Software Requirements

Specification

For

**Pizza Connection**

Version 1.2

Approved By

Saeed Bakhshan

December 5, 2023

Matthew Krol

Prepared By: William Esparza

Mehedi Zihad

Kanta Islam

# 

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Description** | **Comments** | **Author** |
| 09/29/23 | Version 1.0 | First Draft | Kanta Islam  Matthew Krol  Mehedi Zihad  William Esparza |
| 10/10/23 | Version 1.1 | Fixed sections based on feedback by GTA and professor |
| 12/05/23 | Version1.2 | Final |

# Document Approval

The following Software Requirements Specification has been accepted and approved by the following:

|  |  |  |  |
| --- | --- | --- | --- |
| **Signature** | **Printed Name** | **Title** | **Date** |
|  | Saeed Bakhshan | GTA |  |
|  | Matthew Krol | Team Lead / Client |  |
|  | William Esparza | UI Lead |  |
|  | Mehedi Zihad | Technical Lead |  |
|  | Kanta Islam | Documentation Lead |  |

Table of Contents

Revision History 2

[Document Approval 2](#_Toc448311303)

[1. Introduction 3](#_Toc1167101868)

[1.1 Purpose 4](#_Toc341155838)

[1.2 Scope 4](#_Toc363099166)

[1.3 Definitions, Acronyms, and Abbreviations 5](#_Toc1550029298)

[1.4 References 6](#_Toc1533937918)

[1.5 Overview 6](#_Toc1108600185)

[2. General Description 6](#_Toc562707788)

[2.1 Product Perspective 7](#_Toc645583349)

[2.2 Product Functions 7](#_Toc2064596305)

[2.3 User Characteristics 7](#_Toc299426816)

[2.4 General Constraints 8](#_Toc980099740)

[2.5 Assumptions and Dependencies 8](#_Toc922676421)

[3. Specific Requirements 8](#_Toc1682829989)

[3.1 External Interface Requirements 9](#_Toc1308835504)

[3.1.1 User Interfaces 9](#_Toc1410380474)

[3.1.2 Software Interfaces 11](#_Toc1260044077)

[3.1.3 Communications Interfaces 12](#_Toc1629303768)

[3.2 Functional Requirements 12](#_Toc1349223869)

[3.2.1 <Functional Requirement or Feature #1> 13](#_Toc1695211368)

[3.2.2 <Functional Requirement or Feature #2> 13](#_Toc1134855240)

[3.2.3 <Functional Requirement or Feature #3> 14](#_Toc788769072)

[3.2.4 <Functional Requirement or Feature #4> 14](#_Toc1562146173)

[3.2.5 <Functional Requirement or Feature #5> 15](#_Toc590189058)

[3.2.6 <Functional Requirement or Feature #6> 15](#_Toc1505087159)

[3.2.7 <Functional Requirement or Feature #7> 16](#_Toc1627982729)

[3.2.8 <Functional Requirement or Feature #8> 17](#_Toc775059426)

[3.2.9 <Functional Requirement or Feature #9> 17](#_Toc303086842)

[3.2.10 <Functional Requirement or Feature #10> 18](#_Toc425042744)

[3.2.11 <Functional Requirement or Feature #11> 18](#_Toc543420241)

[3.2.12 <Functional Requirement or Feature #12> 19](#_Toc481435692)

[3.3 Non-Functional Requirements 19](#_Toc1530478323)

[3.3.1 Performance 19](#_Toc2092688797)

[3.3.2 Reliability 20](#_Toc399770834)

[3.3.3 Availability 20](#_Toc1546115336)

[3.3.4 Security 20](#_Toc1677687463)

[3.3.5 Maintainability 20](#_Toc1120771458)

[3.3.6 Portability 21](#_Toc471757646)

[3.4 Design Constraints 21](#_Toc1925083418)

[3.5 Logical Database Requirements 23](#_Toc1700799029)

[3.6 Other Requirements 24](#_Toc1293860635)

[4. Analysis Models 25](#_Toc996459994)

[4.1 Data Flow Diagrams (DFD) 25](#_Toc728164799)

[A. Appendices 25](#_Toc1148750454)

[A.1 Appendix 1 25](#_Toc222270865)

# 1. Introduction

## Purpose

The purpose of the software requirements specification document for the Pizza Connection web application is to serve as a comprehensive, detailed, and structured reference guide that covers all aspects of the project. It aims to provide clarity and guidance to all project stakeholders, enabling accurate design, development, and deployment of the software. The primary goal is to ensure that the software engineers have the necessary information to design and implement the Pizza Connection web application accurately.

## 1.2 Scope

The software product is to be produced the Pizza Connection web application, which serves

as a comprehensive solution for the pizza industry, addressing the needs of both

pizza businesses and customers. It includes features for customer pizza orders,

Manage Employees, and manage Inventory, Time Clock, Order List making it a complete solution for all aspects of the pizza industry.

Easy Ordering Pizza:

The web application will have an interactive menu where customers can explore a variety of pizza options. They can customize their orders extensively, including choosing toppings, specifying the size of the pizza, and even making special requests. Once they have tailored their order to their liking, they can easily place it online, with options for delivery or pickup.

Employee Time Clock:

This tool allows pizza businesses to monitor employee work hours meticulously. It records when employees start and finish their shifts, providing accurate data for payroll processing. Additionally, it enables businesses to evaluate employee productivity and assess labor costs to make informed decisions.

Manage Employees:

The application assists pizza businesses in managing their workforce effectively. It includes features for scheduling shifts, tracking employee work hours, and assigning specific roles or tasks. Managers can easily create work schedules and ensure the right number of employees are available during peak hours, optimizing staff productivity.

Order List:

The application's order list feature provides a comprehensive overview of a pizza business's performance. It captures data on the number of pizzas sold, and other relevant order information. This information is presented through clear and customizable reports, allowing businesses to analyze their performance over time and make data-driven decisions to enhance sales strategies and profitability.

Manage Inventory:

This feature is designed to streamline how pizza businesses handle their ingredients and supplies. It offers a digital toolkit to keep track of available ingredients in real-time. When an ingredient is used to make a pizza, the system automatically updates the inventory, ensuring that businesses always know what they have on hand. This feature helps prevent running out of crucial ingredients and improves overall operational efficiency.

## 1.3 Definitions, Acronyms, and Abbreviations

|  |  |
| --- | --- |
| **Term** | **Definition** |
| GTA | Graduate Teaching Assistant |
| JSON | JavaScript Object Notation |
| DOB | Date of Birth |
| 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, providing a detailed account of their work activities. |
| Order List | The owner can monitor pizza order list, list of items, and make the order status in progress or completed. |
| Manage Inventory | The owner can input their pizza-making ingredients and maintain a record of the raw materials, effectively managing inventory. |

## 1.4 References

[1] IEEE Software Engineering Standards Committee, “IEEE Std 830-1998, IEEE Recommended  
Practice for Software Requirements Specifications,” October 20, 1998.

[2] MongoDB Atlas, “[MongoDB Atlas Trust Center | MongoDB](https://www.mongodb.com/products/platform/trust)” September 30, 2023.

## 1.5 Overview

The Software Requirements Specification (SRS) document outlines the detailed requirements for the Pizza Connection web application. It begins with an introduction that defines the purpose and scope of the document. The SRS is organized into sections, including a general description, specific requirements, analysis models, and appendices.

The general description provides an overview of the project, explaining its functions, user characteristics, constraints, and assumptions. The specific requirements section is the core of the document, presenting detailed requirements such as external interfaces, functional features, non-functional attributes, design constraints, and database requirements. Each requirement is documented with its unique ID, title, description, inputs, processing, outputs, error handling, dependencies, and priority.

Analysis models, specifically data flow diagrams, are included to visualize and clarify the requirements. Finally, appendices contain supplementary information that supports the SRS but is not considered part of the core requirements.

Overall, the SRS is structured to provide a comprehensive and well-organized reference guide for the development of the Pizza Connection web application, ensuring clarity and guidance for all project stakeholders.

# 2. General Description

These next few sections will outline a general description of our application. It will go through what the product perspective is, the product functions, the user characteristics, general constraints, and finally the assumptions and dependencies. These will cover why our team is doing this project, and what we hope to accomplish with doing this project.

## 2.1 Product Perspective

Pizza Connection compares to comparable products and/or projects in the market in that it provides an accessible user interface for customers of pizza businesses and comparable features such as customer account creation, order tracking, and online ordering. What other services in the same market do not offer that Pizza Connection will offer is a clean, modern, and user-friendly interface and will integrate customer and business functionalities in one application. These business functionalities include inventory management, employee time tracking, and sales tracking. No other product in the market offers this kind of comprehensive two in one service that makes it both convenient and accessible to pizza businesses and their customers.

## 2.2 Product Functions

**Customer Account Management** – Customers of pizza businesses will be able to create their own accounts to keep track of their purchases and receive coupons or discounts as offered by the businesses. This will employ login, signup, and authentication token functionalities to achieve this.

**Online Order Menu** – Pizza businesses can provide their menu of products they sell online for their customers to be able to place an order then the order will be available to view and process at the business end.

**Time Clock –** Employees can make use of the time clock to clock in and clock out to record the time worked as well as the days they worked.

**Manage Inventory** – Businesses can manage their stock of items by adding or removing what items they have as well as updating the number of items left in stock.

**Manage Employees** – Business owners or managers can add, edit or remove employees from the application so they can make use of the time clock feature and be able to receive orders from customers.

**Order List**– This feature will allow pizza business owners or managers to see how much of each item is ordering.

## 2.3 User Characteristics

The two classes of users that will be using this app will be pizza business employees and their customers. Both classes of users will need to be able to read and understand English as well as have internet access to access this application.

**Customers** – The customers will be looking to be able to easily navigate the site with no hassle placing their order. Some customers will also be wanting to know the nutritional value of the items they will be ordering to track their own nutrition. Lastly the customer will also want to know the allergy information for each item as some customers may be dealing with food allergies. All this information the customer will want to know and easily access it at any given time and place their orders without hassle.

**Employees** – The pizza business employees will want to reduce the hassle of clocking in and out and to be able to keep track of their time for proper compensation. The owner and/or manager(s) will want to have a tool where they can manage their business and menu in one spot for cohesiveness and ease of use. They will also want to maximize this application's usability to make a return on investment and help grow their business.

## 2.4 General Constraints

**Time Constraints** – Due to time constraints a fully functional payment processing feature will not be implemented as this will require implementation of various security features and checks to ensure customer credit card information remains private and secure. Instead, we will opt for a third-party service to fulfill this requirement.

**Financial Constraints –** Since this application is not being funded nor there is a revenue stream to support paid features, we will be limited with web hosting services as well as database hosting services, namely Mongo DB since that is the service we are utilizing. Due to this constraint, we are unable to implement a database server that allows for node connections of over 500 as well as for web hosting we will have to use a free web hosting service which may limit the amount of site visitors the application will be able to get as well as the amount content we can display.

## 2.5 Assumptions and Dependencies

**Mongo DB –** It is assumed that Mongo DB’s uptime will be at least 99% or higher and that it will be able to handle up to 500 node connections as advertised/stated on the Mongo DB site for the cluster tier we are utilizing – M0.

# 3. Specific Requirements

Pizza Connect is going to be divided into three main categories: external interface requirements, functional requirements, and nonfunctional requirements. The external interface will show off how the user is meant to interact with our project. These requirements are necessary because it is the core application of our project. Optional features are not essential for the app to work. These features will be added if time allows.

## 3.1 External Interface Requirements

### 3.1.1 User Interfaces

Our Pizza Connection web application has two user interfaces. They are customer user interface and business management user interface.

Customer user interface:

A screenshot of a computer

Description automatically generated

Figure 1: Customer Log In

A close up of a pizza

Description automatically generated

Figure 2: Customer Home Page

Business management user interface:

A screenshot of a dashboard

Description automatically generated

Figure 3: Management Dashboard

### 3.1.2 Software Interfaces

For a software interface, one crucial aspect to consider is browser compatibility. Seamless functionality across diverse web browsers is vital, considering the varying browsing preferences of users. Achieving browser compatibility entails thorough testing across a range of platforms and versions, including Chrome, Firefox, Safari, Internet Explorer, and more. Developers enhance compatibility by adopting industry-standard practices like utilizing web standards, CSS3, and HTML5. Additionally, techniques such as progressive enhancement, graceful degradation, and responsive design principles are employed to optimize compatibility and adaptability across a spectrum of screen sizes and resolutions.

In addition to browser compatibility, cross-platform functionality is equally important in defining external interface requirements, particularly for software interfaces. Ensuring compatibility across various operating systems like Windows, macOS, Linux, and mobile platforms such as iOS and Android area critical consideration. Rigorous testing and optimization across these platforms are essential to guarantee a seamless user experience, regardless of the device or system in use. This comprehensive approach significantly enhances the accessibility and usability of the software interface, expanding its utility to a broader audience. Prioritizing compatibility across both browsers and diverse platforms ensures a consistent, user-friendly experience, fostering widespread accessibility and maximizing the reach of the external interface.

### 3.1.3 Communications Interfaces

Integrating MongoDB Atlas into our web application is a fundamental aspect of developing an effective external interface with seamless communication capabilities. MongoDB Atlas is our chosen database solution, providing a reliable and scalable platform to manage our application's data efficiently. Communication with MongoDB Atlas occurs primarily through online network protocols, including HTTP and HTTPS, ensuring secure and efficient data transmission between our application and the database over the Internet.

Authentication and authorization mechanisms in MongoDB Atlas play a crucial role in securing our communication interface. MongoDB Atlas offers robust authentication methods, allowing us to implement secure user access controls. With role-based access control and authentication protocols, we can ensure that only authorized individuals can access and modify the data within our database. These mechanisms guarantee a secure environment for our application's interactions with MongoDB Atlas.

In terms of performance optimization, MongoDB Atlas offers a range of tools and features to enhance our communication interface. Through query optimization, indexing, and efficient data modeling, we can optimize the speed and responsiveness of our database queries. This ensures that our web application communicates with MongoDB Atlas in a highly efficient and streamlined manner, enhancing the overall user experience.

By integrating MongoDB Atlas into our external interface requirement and leveraging its capabilities in terms of network protocols, authentication, authorization, and performance optimization, we establish a secure, efficient, and scalable communication interface for our web application. This integration empowers us to manage our data effectively and maintain optimal performance levels while adhering to stringent security measures.

## 3.2 Functional Requirements

### 3.2.1 <Functional Requirement or Feature #1>

|  |  |
| --- | --- |
| **ID:** | FR 3.2.1 |
| **Title:** | Sign up |
| **Description:** | This will allow a user to sign up and make an account for the website. |
| **Input:** | Email, Password |
| **Processing:** | Check against the backend to make sure the user is not already a member, and the password is up to the requirements. |
| **Outputs:** | A successful message saying the user is signed up, and a redirect to the home page. |
| **Error Handling:** | This will show the user what the error they are facing is, such as an invalid email, or invalid password. |
| **Dependencies:** | FR 3.2.2, FR 3.2.3, FR 3.2.4, FR 3.2.5, FR 3.2.6, FR 3.2.7, FR 3.2.8, FR 3.2.9, FR 3.2.10 |
| **Priority:** | High priority. |

### 3.2.2 <Functional Requirement or Feature #2>

|  |  |
| --- | --- |
| **ID:** | FR 3.2.2 |
| **Title:** | Login |
| **Description:** | This will allow a user to login after they have created an account. |
| **Input:** | Email, Password |
| **Processing:** | Check against the backend to make sure the user is already a member, and the password is the same as the one they put in. |
| **Outputs:** | A successful message saying the user is login, and a redirect to the home page |
| **Error Handling:** | This will show the user what the error they are facing is, such as an invalid email, or an invalid password. |
| **Dependencies:** | FR 3.2.1, FR 3.2.3, FR 3.2.4, FR 3.2.5, FR 3.2.6, FR 3.2.7, FR 3.2.8, FR 3.2.9, FR 3.2.10 |
| **Priority:** | High priority. |

### 3.2.3 <Functional Requirement or Feature #3>

|  |  |
| --- | --- |
| **ID:** | FR 3.2.3 |
| **Title:** | Profile |
| **Description:** | This will display the user's information to them. Will also allow them to edit their information. |
| **Input:** | Name, Phone, Address, DOB |
| **Processing:** | This will update the back end with the latest information or display what current information the user has. |
| **Outputs:** | A successful message saying the user is login, and a redirect to the home page |
| **Error Handling:** | Display the user data that they have inputted. |
| **Dependencies:** | FR 3.2.2 |
| **Priority:** | High priority. |

### 3.2.4 <Functional Requirement or Feature #4>

|  |  |
| --- | --- |
| **ID:** | FR 3.2.4 |
| **Title:** | Pizza Menu |
| **Description:** | Allows a user to place an order. Customize users on the menu to the users liking. |
| **Input:** | Select items for an order. |
| **Processing:** | Add order to MongoDB when order is placed. |
| **Outputs:** | Display a recipe for the order. |
| **Error Handling:** | Make sure all items meet with the current requirements for each. For example, A pizza has a size, type, sauce. |
| **Dependencies:** | FR 3.2.2 |
| **Priority:** | High priority. |

### 3.2.5 <Functional Requirement or Feature #5>

|  |  |
| --- | --- |
| **ID:** | FR 3.2.5 |
| **Title:** | Employee Editing |
| **Description:** | This will allow a manager or owner to edit any user in their system. |
| **Input:** | First Name, Last Name, Phone, Address, DOB, Email, Role |
| **Processing:** | Pull up a specific employee to edit their information. Then upon editing the information it will check for errors before updating the database. |
| **Outputs:** | This will display information about the user they are dealing with. |
| **Error Handling:** | Make sure the DOB and Email are in the correct format. |
| **Dependencies:** | FR 3.2.2 |
| **Priority:** | High priority. |

### 3.2.6 <Functional Requirement or Feature #6>

|  |  |
| --- | --- |
| **ID:** | FR 3.2.6 |
| **Title:** | Employee Adding |
| **Description:** | This feature empowers managers or owners to modify the details of any user within their system. The editable fields include First Name, Last Name, Phone Number, Address, Date of Birth (DOB), Email, and Role. |
| **Input:** | First Name, Last Name, Phone, Address, DOB, Email, Password, Role.  Users can choose a specific employee whose information they want to edit. |
| **Processing:** | When a new user is added to the system, it checks for errors first.  If someone wants to change an employee's information, they can do that. But before saving the changes to the database, the system makes sure there are no mistakes in the updated info to keep things accurate and consistent. |
| **Outputs:** | During the editing process, the system keeps the current screen open and shows a success message. Additionally, it continuously displays pertinent information about the user being edited, providing a transparent view of the changes being made. |
| **Error Handling:** | To maintain data integrity, the system verifies that the Date of Birth (DOB) and Email are in the correct format, preventing any inconsistencies.  In addition, it will check the strength of the password to make sure it meets requirements. |
| **Dependencies:** | FR 3.2.2 |
| **Priority:** | High priority. |

### 3.2.7 <Functional Requirement or Feature #7>

|  |  |
| --- | --- |
| **ID:** | FR 3.2.7 |
| **Title:** | Forgot Password |
| **Description:** | Allows users to update their password. |
| **Input:** | Password, Confirm password |
| **Processing:** | Will update the user's password on the backend. |
| **Outputs:** | Display a success message saying the password was updated, then take them to the home page. |
| **Error Handling:** | Make sure the password meets the minimum requirements. |
| **Dependencies:** | FR 3.2.2 |
| **Priority:** | High priority. |

### 3.2.8 <Functional Requirement or Feature #8>

|  |  |
| --- | --- |
| **ID:** | FR 3.2.8 |
| **Title:** | Reset Password |
| **Description:** | If user forgot their password, they are allowed to reset the password by email link |
| **Input:** | Email |
| **Processing:** | User will receive the email link to resets the password |
| **Outputs:** | Password reset link send to your email |
| **Error Handling:** | Make sure entered the correct email |
| **Dependencies:** | 3.2.7 |
| **Priority:** | High priority |

### 3.2.9 <Functional Requirement or Feature #9>

|  |  |
| --- | --- |
| **ID:** | FR 3.2.9 |
| **Title:** | Time Clock |
| **Description:** | This feature enables employees to record their working hours. |
| **Input:** | No input (Just click on the clock in or clock out button) |
| **Processing:** | When the employee clicks on the clock in or clock out the database automatically records the time. |
| **Outputs:** | Will display out of a successful clock in or out and track previous time clocks. |
| **Error Handling:** | Make sure clock in / clock out on time |
| **Dependencies:** | FR 3.2.2 |
| **Priority:** | Medium priority. |

### 3.2.10 <Functional Requirement or Feature #10>

|  |  |
| --- | --- |
| **ID:** | FR 3.2.10 |
| **Title:** | Inventory View |
| **Description:** | Allows the admin to view, search and sort the current inventory. |
| **Input:** | Inventory table from the database. User input in the search bar and user action on clicking the arrows to sort the list. |
| **Processing:** | Populates the table view with the data from the inventory table. For searching it searches for relevant terms from user input. For sorting it sorts the table in ascending or descending order. |
| **Outputs:** | Will display the inventory in a list/table with pagination (if over 10 items in the list). Sorting will display table contents organized in ascending or descending order. Searching displays only the relevant inventory items in the list. |
| **Error Handling:** | Error message out put that the inventory cannot be displayed. |
| **Dependencies:** | FR 3.2.2 |
| **Priority:** | High priority. |

### 3.2.11 <Functional Requirement or Feature #11>

|  |  |
| --- | --- |
| **ID:** | FR 3.2.11 |
| **Title:** | Inventory Adding |
| **Description:** | Allows the admin to add items to the inventory. |
| **Input:** | Item details: name, category, quantity and unit type. |
| **Processing:** | Upon form submission it creates a new entry in the inventory table in the database. |
| **Outputs:** | Displayed on the inventory list on the pages that display it. |
| **Error Handling:** | All fields required message. If the form cannot connect to DB an error appears informing that the item could not be added. |
| **Dependencies:** | FR 3.2.2 |
| **Priority:** | High priority. |

### 3.2.12 <Functional Requirement or Feature #12>

|  |  |
| --- | --- |
| **ID:** | FR 3.2.12 |
| **Title:** | Inventory Editing |
| **Description:** | Allows the admin to edit any item in the inventory list. |
| **Input:** | Modified details: name, category, quantity and unit type. |
| **Processing:** | Changes/modifications are saved to the inventory list and updated as well in the db and the inventory view page. |
| **Outputs:** | Displayed on the inventory list on the pages that display it. |
| **Error Handling:** | Displays message that inventory could not be updated. |
| **Dependencies:** | FR 3.2.2 |
| **Priority:** | High priority. |

### 3.2.13 <Functional Requirement or Feature #13>

|  |  |
| --- | --- |
| **ID:** | FR 3.2.13 |
| **Title:** | Inventory Tracking |
| **Description:** | Allows the admin to view and create tracking records for a particular inventory item. Also, the user can sort and search the tracking list for inventory items. This feature is used to track and modify inventory quantity for a particular item. |
| **Input:** | Filled out add tracking inventory form including quantity and the item being tracked. Search bar input and sort button click. |
| **Processing:** | An inventory tracking record for the form sumission is created in the tracking table in the db and displayed in the tracking inventory view. The inventory table is updated with the quantity being added/used up. |
| **Outputs:** | Displays tracking record in the tracking inventory list and inventory list is updated with the new quantity (increased or decreased) for that particular inventory item. |
| **Error Handling:** | Displays message depending on action user is trying to take. If creating a tracking record, then an error message appears that the record could not be added. Also gives an error message if a user attempts to create a tracking entry if it lowers the quantity of a particular item in the inventory below zero. |
| **Dependencies:** | FR 3.2.2 |
| **Priority:** | Medium priority. |

### 3.2.14 <Functional Requirement or Feature #14>

|  |  |
| --- | --- |
| **ID:** | FR 3.2.14 |
| **Title:** | Order Track |
| **Description:** | Within the Sales Tracking System, managers and owners can easily retrieve the purchase history of all sales transactions. This feature empowers them with a holistic view of their customers' buying behaviors and transaction patterns.  To facilitate meaningful analysis, the system allows users to sort and filter sales data. This functionality assists in organizing data based on various criteria such as date, product, customer, or sales representative. This feature helps in identifying trends and patterns within the sales data. |
| **Input:** | No Input |
| **Processing:** | Will pull up the purchase history from the orders table to be able to see all the purchase history |
| **Outputs:** | Will display the purchase history on a nice, neat table. |
| **Error Handling:** | When pulling data from the database, we will ensure that the data is accurate. |
| **Dependencies:** | FR 3.2.2 |
| **Priority:** | Medium priority. |

### 3.2.15 <Functional Requirement or Feature #15>

|  |  |
| --- | --- |
| **ID:** | FR 3.2.15 |
| **Title:** | Payment |
| **Description:** | This feature enables customers to buy items from our website. |
| **Input:** | Payment information from the user |
| **Processing:** | The system processes the payment, and if it goes through successfully, it transfers money from the customer's account to the business's account. |
| **Outputs:** | Will display if payment is successful or unsuccessful. |
| **Error Handling:** | The system ensures that the card information consists of only numbers and that the date includes only the month and year. Additionally, it checks if the card information is accurate and not expired to enhance security. |
| **Dependencies:** | FR 3.2.2, FR 3.2.4 |
| **Priority:** | Medium priority. |

## 

## 3.3 Non-Functional Requirements

### 3.3.1 Performance

The website’s performance needs to be fast. We should have the page load fast to catch the user’s attention right away. This can be done in the following ways. We will be striving to make the website as fast as possible. This can be done in different ways by not using huge images, therefore, it has less to load. In addition, we will make sure that the user knows what is aware of the current state of the application with prompts of success or error failures. This way there is no doubt in a user’s mind about what they should do next. This will make our website more desirable than competitors'.

### 3.3.2 Reliability

Our webpage will exhibit high reliability in the sense of uptime and high performance, but also with information being up to date, factual, an exact representation of what is collected in the database, and secure. Our chosen hosting platform Mongo DB, plays a critical role in achieving this goal. We will implement security measures, regular maintenance, and a comprehensive disaster recovery plan to further enhance MongoDB's reliability in our web application.

### 3.3.3 Availability

Our Pizza Connection web application is committed to maintaining an exceptional level of availability, guaranteeing uninterrupted access and seamless usability for our users. The application will be accessible and operational 24 hours a day, 7 days a week, ensuring that users can enjoy our services whenever they desire. In cases where downtime is necessary for planned maintenance or updates, we will provide transparent communication by prominently displaying a message on the homepage at least 48 hours (about 2 days) in advance, clearly outlining the schedule and reason for the downtime. We aim to have our application available with little downtime, minimizing any outages that may occur.

### 3.3.4 Security

This application will be designed to keep the customer’s privacy and security in mind. We will ask only for information that is necessary to complete and fulfill a customer’s order. Security measures will be implemented to safeguard against accidental or malicious access, use, modification, destruction, or disclosure of sensitive data. These security measures will contribute to the resilience and trustworthiness of our Pizza Connection web application, fostering a secure environment for both the application and our users.

### 3.3.5 Maintainability

Maintaining the Pizza Connection software is an ongoing commitment to ensuring its long-term stability and reliability. We follow certain rules and standards when writing our code to make sure it is neat and consistent. The application will be designed so that in the future any additions to the application can be added easily without compromising the integrity and functionality of the existing components. A user manual will be written and provided as well to explain the ins and outs of our application and any issues that could come up and how to troubleshoot those issues. We aim for our users to have a reliable and easy-to-use experience.

### 3.3.6 Portability

The system will be accessible on modern web browsers, regardless of operating system. This includes operating systems such as Windows, macOS, Linux, iOS, and Android. The application will be compatible across all these systems and will be tested for any responsiveness and user interface issues. The system will also be cross-browser compatible, ensuring the web application is compatible with all major web browsers. This includes browsers such as Google Chrome, Mozilla Firefox, Safar, and Microsoft Edge. The web application will also be portable across devices, including tablets, desktops, laptops, and mobile phones. The desktops and laptops will be able to access the web application across multiple browsers, and all mobile devices can access the application from any modern browser mobile application. This site will have no mobile application component, but each modern browser application will be tested for responsiveness and ensure a seamless user experience across multiple screen sizes.

## 3.4 Design Constraints

The Pizza Connection software project has some important rules and limits that we need to follow. These rules come from industry standards, company policies, hardware restrictions, and the technology we use. Let us look at these limits in a straightforward way:

As our application expands and complexity increases, achieving top-tier performance can be demanding. We might need to implement advanced techniques like code splitting, lazy loading, and performance profiling to attain high-speed performance. These optimizations often require significant resources and ongoing maintenance.

Web applications rely heavily on internet connectivity. In regions with unreliable or limited internet access, certain aspects of our website may not function correctly. This limitation can negatively impact user experience, particularly for users in areas with poor internet infrastructure.

While Node.js and Express.js offer excellent performance, ensuring seamless scalability as our website grows can be intricate. Effective scaling may entail the deployment of load balancing, caching, and horizontal scaling strategies to handle surges in traffic. Failing to plan for scalability can result in performance bottlenecks during traffic spikes.

Integrating external tools and libraries with Node.js and Express.js can be complex. Ensuring that these components work harmoniously and do not introduce conflicts or security vulnerabilities requires ongoing attention and maintenance.

Security is paramount and maintaining the security of our Node.js and Express.js stack is critical. It involves keeping dependencies up to date, promptly addressing security vulnerabilities, and implementing robust security practices to safeguard our application from potential threats and breaches.

Operating a website, especially one with substantial traffic and complex functionality, may demand substantial computing resources. Effective planning for robust hosting infrastructure, efficient resource management, and diligent monitoring are essential for maintaining optimal performance. Neglecting resource considerations can lead to sluggish response times and service disruptions.

Due to a free MongoDB Atlas plan, the project faces a database size limit of 500 MB. Complying with these constraints requires careful database schema design and efficient data storage management.

The Pizza Connection project omits a mobile app, limiting user interaction to the web browser on their device. This limitation affects mobile usability and accessibility.

The application refrains from storing sensitive or private user information, preventing data storage and data retrieval services for future use. This approach is aligned with strict data privacy and security principles.

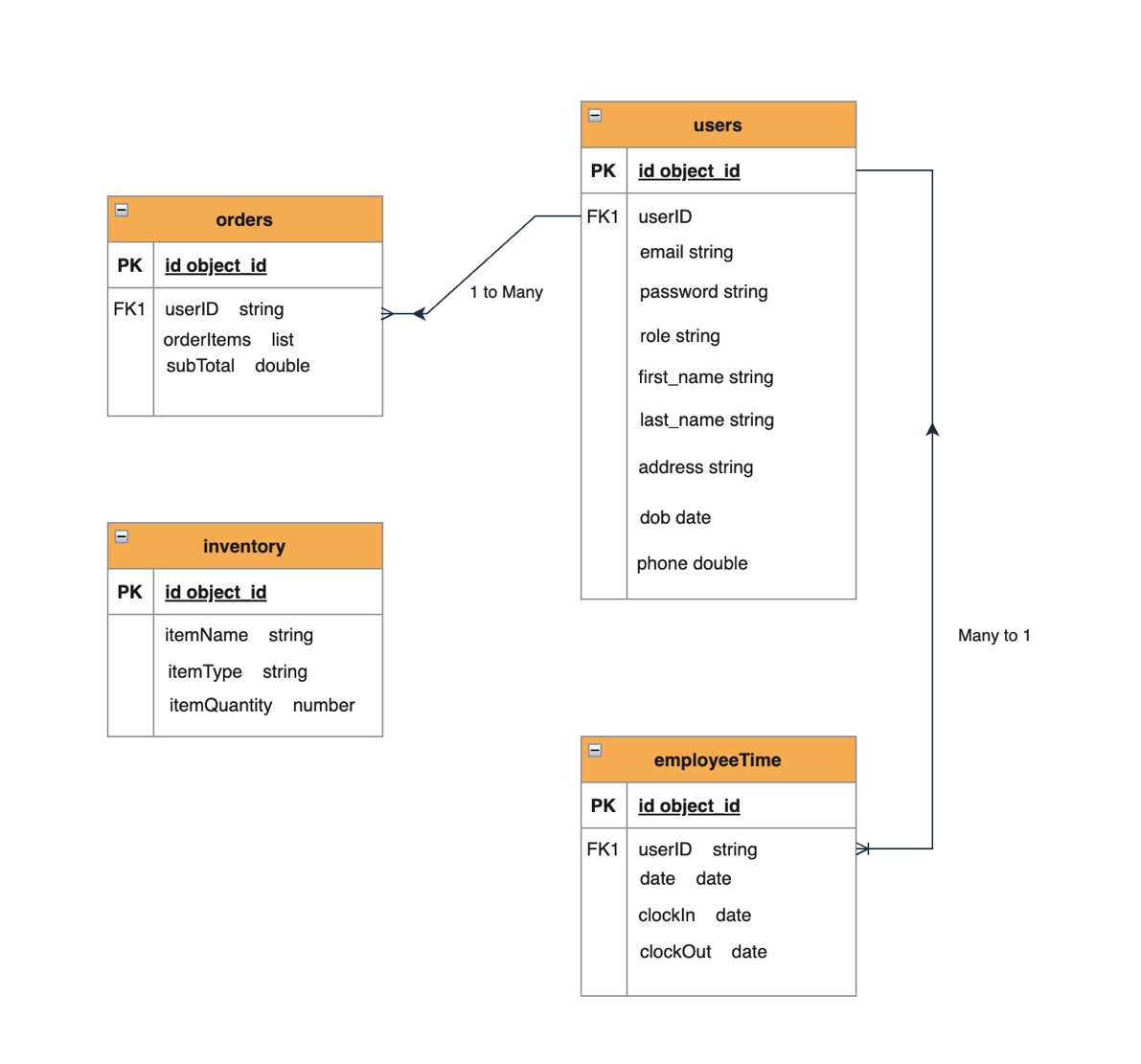
Although the project incorporates an employee time clock feature, it does not require additional external hardware. The possible inclusion of an external time clock machine is not part of the project design.

The Pizza Connection application features three user roles Owner or Manager, Employee, and Customer (General User). Owners and managers have administrative control, granting them authority to elevate general users to employee status and make important business decisions. Employees have standard user permissions, while customers can place orders and explore menu options. This structure ensures effective management of pizza businesses while catering to the needs of different users.

Possible issues with this system include admin misuse, permission confusion, security risks, employee role clarity, customer experience, scalability challenges, and maintenance complexity. Careful planning and clear definitions are essential to address these concerns.

## 3.5 Logical Database Requirements

MongoDB Atlas will be used for this project. The format that we are using in the database will be JSON. This will allow us to easily track data between the backend and the frontend. Password will be encrypted with bcrypt using salt and hash to make it extra secure. The type of data that is going to be used for users is everything will be kept in a JSON format. All the user’s information will be on one table. The second table will be for the order data from the pizza menu. This information will also be in JSON format. It will help keep everything consistent and usable for everyone. The third table will be for the employee time clock. This will link to the user table for information on the user but will keep track of shifts the employee has worked. MongoDB Atlas is encrypted by default so any information pulled from there will be encrypted in addition to the us encrypting the password upon uploading as well. There will be a fourth table included which will display inventory for the store. This information will also be in JSON format because it makes it easy to pull that data out of the database. All but the inventory data will be kept until the owner of the database decides to delete the information.

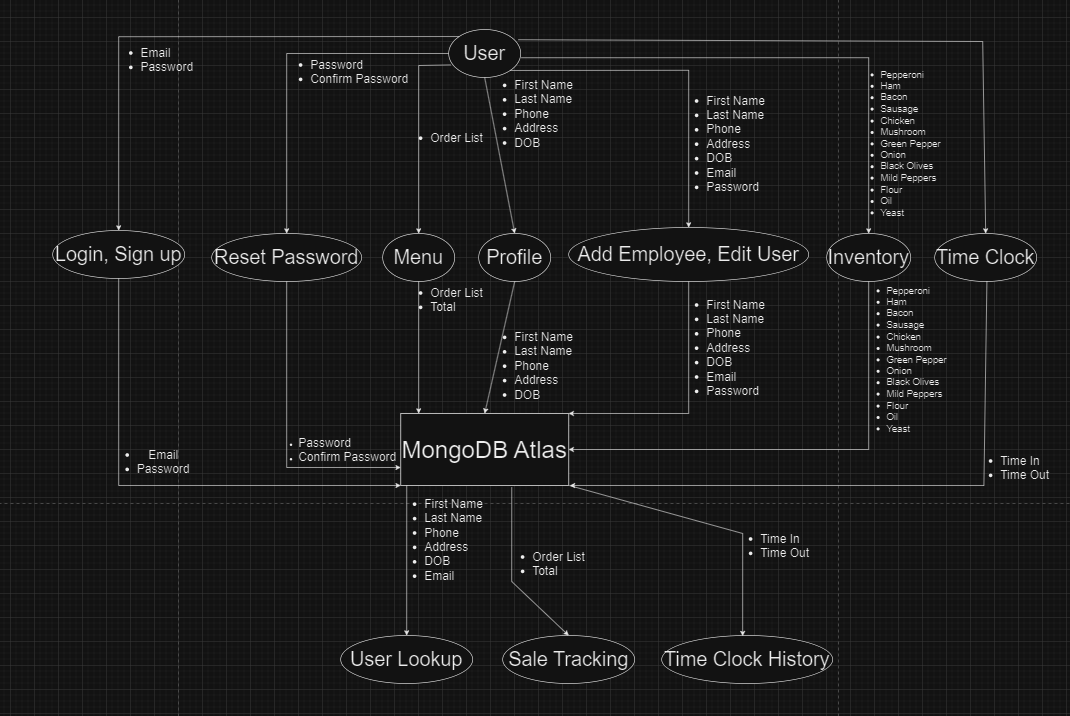


## 3.6 Other Requirements

As of this design stage, no further requirements have been identified for the Pizza Connection web application.

# 4. Analysis Models

## 4.1 Data Flow Diagrams (DFD)



# A. Appendices

The first appendix will be information on how our database is set up. Along with mock information contained inside.

## A.1 Appendix 1

This will display relevant information about our database. It will show how we store and keep information. This is all mock information which makes it safe to share with others. This is the Employee/User section of the database to show what information we require so far up sign up. These screenshots are from MongoDB Atlas.

