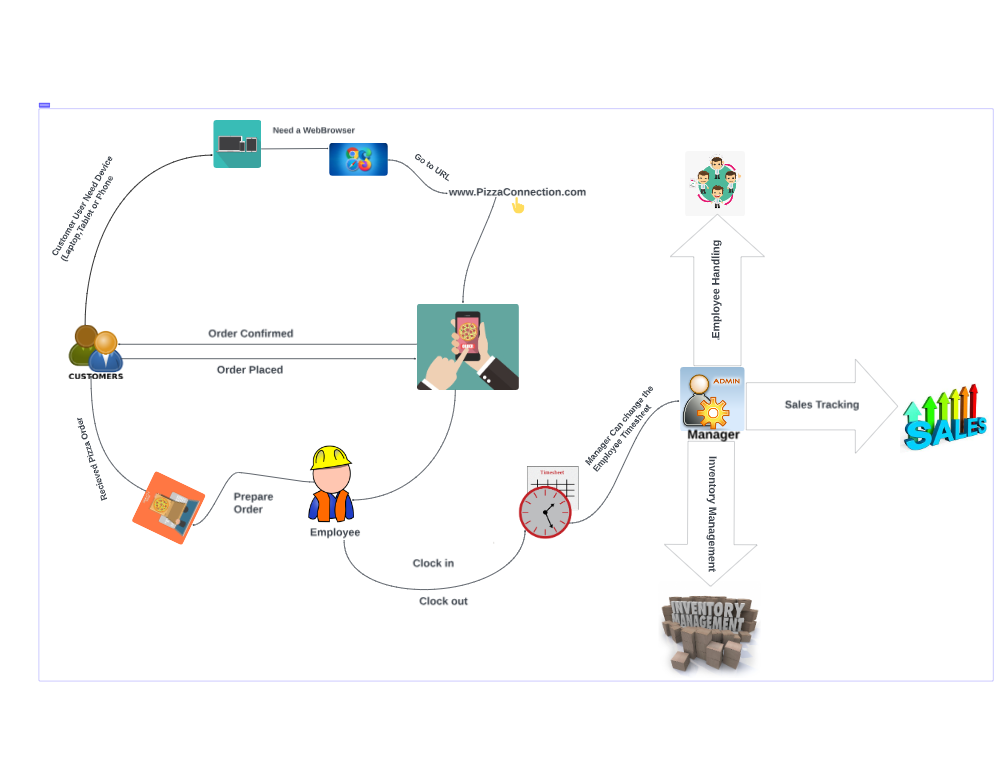
**Quality assurance:** It contributes to the overall product quality by setting clear design standards and expectations, helping prevent design flaws and inconsistencies.

**Reference for future phases:** the product design specification document can serve as a reference for later project phases, including implementation, testing, and maintenance, ensuring that the original design intent is preserved throughout the product's lifecycle.

The product design specification document is like a project's playbook. It is super important because it helps everyone involved understand how to make the product, what needs to be done, and what users want. It is like a map that keeps the project on track and helps avoid problems.

# GENERAL OVERVIEW

The "Pizza Connection" web application is a comprehensive solution designed to cater to the needs of both pizza businesses and their customers. It offers a user-friendly interface that integrates various functionalities, including customer account management, online ordering, employee time tracking, inventory management, employee management, and sales tracking. This multifaceted approach provides a unique value proposition in the pizza industry, making it both convenient and accessible for businesses and customers.



Customers have the option to sign up and create an account using their name, email address, and password. We want to emphasize that we do not collect sensitive information, as these details are solely used for the convenience of placing pizza orders. Customers can also make payments using external methods like PayPal or Google Pay, enhancing their order experience.

In a separate context, admins have the opportunity to sign up as customers. Subsequently, the manager, who serves as the admin for user accounts, can provide the employee with the necessary credentials by adding their information to the user database. Once the manager grants approval, employees gain access to their dedicated employee dashboard, allowing them to clock in and clock out for their shifts. They also retain the ability to update general profile information, such as their name, date of birth, and address. However, any changes beyond these details, such as their role or time clock adjustments, require managerial intervention.

If an employee forgets to clock in on time due to any reason, they must seek the manager's assistance to rectify this. Furthermore, employees have access to the order list to facilitate order preparation. It's important to note that only the manager possesses access to inventory management, employee handling, and sales tracking. This access enables them to gather crucial insights into the business's status, aiding them in making informed decisions to drive the business forward.

* 1. **DESIGN GUIDELINES/APPROACH**

System Architecture:

* Use the MERN stack for development.
* MongoDB for database storage, Express.js for the backend server, React for the frontend, and Node.js to run JavaScript on the server.
* Employ a RESTful API for communication between the frontend and backend.

Database Design:

* Create MongoDB collections for Customers, Employees, Orders, and other relevant data.
* Define the schema for each collection to store the required information.

Security Authentication and Authorization:

* Implement user authentication using a library like JSON Web Tokens (JWT).
* Differentiate between customers and employees, with varying levels of access.
* Use middleware to enforce authorization based on user roles.
* Use HTTPS to secure data transmission.

User Registration and Profile Management:

* Create registration and login forms in the React front end.
* Develop APIs for user registration, login, and profile management on the Express backend.
* Securely store user passwords using bcrypt.

External Payment Integration:

* Implement secure stripe API endpoints for processing payments.

Employee Management:

* Allow employees to sign up as customers but differentiate them in the system.
* Implement a mechanism for managers to approve and manage employee accounts.

Time Tracking:

* Create a clock-in and clock-out feature for employees within their dashboard.
* Implement logic to record time entries and track working hours.

Manager Controls:

* Develop manager-specific interfaces for inventory management, employee handling, and sales tracking.
* Ensure that only authorized managers have access to these features.

User Dashboards:

* Design user-friendly dashboards for customers and employees using React.
* Provide access to profile editing, order placement, tracking, and time clock functionalities.

Password Recovery:

* Implement a password reset feature that sends a reset email to the user's registered email address.
* Include a link in the email to reset the password securely.

Inactivity Timeout:

* Implement logic to automatically log out users after a specified period of inactivity.

Testing and Quality Assurance:

* Thoroughly test the application, both manually and through automated testing, to ensure functionality and security.
* Perform load testing to evaluate the system's performance under heavy usage.

Deployment:

* Deploy the frontend (React) and backend (Express/Node.js) components on hosting by AWS service.

Monitoring and Maintenance:

* Set up monitoring and logging to track system performance and identify issues.
* Regularly update and maintain the application to address bugs and security vulnerabilities.

## ASSUMPTIONS & CONSTRAINT

* The project operates within specified time constraints and is executed by a limited team of four members. Its primary focus is on the development of high-priority features to achieve the project's objectives within the designated timeframe.
* Due to the utilization of free version of MongoDB and API, the project has specific data storage constraints. Free version of MongoDB allows 500 mb & Free version of API 200 limit.
* The project is assuming the use of a specific technology stack, with consideration for compatibility and resource availability.
  1. **STANDARD**

**File and Folder Structure**: Organize the project into separate folders for the front-end (React), back-end (Node.js/Express), and database (MongoDB) components. Maintain a clear folder structure for components, routes, middleware, and models.

**Naming Conventions:** Use descriptive and meaningful names for variables, functions, and classes in both the front-end and back-end. Name API routes and database tables in a clear, consistent, and self-explanatory manner.

**Code Formatting:**

In our web application enforcing consistent code formatting and indentation is crucial for maintaining code readability, consistency, and collaboration among developers. This practice is achieved using linters like ESLint. Here's how this is implemented in the distinct parts of the stack:

**Front-End (React):**

**ESLint Configuration:** Configure ESLint to define coding standards and enforce best practices for React components, JavaScript, and JSX syntax.

**Indentation:** Use a consistent indentation style (commonly 2 or 4 spaces) to ensure that code is structured uniformly throughout the front-end codebase.

**Consistent Naming Conventions:** Enforce a consistent naming convention for variables, functions, and components to improve code maintainability and readability.

**Code Consistency:** Ensure that code lines and blocks are consistently formatted to improve visual clarity. Avoid long lines of code by breaking them into smaller, manageable segments.

**Code Comments:** Encourage the use of comments and docstrings to explain complex logic, components, or functions. Clearly document the purpose of components, their props, and any side effects.

**Back-End (Node.js/Express):**

**ESLint Configuration:** Set up ESLint for the Node.js/Express part of the application. Define coding standards for JavaScript, routes, middleware, and database interactions.

**Indentation:** Maintain a uniform indentation style, ensuring that routes, middleware, and database queries are consistently formatted.

**Consistent Variable Names:** Adhere to a consistent naming convention for variables, functions, and routes. This promotes better code understanding and collaboration.

**Route Organization:** Structure routes and controllers logically and consistently, making it easier for developers to find specific endpoints and related functionality.

**Error Handling:** Implement standardized error handling throughout the codebase. Use meaningful error messages and status codes for clear identification of issues.

**Overall Best Practices:**

**Version Control:** Utilize version control systems like Git to track and manage changes to the codebase, ensuring that code is well-documented through meaningful commit messages.

**Peer Review:** Encourage peer code reviews to maintain code quality and consistency. Code reviews help catch issues early and ensure that best practices are followed.

**Automated Formatting:** Integrate code formatting tools like Prettier into the development workflow to automatically format code to match the defined standards.

Enforcing consistent code formatting and indentation is a fundamental aspect of software development best practices. It enhances code quality, readability, and maintainability while fostering a collaborative development environment where team members can work seamlessly together.

**UI Design and Styling:** Maintain a consistent and visually appealing user interface design throughout the application. Use the provided color scheme for the UI elements to maintain brand consistency.

**Color Usage:** Use the main color #f5370c for headers, footers, and large color areas in the user interface to create a strong visual identity.

* Utilize the secondary color #ffc421 for accents or small areas to draw attention to specific elements or actions.
* The tertiary color #ebeae8 should be applied to tabs, bars, and other UI components to provide a clear visual distinction.
* For backgrounds or text that will be displayed against the tertiary color, use the full white background #ffffff to ensure readability.
* Maintain text readability by using the full black text color #000000 for text elements, ensuring good contrast with the background.

**CSS and Styling:** Use CSS preprocessors (e.g., SASS or LESS) to write organized and maintainable styles. Employ a modular and component-based approach for styling, encapsulating styles within components. Follow a naming convention for CSS classes that reflects their purpose and usage to improve code maintainability.

**Responsive Design:** Ensure that the user interface is responsive and adapts to various screen sizes and devices, including mobile, tablet, and desktop.

Use media queries to optimize the layout and styling for different screen sizes.

**Error Handling:** Implement proper error handling throughout the project, with meaningful error messages and status codes. Log errors and security-related events for auditing and debugging purposes.

**Modularization:** Encourage modularization of code into small, reusable components, both in the front-end and back-end. Use ES6 modules (import/export) for client-side code and CommonJS for server-side code.

**API Design:** Follow REST API design principles for Express.js routes. Clearly define and document API routes, their methods, and their expected behavior.

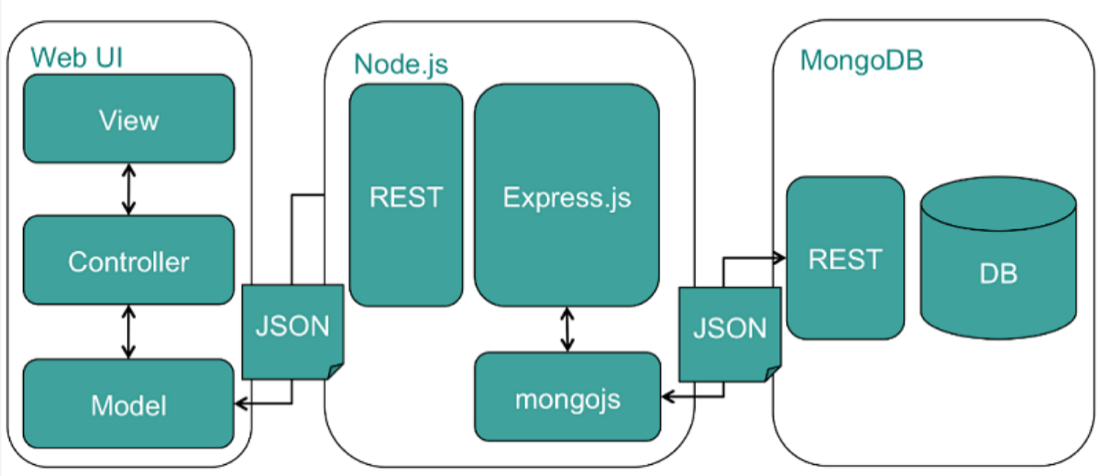
**Security:** Implement and follow security best practices, such as using JSON Web Tokens (JWT) and bcrypt for user authentication and data protection. Keep sensitive data and security credentials securely stored, away from the codebase. Enforce secure communication with the database to prevent SQL injection and other vulnerabilities.

**Testing:** Write unit tests and integration tests, especially for authentication and security-related code. Use testing libraries like Jest for Node.js and React Testing Library for React.

**Version Control:** Use a version control system like Git and follow branching and merging strategies suitable for collaborative development. Keep sensitive information, such as API keys and credentials, out of the version control system.

**Dependency Management:** Use a package manager (npm) to manage project dependencies. Keep dependencies up to date to ensure security patches are applied promptly.

**Communication Architecture:** Ensure that REST API endpoints are well-documented, and their usage is clear for developers working on the project. Implement secure communication practices for password resets and order updates via email, including appropriate validation and verification processes.



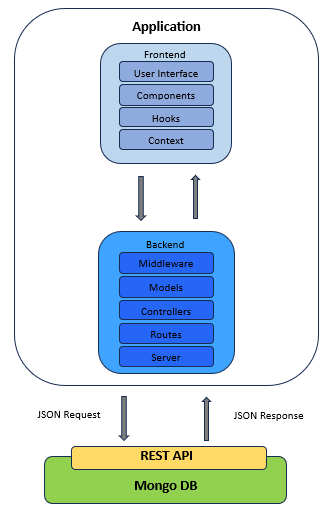
# Architecture Design

## HARDWARE ARCHITECTURE

The hardware utilized by this application is minimum, the only hardware interface being through devices that any user seeking to utilize this application would use. These devices would be any desktop, laptop, tablet, or smartphone that can run any modern web browser. The system requirements to run this application are any computer with at least 4GB of RAM (Google Chrome & Microsoft Edge require at least 4GB of RAM to run properly on any system, other browsers may utilize the same or less RAM), 64GB of storage memory and has a central processing unit that can run at least 1GHz speeds or more, as these would be the minimum requirements needed to run any modern browser with the latest updates. The browser also must be capable of running applications that utilize the latest version of HTML 5 standards.

## SOFTWARE ARCHITECTURE

Pizza Connection's software architecture consists of the front & backend layers in JavaScript, then there are API connections between the backend and the database, finally we have the MongoDB database server.



The front-end sublayers consist of the UI, components, hooks, and context. The UI, components, and pages are responsible for displaying the actual site that the user sees and interacts with. The hooks and context sub-layers are responsible for handling the server responses (JSON data) and provide system updates on the context state of the data of a user (if data has been retrieved, sent, errors, etc.) and what it will be used for. The backend consists of the middleware, models, routes, controllers and server sublayers. The middleware handles the authentication token checks to ensure that a user is authenticated. The models set the schemas for the data to be arranged in the way the table in the database is set up. Controllers handle the JSON methods and requests to perform the operations on the database, routes make use of the controller functions to create the API pathways/routes to communicate these methods to the database. The server sublayer handles the actual communication between the backend and the database itself, using REST API calls. The final layer, MongoDB, is where the data itself is stored, which accepts incoming requests and if successful will respond with JSON bodies back to the server sublayer of the backend.

## SECURITY ARCHITECTURE

Our security architecture consists of two components, JSON web tokens and bcrypt salt and hashing of passwords.

* JSON Web Tokens: A JSON Web Token is generated upon login or signup, which is then added to the payload of the user's information when this information is sent to the database. This ensures the user's information stays private and encrypted. The web token is also used for authentication & persistence whilst the user is logged into their account. Each user's web token will expire in 3 days after creation, during which time the user can access their account within those 3 days without having to log in again. After those 3 days if they wish to access their account, they must log in again.
* Bcrypt: The user's password is salted and hashed when their account is created during the signup phase. This library used in the react framework allows for passwords to be encrypted for added security.
* Payment: We will be using Stripe, which comes with its own security layers and will only communicate using secure and secret API keys to share information between the application and Stripe. Authentication will occur by use of HTTP basic auth protocol.

## COMMUNICATION ARCHITECTURE

Communication primarily happens between the application (backend) and the database server through using REST API format to obtain, modify, create, or delete information. External communication happens between the application and user's email for password resets and order updates, and between the application and Stripe for payment.

* Email: Once the user either attempts to reset their password or places an order, the application will send out an email by way of emailjs so the user is able to perform the aforementioned functions.
* Payment: This communication will utilize API keys to connect to Stripe. With these API calls they will be performed over HTTPS as that is the only way to ensure successful communication between the application and Stripe

# System Design

## USE-CASE

## USE\_CASE-1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Use Case ID:** | UC-1 | | | |
| **Use Case Name:** | Sign up | | | |
| **Created By:** | Matthew Krol | | **Last Updated By:** | William Esparza |
| **Date Created:** | 9-21-2023 | | **Last Revision Date:** | 9-30-2023 |
| **Actors:** | | Application User | | |
| **Description:** | | This use case will describe how an application user will sign up on the Pizza Connection website | | |
| **Trigger:** | | The user will click on the signup button | | |
| **Preconditions:** | | The system does not have a user logged in. User is on the signup page. | | |
| **Postconditions:** | | The user will receive a popup displaying that they have successfully signed up, and once they hit okay on the popup. They will be redirected to the home page. | | |
| **Normal Flow:** | | 1. User enters email and password. 2. User clicks the sign-up button. 3. User is displayed a success message. 4. User clicks okay on the success message. 5. Users are redirected to the home page. | | |
| **Alternative Flows:** | | N/A | | |
| **Exceptions:** | | When a user enters an invalid email, the system will stay on the sign up page and display an error saying, “Please enter a valid email”. When a user enters a password that does not meet requirements, the system will stay on the sign up page, and display an error saying, “Please enter a valid password”. | | |
| **Frequency of Use:** | | One time when a user signs up to Pizza Connection website | | |
| **Assumptions:** | | The database is available. The user has an email and password to create an account with. | | |

## USE\_CASE-2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Use Case ID:** | UC-2 | | | |
| **Use Case Name:** | Login | | | |
| **Created By:** | William Esparza | | **Last Updated By:** | Matthew Krol |
| **Date Created:** | 9-21-2023 | | **Last Revision Date:** | 9-30-2023 |
| **Actors:** | | Application User | | |
| **Description:** | | This use case will describe how an application user will login to the Pizza Connection website | | |
| **Trigger:** | | Log in button | | |
| **Preconditions:** | | The user is already on the login page. The system does not have a user already logged in. | | |
| **Postconditions:** | | The user will be redirected to the home page. | | |
| **Normal Flow:** | | 1. User enters email and password. 2. User clicks on the login button. 3. The user is redirected to the home page. | | |
| **Alternative Flows:** | | N/A | | |
| **Exceptions:** | | When a user enters an invalid email, the system will stay on the login page and display an error saying, “Please enter a valid email”. When a user enters a password that does not meet requirements, the system will stay on the login page, and display an error saying, “Please enter a valid password”. | | |
| **Frequency of Use:** | | Every time a user returns to the Pizza Connection website after being logged out. | | |
| **Assumptions:** | | The database is available. The user has an email and password that they previously signed up with. | | |

## USE\_CASE-3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Use Case ID:** | UC-3 | | | |
| **Use Case Name:** | Profile | | | |
| **Created By:** | Matthew Krol | | **Last Updated By:** | Matthew Krol |
| **Date Created:** | 10-2-2023 | | **Last Revision Date:** | 10-2-2023 |
| **Actors:** | | Application User | | |
| **Description:** | | This use case will describe how an application user will use the profile on the Pizza Connection website. | | |
| **Trigger:** | | Update Profile | | |
| **Preconditions:** | | The user must be logged in. | | |
| **Postconditions:** | | The user will see a popup displaying they have successfully updated the profile. They will remain on the profile page. | | |
| **Normal Flow:** | | 1. Users will click the button with a pencil icon on the field they wish to edit. 2. User enters the information. 3. User clicks on the save button. 4. The above process can be repeated to fill out all the fields in the profile page, or the user can continue to the next steps. 5. User clicks on the update profile button. 6. User will see a success popup. | | |
| **Alternative Flows:** | | N/A | | |
| **Exceptions:** | | Will display an error if trying to add a future DOB (Date of Birth). | | |
| **Frequency of Use:** | | Every time a user would like to edit the information in their profile. | | |
| **Assumptions:** | | The database is available. The user already has the information they would like to update in the profile. | | |

## 

## USE\_CASE-4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Use Case ID:** | UC-4 | | | |
| **Use Case Name:** | Add Item to Cart | | | |
| **Created By:** | Matthew Krol | | **Last Updated By:** | Matthew Krol |
| **Date Created:** | 10-10-2023 | | **Last Revision Date:** | 10-18-2023 |
| **Actors:** | | Application User | | |
| **Description:** | | This use case will describe how to add an item to the cart on the Pizza Connection Website. | | |
| **Trigger:** | | The user clicks on the menu tab on the sidebar. | | |
| **Preconditions:** | | The user is on the menu page. | | |
| **Postconditions:** | | The user will see a popup of “Successfully add to cart” then is redirected back to the menu. | | |
| **Normal Flow:** | | 1. User clicks on a customize button for the product they want. 2. User will be redirected to another page to customize the product. 3. User makes changes to the product such as size or what comes on it based on the selections there. 4. User clicks on the add to cart button. 5. User will receive a successful popup. | | |
| **Alternative Flows:** | | N/A | | |
| **Exceptions:** | | N/A | | |
| **Frequency of Use:** | | Every time a user would like to place an order on Pizza Connection. | | |
| **Assumptions:** | | The database is available. The user knows what they would like to place an order for. | | |

## USE\_CASE-5

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Use Case ID:** | UC-5 | | | |
| **Use Case Name:** | Place Order | | | |
| **Created By:** | Matthew Krol | | **Last Updated By:** | Matthew Krol |
| **Date Created:** | 10-10-2023 | | **Last Revision Date:** | 10-18-2023 |
| **Actors:** | | Application User | | |
| **Description:** | | This use case will describe how to use the pizza menu on the Pizza Connection Website. | | |
| **Trigger:** | | Place Order button. | | |
| **Preconditions:** | | The user is on the menu screen. | | |
| **Postconditions:** | | The user will see a popup of “Successfully ordered,” and then be shown a receipt of the order they have placed. | | |
| **Normal Flow:** | | 1. User will click on the cart button. 2. User will select pickup or delivery. 3. User will enter first name, last name, address, phone. 4. User will hit the place order button. 5. User will see a successful popup | | |
| **Alternative Flows:** | | N/A | | |
| **Exceptions:** | | N/A | | |
| **Frequency of Use:** | | Every time a user would like to place an order on Pizza Connection. | | |
| **Assumptions:** | | The database is available. The user already has items in the cart. | | |

## 

## USE\_CASE-6

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Use Case ID:** | UC-6 | | | |
| **Use Case Name:** | Employee Create | | | |
| **Created By:** | Mehedi Zihad | | **Last Updated By:** | Mehedi Zihad |
| **Date Created:** | 10-10-2023 | | **Last Revision Date:** | 10-18-2023 |
| **Actors:** | | Application User (Admin) | | |
| **Description:** | | This will allow a user with admin access to add a new user to the database. | | |
| **Trigger:** | | Add Employee | | |
| **Preconditions:** | | The user is logged in as an admin user. The user is on the employee management page. | | |
| **Postconditions:** | | The user will see a popup message saying, “Employee created successfully.” and will remain on the same page. | | |
| **Normal Flow:** | | 1. Admin clicks on the employee Create link. 2. Admin will enter the information of the new user. 3. Admin will click the add user button. 4. Admin will see a success popup | | |
| **Alternative Flows:** | | N/A | | |
| **Exceptions:** | | There will be an error display if a user enters a future date. If a user enters an invalid hourly rate. If a user enters an invalid password. | | |
| **Frequency of Use:** | | Every time an admin would like to add a new employee. | | |
| **Assumptions:** | | The database is available. The admin has all the information to create a new employee. | | |

## USE\_CASE-7

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Use Case ID:** | UC-7 | | | |
| **Use Case Name:** | Employee Edit | | | |
| **Created By:** | Mehedi Zihad | | **Last Updated By:** | Mehedi Zihad |
| **Date Created:** | 10-10-2023 | | **Last Revision Date:** | 10-18-2023 |
| **Actors:** | | Application User (Admin) | | |
| **Description:** | | This will allow a user with admin access to edit the information of another user. | | |
| **Trigger:** | | Save button | | |
| **Preconditions:** | | User is logged in as an admin user, and they are on the home page | | |
| **Postconditions:** | | The user will see a popup message saying, “Employee updated successfully” and will remain on the same page. | | |
| **Normal Flow:** | | 1. Admin clicks the employee management button. 2. Admin will search for the employee they want to edit. 3. Admin will click the edit button they want to edit. 4. Admin will change the desired information and click on save. 5. A display will pop up with a successful message. | | |
| **Alternative Flows:** | | N/A | | |
| **Exceptions:** | | There will be an error display if a user enters a future date. If a user enters an invalid hourly rate. If a user enters an invalid password. | | |
| **Frequency of Use:** | | Every time an admin needs to create a new employee. | | |
| **Assumptions:** | | The database is available. The admin has information already to edit. | | |

## 

## USE\_CASE-8

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Use Case ID:** | UC-8 | | | |
| **Use Case Name:** | Delete Employee | | | |
| **Created By:** | Mehedi Zihad | | **Last Updated By:** | Mehedi Zihad |
| **Date Created:** | 10-10-2023 | | **Last Revision Date:** | 10-18-2023 |
| **Actors:** | | Application User (Admin) | | |
| **Description:** | | This will allow a user with admin access to delete an employee from the database. | | |
| **Trigger:** | | Delete button | | |
| **Preconditions:** | | The user is logged in as an admin user. The user is on the home page. | | |
| **Postconditions:** | | The user will see a popup message saying, “Employee deleted successfully.” and will remain on the same page.  If the deletion fails, the user will see a popup message indicating the failure and will remain on the same page. | | |
| **Normal Flow:** | | 1. Admin clicks on the employee delete button. | | |
| **Alternative Flows:** | | N/A | | |
| **Exceptions:** | | N/A | | |
| **Frequency of Use:** | | Every time an admin would like to delete an employee. | | |
| **Assumptions:** | | The database is available. The admin has which employee to delete. | | |

## USE\_CASE-9

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Use CaseID:** | UC-9 | | | |
| **Use Case Name:** | Edit Time Clock | | | |
| **Created By:** | Mehedi Zihad | | **Last Updated By:** | Mehedi Zihad |
| **Date Created:** | 10-10-2023 | | **Last Revision Date:** | 10-18-2023 |
| **Actors:** | | Application User (Admin) | | |
| **Description:** | | This will allow an admin to edit the employee’s time clock. | | |
| **Trigger:** | | Update button | | |
| **Preconditions:** | | The user is logged in as an admin user. The user is on the employee management page. | | |
| **Postconditions:** | | Once the admin clicks submit, a popup message saying, “Successfully updated time clock” will appear, and redirect you to the employee management page. | | |
| **Normal Flow:** | | 1. Admin clicks on the time sheet button next to a specific user. 2. A list of the employees’ time clocks will be displayed. 3. Click the edit button next to one of the entries to change it. 4. Admin clicks the submit button to update the time clock entry. 5. A successful popup message will be displayed | | |
| **Alternative Flows:** | | N/A | | |
| **Exceptions:** | | Will make sure the date and time is formatted correctly. | | |
| **Frequency of Use:** | | Every time an admin needs to change an employee’s timeclock | | |
| **Assumptions:** | | The database is available. Admin has the correct time clock information. | | |

## USE\_CASE-10

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Use CaseID:** | UC-10 | | | |
| **Use Case Name:** | Add Time Clock | | | |
| **Created By:** | Mehedi Zihad | | **Last Updated By:** | Mehedi Zihad |
| **Date Created:** | 10-10-2023 | | **Last Revision Date:** | 10-18-2023 |
| **Actors:** | | Application User (Admin) | | |
| **Description:** | | This will allow an admin to add the employee’s time clock. | | |
| **Trigger:** | | Clock in or clock out button | | |
| **Preconditions:** | | The user is logged in as an admin user. The user is on the employee management page. | | |
| **Postconditions:** | | Once the admin clicks submit, a popup message saying, “Successfully added time clock” will appear, and redirect you to the employee management page. | | |
| **Normal Flow:** | | 1. Admin clicks on the time sheet button next to a specific user. 2. Admin will click on the add button. 3. Admin will enter the clock type. 4. Admin will enter the date and time. 5. Admin will hit the submit button. 6. A successful popup message will display. | | |
| **Alternative Flows:** | | N/A | | |
| **Exceptions:** | | Will make sure the date and time is formatted correctly. | | |
| **Frequency of Use:** | | Every time an admin needs to add an employee’s timeclock | | |
| **Assumptions:** | | The database is available. Admin has the correct time clock information. | | |

## USE\_CASE-11

|  |  |  |  |
| --- | --- | --- | --- |
| **Use Case ID:** | UC -11 | | |
| **Use Case Name:** | Forgot Password | | |
| **Created By:** | Kanta Islam | **Last Updated By:** | Kanta Islam |
| **Date Created:** | 9-21-2023 | **Last Revision Date:** | 9-21-2023 |
| **Actors:** | Application User | | |
| **Description:** | This will allow a user to receive an email to reset password. | | |
| **Trigger:** | User clicks on the link "Click Here” | | |
| **Preconditions:** | User is not logged in and is on the account page. | | |
| **Postconditions:** | The user will see a popup message saying, “Email sent,” and will remain on the forgotten password page. | | |
| **Normal Flow:** | 1. User clicks on the click here link 2. User enters their email. 3. User hits forgot password button. 4. User is displayed a popup message saying that an email was successfully sent. | | |
| **Alternative Flows:** | N/A | | |
| **Exceptions:** | Displays an error if the user is not already in the database. | | |
| **Frequency of Use:** | Every time a user forgets their password | | |
| **Assumptions:** | The database is available. | | |

## USE\_CASE-12

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Use Case ID:** | UC-12 | | | |
| **Use Case Name:** | Reset Password | | | |
| **Created By:** | Kanta Islam | **Last Updated By:** | Kanta Islam | |
| **Date Created:** | 9-21-2023 | **Last Revision Date:** | 9-21-2023 | |
| **Actors:** | Application User | | | |
| **Description:** | This will allow a user to reset their password. | | | |
| **Trigger:** | When a user clicks on the link from the email. | | | |
| **Preconditions:** | User has successfully requested an email from forgot password. | | | |
| **Postconditions:** | The user will see a successful message saying, “Password has been reset” and then returned to the home page. | | | |
| **Normal Flow:** | 1. User goes to their email provider. 2. User opens the email sent by the application. 3. User clicks the link in the email. 4. User is redirected back to the application’s reset password page. 5. User enters new password. 6. User confirms new password. 7. User clicks a reset password button. 8. User is displayed a success message. | | | |
| **Alternative Flows:** | N/A | | | |
| **Exceptions:** | An error message will appear if the user enters an invalid password. | | |
| **Frequency of Use:** | Every time a user forgets their password | | | |
| **Assumptions:** | The database is available. The email service is available. The user’s email is available. | | | |

## USE\_CASE-13

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Use Case ID:** | UC-13 | | | |
| **Use Case Name:** | Clock In | | | |
| **Created By:** | Kanta Islam | | **Last Updated By:** | Kanta Islam |
| **Date Created:** | 10-10-2023 | | **Last Revision Date:** | 10-18-2023 |
| **Actors:** | | Application User (Employee, Admin) | | |
| **Description:** | | This will allow an employee to clock in. | | |
| **Trigger:** | | When the user (Admin) clicks on the time sheet page. And Employee clicks on time clock. | | |
| **Preconditions:** | | The user is logged in as employee, manager, or owner. User is on the home page. | | |
| **Postconditions:** | | The user will see a “Successful clock in” message and return to the home page. | | |
| **Normal Flow:** | | 1. User (Admin) clicks on the time sheet page. 2. User (Employee) clicks on the time clock page. 3. User will hit the button clock in button. 4. User is given a successful popup. | | |
| **Alternative Flows:** | | N/A | | |
| **Exceptions:** | | N/A | | |
| **Frequency of Use:** | | Every time an employee will start their shift. | | |
| **Assumptions:** | | The database is available. | | |

## USE\_CASE-14

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Use Case ID:** | UC-14 | | | |
| **Use Case Name:** | Clock Out | | | |
| **Created By:** | Kanta Islam | | **Last Updated By:** | Kanta Islam |
| **Date Created:** | 10-10-2023 | | **Last Revision Date:** | 10-18-2023 |
| **Actors:** | | Application User (Employee, Admin) | | |
| **Description:** | | This will allow an employee to clock out. | | |
| **Trigger:** | | When the user (Admin) clicks on the time sheet page. And Employee clicks on time clock. | | |
| **Preconditions:** | | The user is logged in as employee, manager, or owner. User is on the home page. The user has started a shift. | | |
| **Postconditions:** | | The user will see a “Successful clocked out” message and return to the home page. | | |
| **Normal Flow:** | | 1. User (Admin) clicks on time sheet page. 2. User (Employee) clicks on the time clock page. 3. User will hit the button clock out button. 4. User is given a successful popup. | | |
| **Alternative Flows:** | | N/A | | |
| **Exceptions:** | | N/A | | |
| **Frequency of Use:** | | Every time an employee ends their shift. | | |
| **Assumptions:** | | The database is available. | | |

## USE\_CASE-15

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Use Case ID:** | UC-15 | | | |
| **Use Case Name:** | Order List | | | |
| **Created By:** | Mehedi Zihad | | **Last Updated By:** |  |
| **Date Created:** | 10-24-2023 | | **Last Revision Date:** |  |
| **Actors:** | | Application User (Admin) | | |
| **Description:** | | This will allow an admin to see the list of orders. | | |
| **Trigger:** | | Order List button on the Sidebar. | | |
| **Preconditions:** | | The user is logged in as an admin. The user is on the home page. | | |
| **Postconditions:** | | The user will see a table of the Order List. | | |
| **Normal Flow:** | | 1. User will click on the Order List page on the Sidebar. 2. User will see a table of the Order List 3. User can update the order status as in progress or completed. | | |
| **Alternative Flows:** | | N/A | | |
| **Exceptions:** | | N/A | | |
| **Frequency of Use:** | | Every time an admin wants to see the Order List | | |
| **Assumptions:** | | The database is available. | | |

## USE\_CASE-16

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Use Case ID:** | UC-16 | | | |
| **Use Case Name:** | Add Inventory Item | | | |
| **Created By:** | William Esparza | | **Last Updated By:** | William Esparza |
| **Date Created:** | 10-17-2023 | | **Last Revision Date:** | 11-25-2023 |
| **Actors:** | | Application User (Admin) | | |
| **Description:** | | This will allow an admin to add an inventory item | | |
| **Trigger:** | | ‘Add Inventory Item’ button | | |
| **Preconditions:** | | The user is logged in as an admin. The name of the item to be added must also be unique. | | |
| **Postconditions:** | | The inventory view/list will be updated with the new inventory item added by the admin user. | | |
| **Normal Flow:** | | 1. User clicks on the ‘Add Inventory Item’ button 2. User inputs the required information for each field: name, category, quantity & unit type. 3. User clicks on the ‘Save’ button 4. Inventory view list updates with the inventory item the user just added. | | |
| **Alternative Flows:** | | None | | |
| **Exceptions:** | | User will see an error message if the inventory item exists, or the inventory database table could not be accessed/updated. | | |
| **Frequency of Use:** | | Every time the admin user has brand new inventory to add. | | |
| **Assumptions:** | | The database is available. | | |

## USE\_CASE-17

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Use Case ID:** | UC-17 | | | |
| **Use Case Name:** | Edit Inventory | | | |
| **Created By:** | William Esparza | | **Last Updated By:** | William Esparza |
| **Date Created:** | 10-17-2023 | | **Last Revision Date:** | 11-25-2023 |
| **Actors:** | | Application User (Admin) | | |
| **Description:** | | This will allow an admin to edit the inventory list. | | |
| **Trigger:** | | ‘Edit Inventory’ button | | |
| **Preconditions:** | | The user is logged in as an admin and there is at least one item in the inventory list to update. | | |
| **Postconditions:** | | The inventory list is updated on both the edit inventory page and the inventory view page. | | |
| **Normal Flow:** | | 1. User clicks on the ‘Edit Inventory’ button. 2. The user clicks on any item in the list they wish to edit. 3. The user can choose to update any input field for any column for that item or can choose to delete that item. 4. The user clicks either the save button if making edits or the delete button if they wish to remove that inventory item from the list. 5. The inventory list is updated on both the edit inventory page and the inventory view page with the user’s changes (edited item details or removed item) | | |
| **Alternative Flows:** | | For inventory item quantity only:   1. User clicks on the ‘Add Tracking Entry’ button 2. User inputs all the required fields. 3. User clicks on the ‘Save’ button. 4. Depending on whether the user chose the ‘added’ or ‘used up’ option in the add tracking entry form the inventory view list will be updated with the new recalculated quantity for the specific inventory item the user chose to track and modify. | | |
| **Exceptions:** | | User will see an error message if input invalid quantity or if the quantity falls below zero. | | |
| **Frequency of Use:** | | At least once daily. | | |
| **Assumptions:** | | The database is available. | | |

## USE\_CASE-18

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Use Case ID:** | UC-18 | | | |
| **Use Case Name:** | Add Tracking Entry | | | |
| **Created By:** | William Esparza | | **Last Updated By:** | William Esparza |
| **Date Created:** | 11-13-2023 | | **Last Revision Date:** | 11-25-2023 |
| **Actors:** | | Application User (Admin) | | |
| **Description:** | | The user can add a tracking entry to track a specific inventory item and update the quantity of that item as they add or use up that item. | | |
| **Trigger:** | | ‘Add Tracking Entry’ button. | | |
| **Preconditions:** | | The user is logged in as an admin and there is at least one item in the inventory list. | | |
| **Postconditions:** | | A tracking entry appears in the inventory tracking view list. | | |
| **Normal Flow:** | | 1. User clicks on the ‘Add Tracking Entry’ button. 2. User enters all the details for the item being tracked. 3. User clicks on the ‘Save’ button. 4. Inventory item quantity increased or decreased in the inventory view list. 5. A tracking entry appears in the tracking view list. | | |
| **Alternative Flows:** | | None | | |
| **Exceptions:** | | An error appears if the form has incorrect information entered or if a connection cannot be established to the database. An error appears when the user attempts to deduct the quantity of an item below zero. | | |
| **Frequency of Use:** | | At least once daily. | | |
| **Assumptions:** | | The database is available. | | |

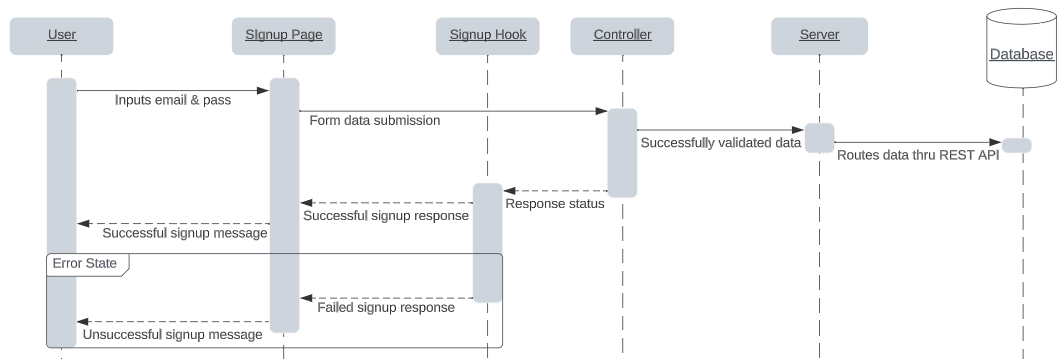
## USE\_CASE-19

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Use Case ID:** | UC-19 | | | |
| **Use Case Name:** | Payment | | | |
| **Created By:** | William Esparza | | **Last Updated By:** | William Esparza |
| **Date Created:** | 10-24-2023 | | **Last Revision Date:** | 12/3/23 |
| **Actors:** | | Application User/Customer | | |
| **Description:** | | The customer is redirected to the stripe payment page to pay and place their order in the system. | | |
| **Trigger:** | | ‘Place order’ button. | | |
| **Preconditions:** | | User is logged in and has at least one item in the cart. | | |
| **Postconditions:** | | The user is redirected to the order confirmation page with details about their order. | | |
| **Normal Flow:** | | 1. User clicks on the ‘place order’ button. 2. User enters their payment information. 3. User clicks on the pay button. 4. The button turns green indicating successful payment 5. The user is redirected to the order confirmation page. | | |
| **Alternative Flows:** | | None. | | |
| **Exceptions:** | | User will see an error message if it cannot verify the card number. User will see an error message if there is not enough money in the account. An error message also appears if the cart is empty, or the user is not logged in prompting the user to log in. | | |
| **Frequency of Use:** | | Every time a user/customer places an order. | | |
| **Assumptions:** | | There is an active internet connection, and the database is available. | | |

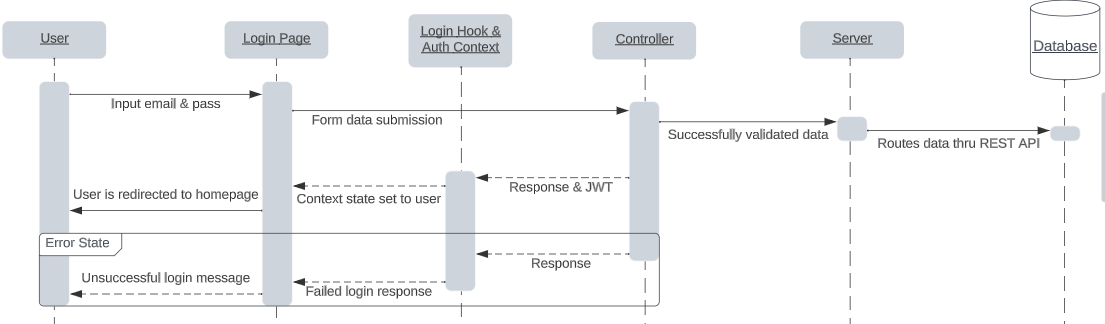
## 

## SEQUENCE DIAGRAM

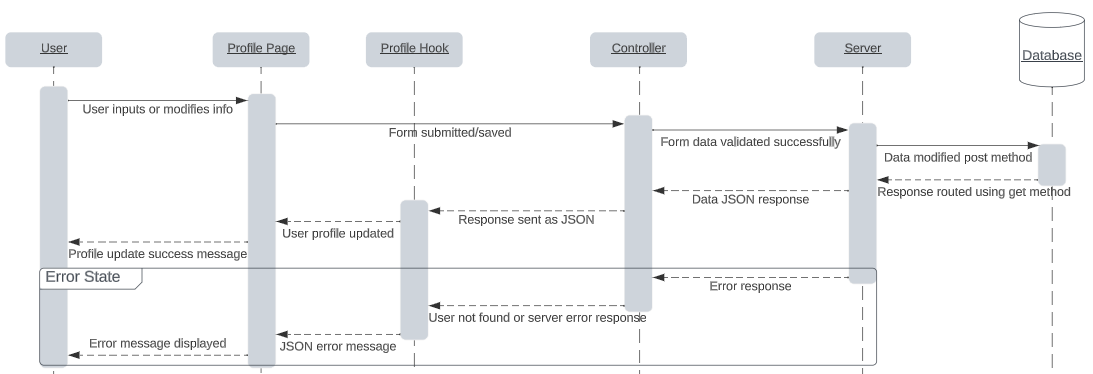
Sequence Diagram for Sign Up (UC-1)



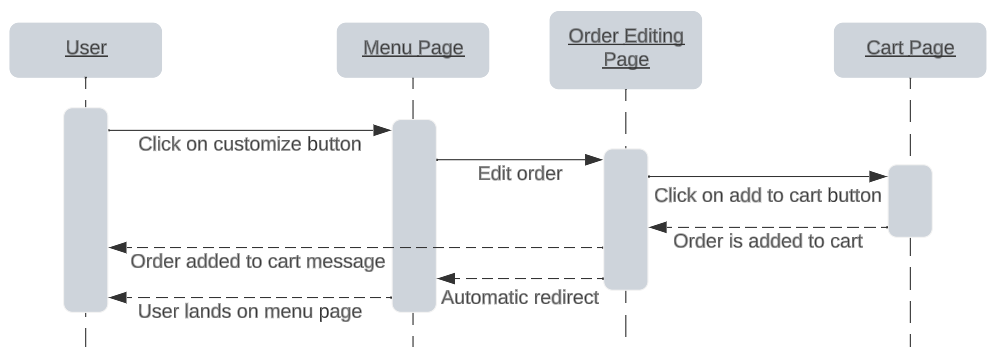
Sequence Diagram for Log In (UC-2)



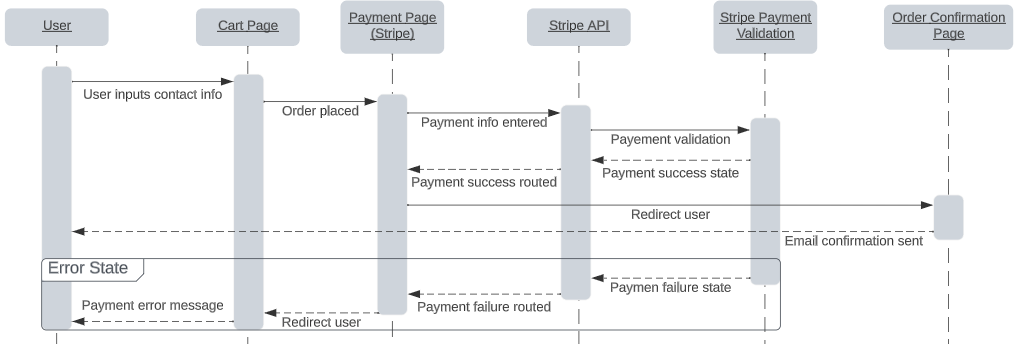
Sequence Diagram for Profile (UC-3)



Sequence Diagram for Adding Item to Cart (UC-4)



Sequence Diagram for Placing an Order (UC-5)



Sequence Diagram for Employee Create (UC-6)

A diagram of a company

Description automatically generated

Sequence Diagram for Employee Edit (UC-7)

A diagram of a company

Description automatically generated with medium confidence

Sequence Diagram for Employee Delete (UC-8) A diagram of a company

Description automatically generated

Sequence Diagram for Edit Time Clock (UC-9)

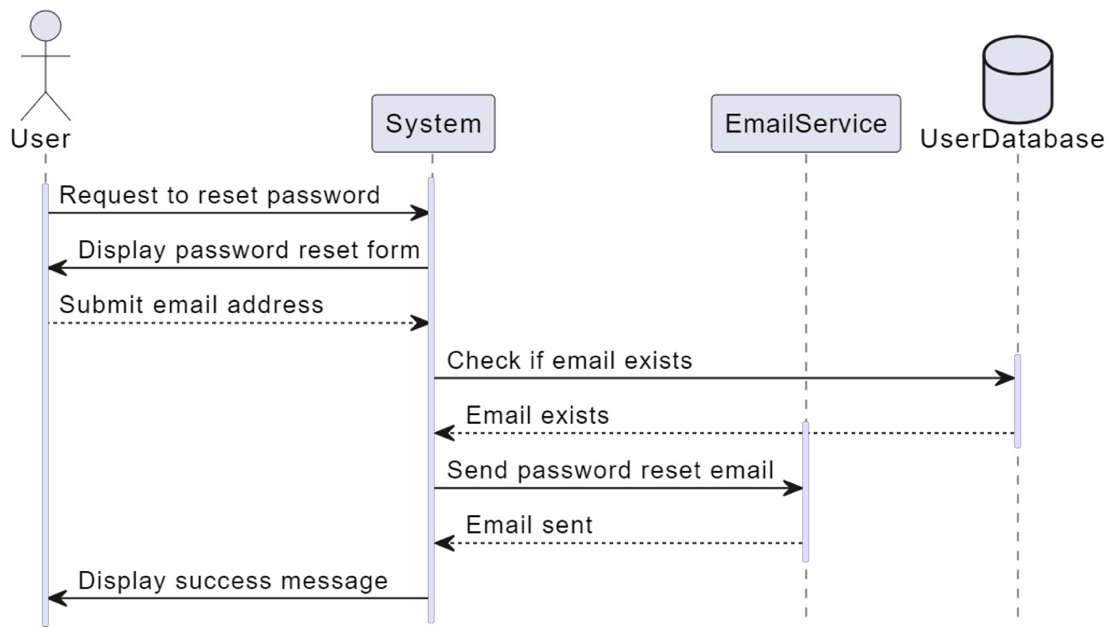


Sequence Diagram for Add Time Clock (UC-10)

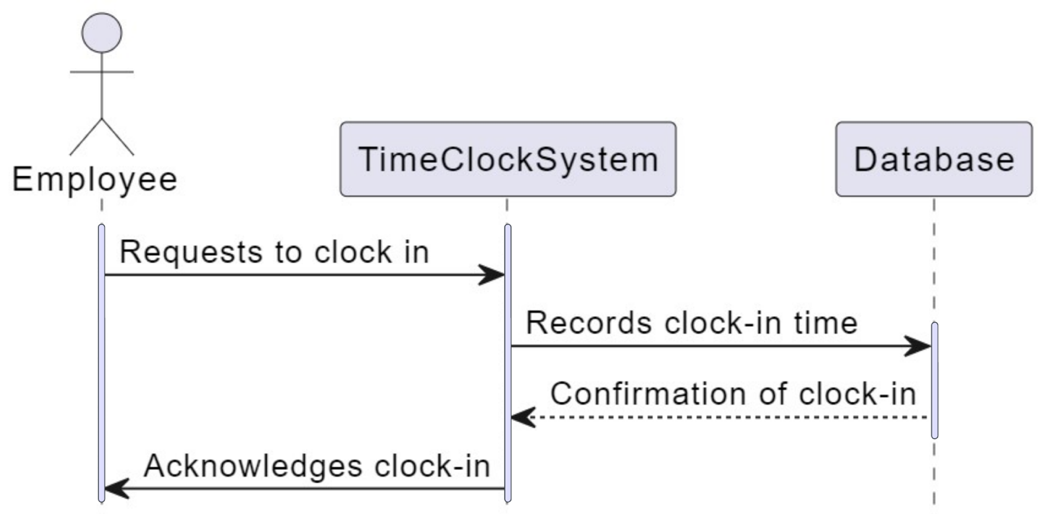
A diagram of a workflow

Description automatically generated

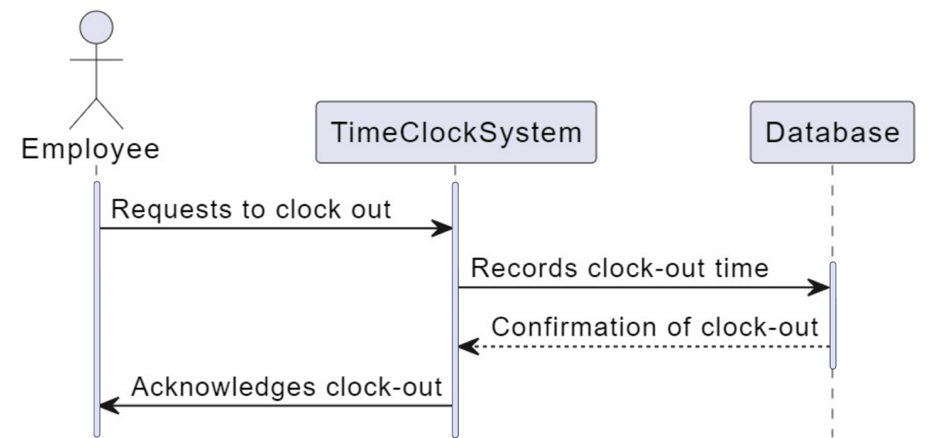
Sequence Diagram for Forgot and Reset Password (UC-11, UC-12)



Use Case for Employee Clock in (UC-13)



Sequence Diagram for Employee Clock Out (UC-14)

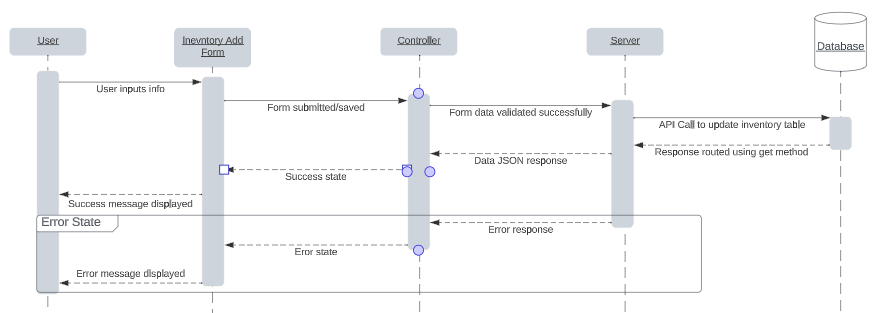


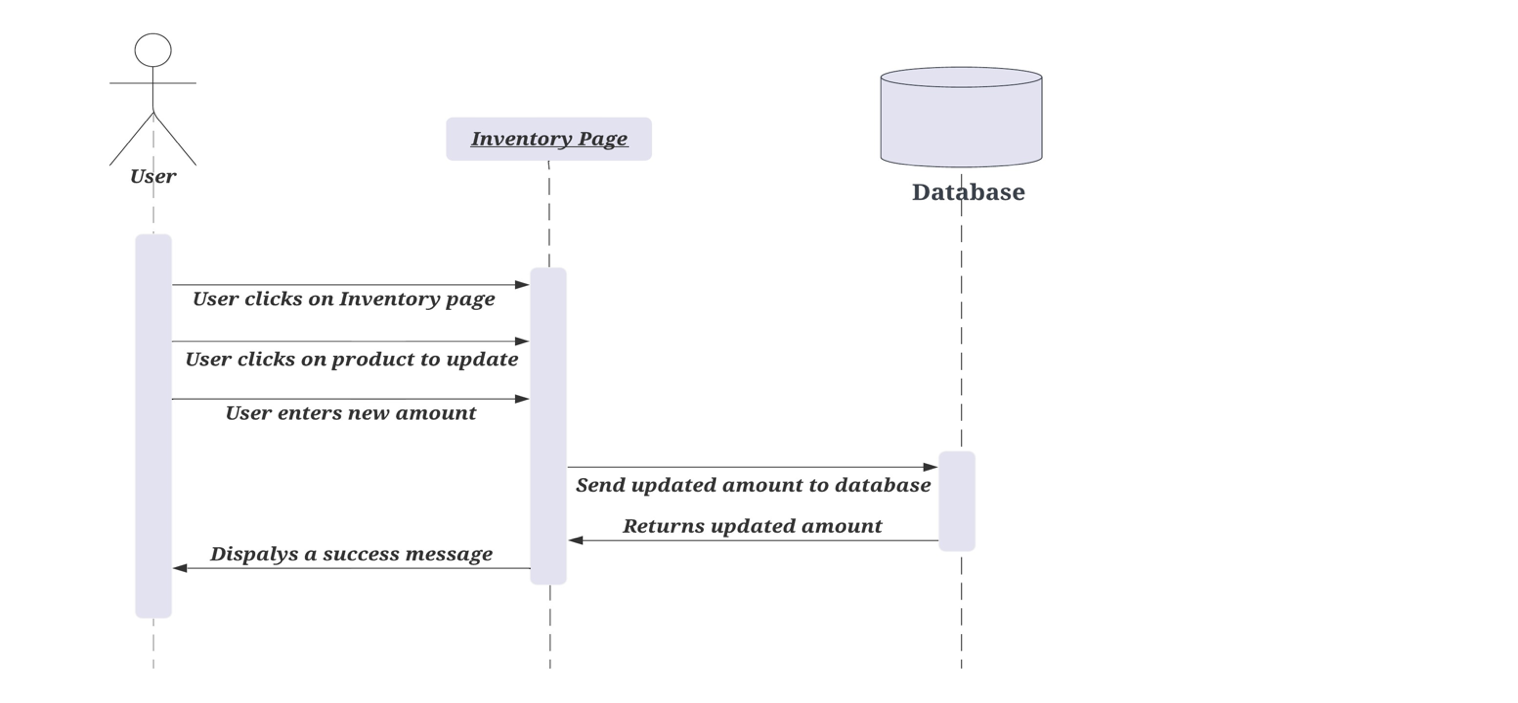
Sequence Diagram for Sale Tracking (UC-15)

A diagram of a sales process

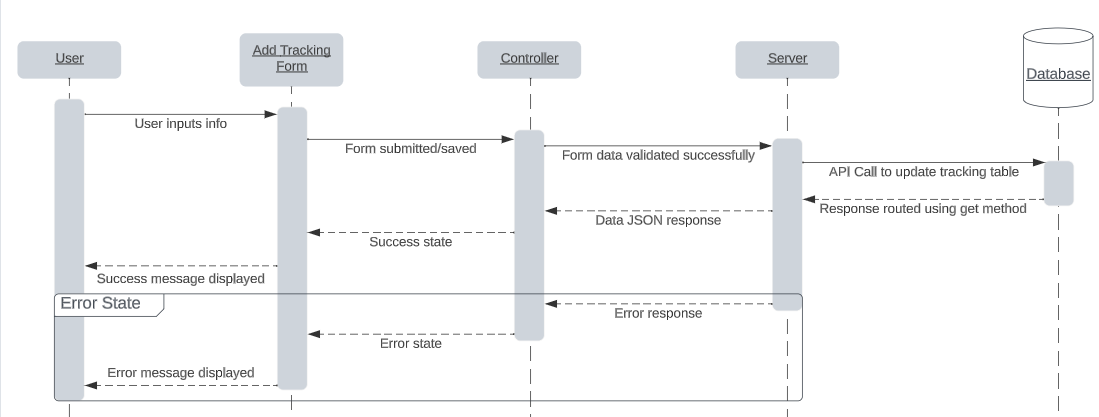
Description automatically generated

Sequence Diagram for Inventory Add (UC-16)

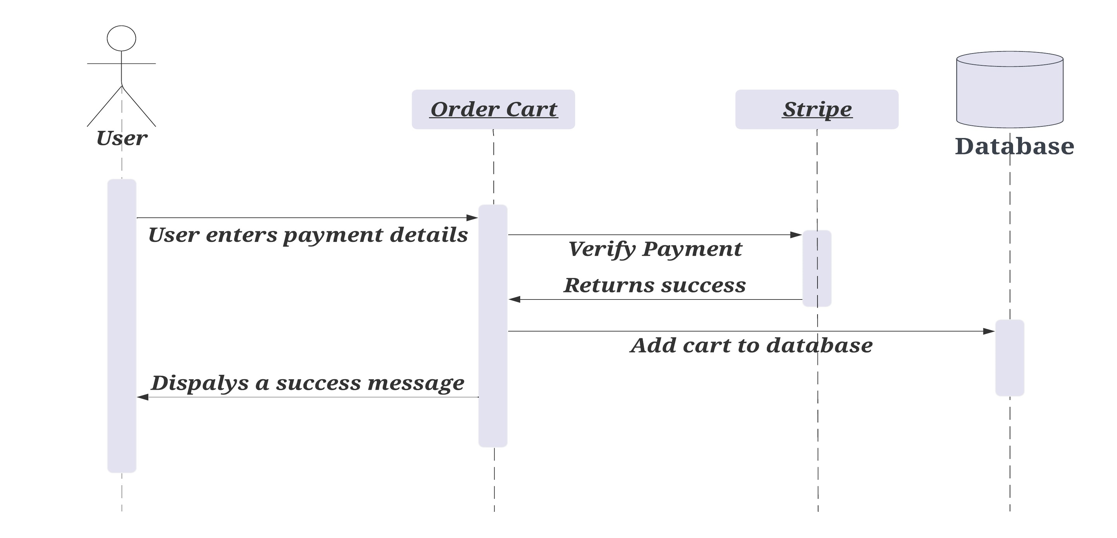


Sequence Diagram for Inventory Edit (UC-17)   


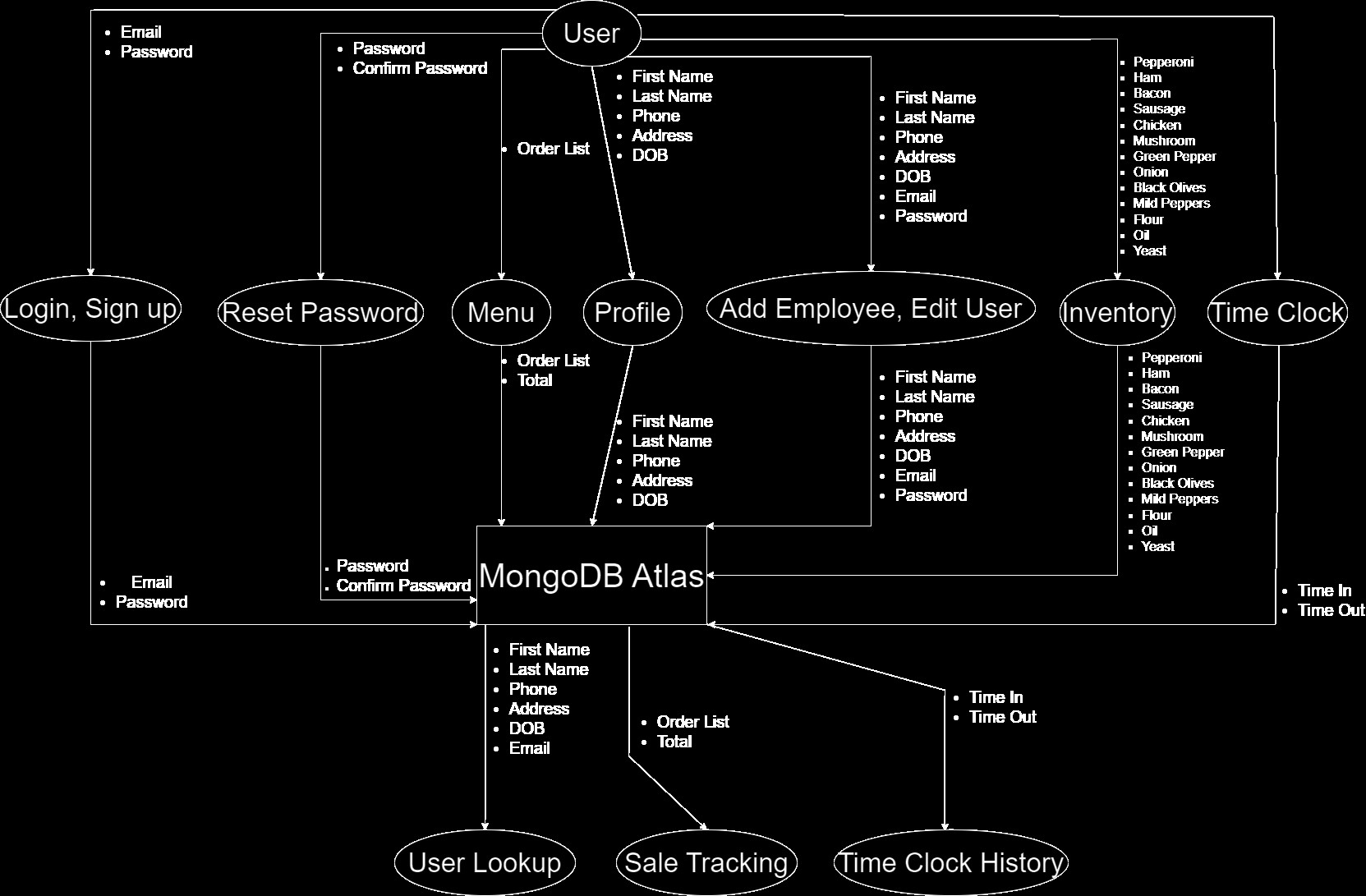
Sequence Diagram for Inventory Tracking (UC-18)



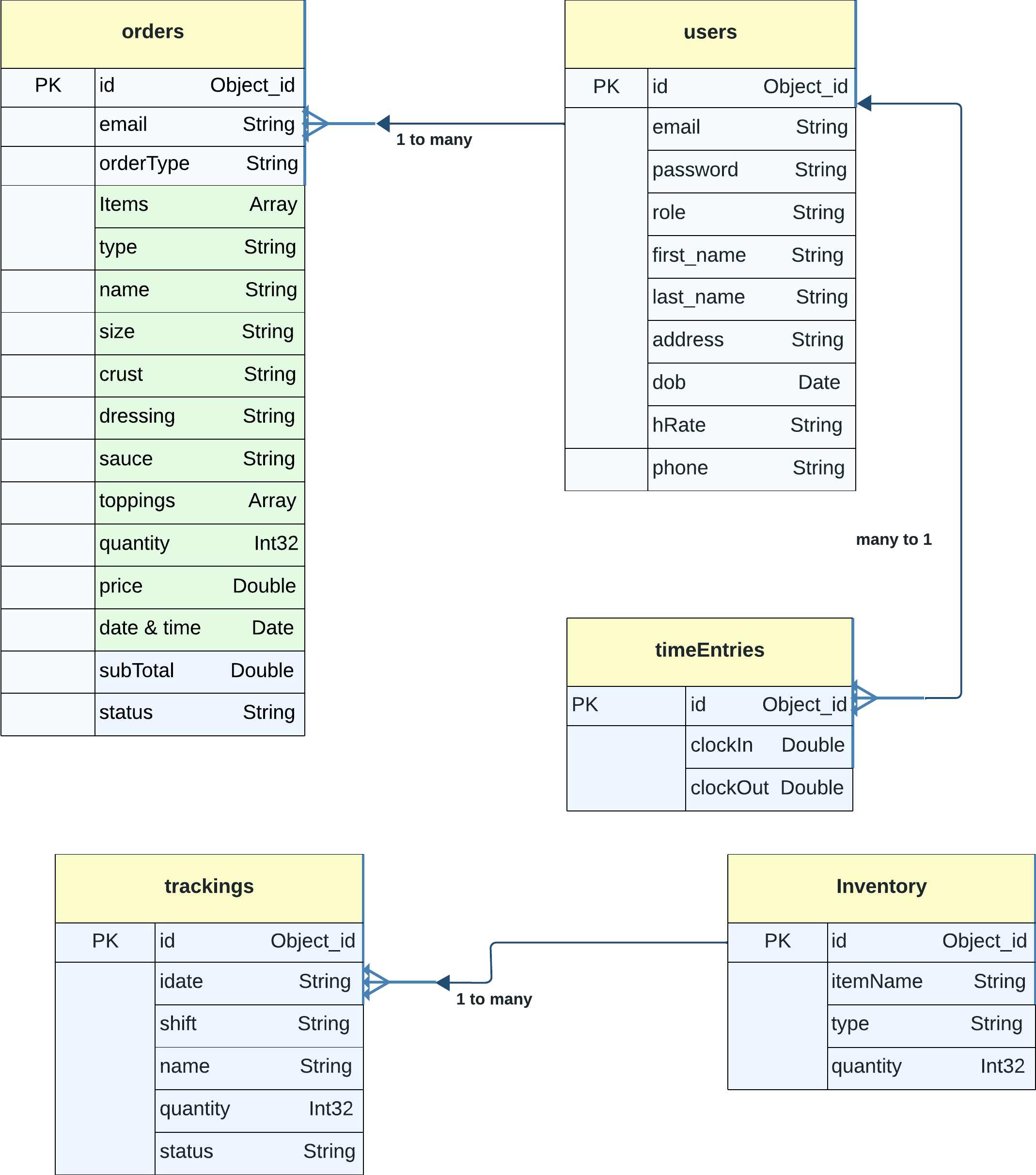
Sequence Diagram for Payment (UC-19)



## DATA FLOW DIAGRAM



## DATABASE DESIGN



## APPLICATION PROGRAM INTERFACES

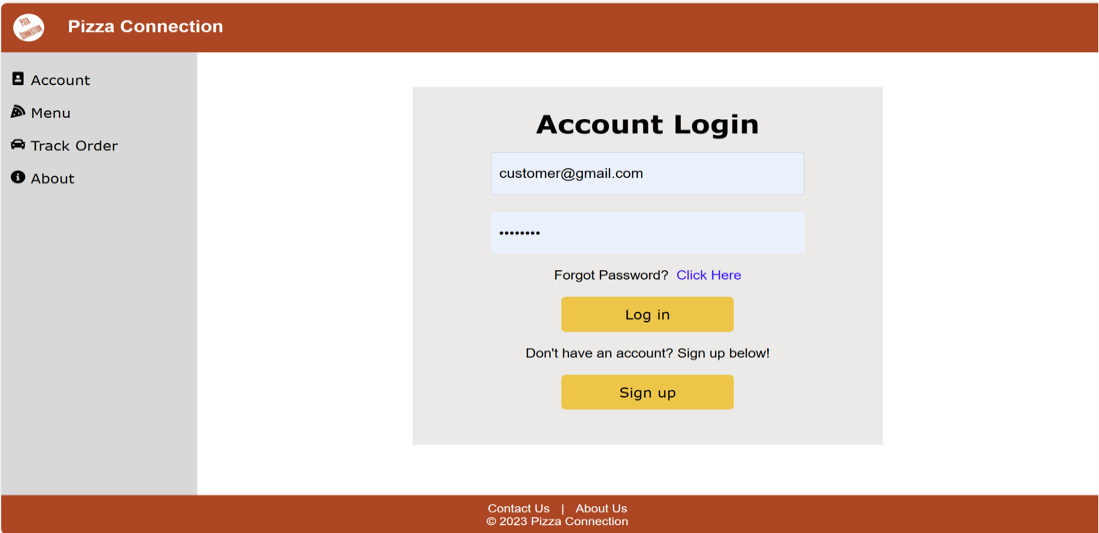
A diagram of a software application

Description automatically generated

Above is an example of REST API design. A REST (Representational State Transfer) API, or simply a RESTful API, is a set of rules and conventions for building and interacting with web services.

## USER INTERFACE DESIGN

Log In:



Customer Dashboard:

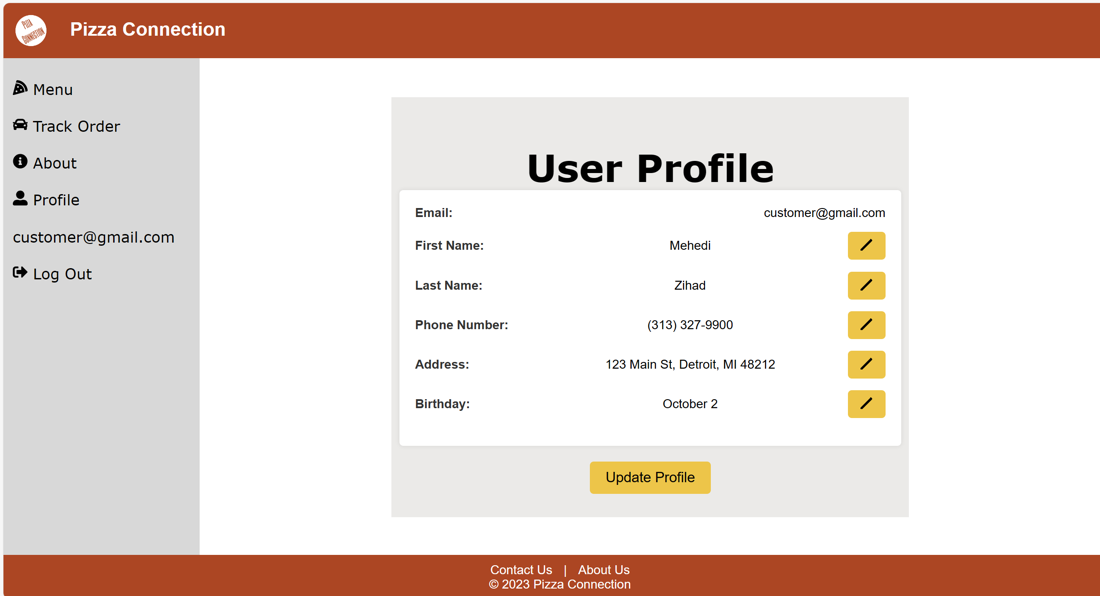


Management Page:

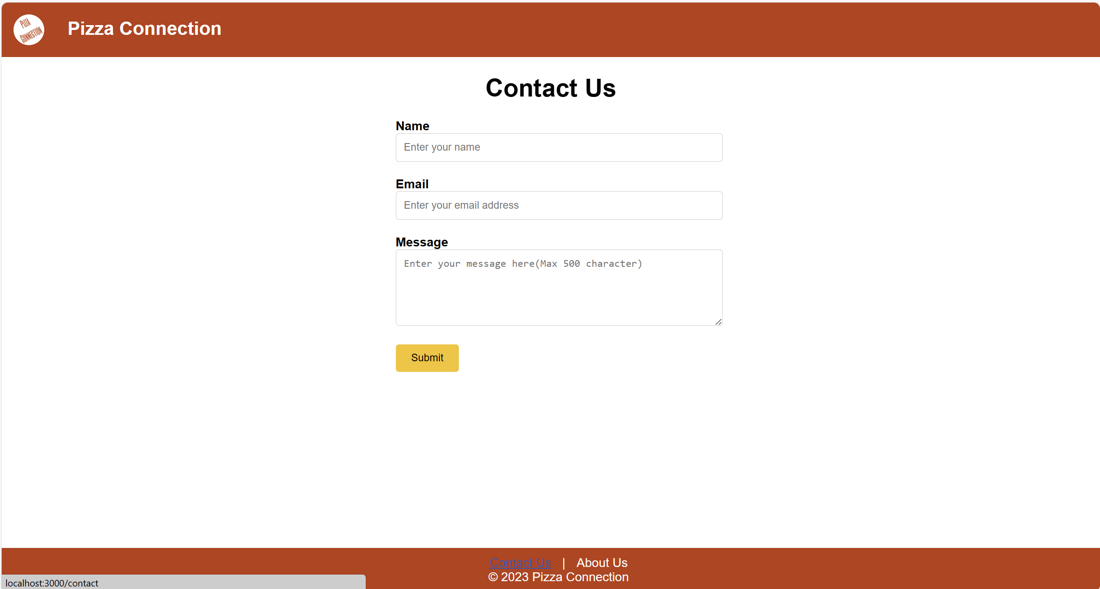
A pizzas with different toppings

Description automatically generated

Profile:



Contact:



# PRODUCT DESIGN SPECIFICATION APPROVAL

The undersigned acknowledge they have reviewed the Pizza Connection **Product Design Specification** document and agree with the approach it presents. Any changes to this Requirements Definition will be coordinated with and approved by the undersigned or their designated representatives.

|  |  |  |  |
| --- | --- | --- | --- |
| Signature: |  | Date: |  |
| Print Name: | Seyed Ziae Mousavi Mojab |  |  |
| Title: | Professor |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Signature: |  | Date: |  |
| Print Name: | Saeed Bakhshan |  |  |
| Title: | Graduate Teaching Assistant |  |  |

## Appendix A: References

<https://app.diagrams.net/> Used for making the Sequence and Data Flow diagrams

## Appendix B: Key Terms

|  |  |
| --- | --- |
| **Term** | **Definition** |
| GTA | Graduate Teaching Assistant |
| JSON | JavaScript Object Notation |
| AWS | Amazon Web Services |
| DOB | Date of Birth |
| REST API | Representational State Transfer Application Programming Interface |
| User | Anyone involved in the pizza industry, whether they are a pizza customer, an employee, or a pizza business owner, can use the web application. |
| Admin | An administrator that will see certain parts of the application. |
| Time Clock | Employees can use the timeclock to clock in when they begin their work, take scheduled breaks, and clock out when they finish their work. The timeclock keeps a record of when employees start their work, take breaks, and work overtime, supplying a detailed account of their work time. |
| Sale Tracking | The owner can check pizza sales, track business expenses, and assess the profit of the business. |
| Inventory Management | The owner can input their pizza-making ingredients and maintain a record of the raw materials, effectively managing inventory. |
| Stripe | Third party payment processing service. Will be used for this site. |