ADORAPLAN

A Eucharistic Adoration Planner

ABSTRACT
ENHANCE THE MANAGEMENT AND ENGAGEMENT OF RELIGIOUS COMMUNITIES THROUGH STREAMLINED TECHNOLOGICAL SOLUTIONS

Presented by: Bria Tran
CSC 3150 Systems Design

Table of Contents

1. Executive Summary

This document provides a comprehensive overview of AdoraPla, the Adoration Scheduler App, a digital solution designed to enhance scheduling adoration time slots. The primary audience for this application includes the app development team, St. Pius X Parish, church administrators, parishioners, and donors. The development team consists of software engineers, UX/UI designers, and managers who deliver high-quality systems tailored to the needs of religious organizations. Our customers are primarily St. Pius X Parish and their leadership and administrative staff. They require a reliable system that can handle a growing number of users, manage time slot reservations, and provide clear communication channels between parishioners.

What We Have Done So Far:

Presently, we have completed the initial design phase of the Adoration Scheduler, and includes:

1. Requirement Gathering
    a. engaged with stakeholders, church administrators, and parishioners to understand their needs
2. System Design
    a. developed detailed use cases, class diagrams, and workflow processes

Where the Project Is Going:

From now on, the project is starting to move into its final stages of development and testing, including:

1. Finalizing Features
    a. Complete the implementation of all features and report to administrators
2. Testing
    a. Conduct extensive system testing to ensure security, reliability, and user satisfaction

What This Document Covers:

This document provides a description of the design specifications of AdoraPlan and includes:

1. System Architecture
    a. Breakdown of the class attributes, methods, and other connections
2. Technical Requirements
    a. Description of the system's functions and user interactions, along with the infrastructure needed to support the system
3. Development Plan
    a. Roadmap outlines the leftover development tasks

What the Next Steps Are:

For the future of AdoraPlan, the project includes:

1. Testing
    a. Select a group of parishioners/users to test the system and get final feedback
2. Final Development
    a. Make any final adjustments to the code and integrate all the system components
3. Launch Preparation
    a. Final all deployment plans
4. Official Launch
    a. Roll out AdoraPlan for all parishioners and provide continued maintenance and support as needed

2. Introduction

AdoraPlan, the Adoration Scheduler App, is a transformative solution designed to simplify the coordination of Eucharistic Veneration time slots within the St. Pius X Parish Community and potentially across the Archdiocese of Seattle. Key features of the app include easy sign-up and management of adoration slots, automated notifications for scheduled times and changes, and the ability to request substitutes or trade times when needed. The app promotes participation in adoration by providing easy-to-use interfaces via mobile devices. In summary, AdoraPlan represents a significant advancement in parish administration and will strengthen the church community of St. Pius X. This document is intended for developers who are just joining the project or need a refresher. If more detail is required, please refer to the System Proposal document.

2.1. Problem Statement / Project Vision

The client for this project is St. Pius X Parish Community. Our task is to develop an Adoration Scheduler App to address the parish's challenge of coordinating Eucharistic veneration time slots. Previously, parishioners registered for adoration using a paper sign-up sheet posted at the chapel's entrance. Individuals must also find substitutes and arrange trades when they cannot fulfill their commitment, which has proven challenging. Continuous presence with the host is imperative for Eucharistic veneration, and the absence of attendees can lead to security concerns and diminish the sanity of the space. The system is being built to address these coordination issues and improve participation in Eucarhistic veneration. The expected benefits include increased participation, better organization, and heightened security of the adoration chapel.

The project scope includes a user-friendly interface accessible via mobile devices, with a centralized database for tracking adoration schedules. This solution must ensure integration with existing parish protocols and compliance with data protection regulations. For the most part, the development of AdoraPlan will involve close collaboration with the parish and other stakeholders without extending to broader needs.

2.2. System Capabilities

This section briefly summarizes the functional requirements within AdoraPlan. For more detail, please refer to the System Proposal Document:

1. Download App (Functional Requirement 1 - Use Case ID: 6)
   a. Users can download the app on both iOS and Android mobile devices
2. User Account Creation (Functional Requirement 2 - Use Case ID: 1)
   a. The system prompts the user to create a login based on whether they are a new or existing user
3. User Login (Functional Requirement 3 - Use Case IDs: 8, 13, 14, 20)
   a. System will save login information of existing users and allow users to recover login information in case they have forgotten
4. User Profile and Account Management (Functional Requirement 4 - Use Case IDs: 2, 10, 15)
   a. Users can access with profile information and make updates to their personal information stored on the system database
5. Share Contact Information (Functional Requirement 5 - Use Case ID: 3)
   a. Users can choose what information to share with other users of the application
6. Time-Slot Reservation (Functional Requirement 6 - Use Case IDs:4, 16, 17, 18, 19)
   a. Users can choose to reserve, trade, drop, or cancel a time slot selected
7. Q&A General Chat (Functional Requirement 7 - Use Case IDs: 5, 7)
   a. System includes an open Q&A chat that allows all users of the application to communicate with each other and receive group notifications
8. Event Creation and Sign-Up (Functional Requirement 8 - Use Case ID: 7)

   a. Users can create new events that require volunteers and post them on the general calendar so other parishioners can sign up

9. Calendar (Functional Requirement 9 - Use Case IDs: 11, 12)
   a. System allows the user to view the General calendar (which has all the open time slots or events) or their Personal calendar (which has all the time slots or events they are participating in
10. Generate Report (Functional Requirement 10 - Use Case ID: 9)
    a. System logs all user interaction with the app and generates a report for the Systems Administrator routinely

2.3. Non-functional Requirements and Design Constraints

Constraints:

1. Budget
   a. Project must adhere to a determined budget, influenced by the amount of donations from donors, which may limit development scope and resource allocation
2. Regulatory Compliance
   a. App must comply with data protection laws such as GDPR and FLSA, as regular compliance checks will be implemented during and after development
3. Resource Availability
   a. Agile methodologies and workflow optimization are recommended to mitigate the limited availability of funds for technological infrastructure
4. Capacity
   a. App must be able to handle at least 200 years, with scalability to accommodate future growth

Non-Functional Requirements:

1. Plaftorm Compatlilibty
   a. App must run on both iOS and Android devices, therefore developed using both Xcode and Android studio
2. Reliability
   a. App should achieve at least 99.9% uptime
3. Security
   a. Passwords must be blurred and only show one character at a time, along with personal data access restricted to only authorized personnel
4. Response Time
   a. Interface response time must be under 2 seconds
5. Acccesiblitly
   a. User interface should allow screen magnification and display lonrger0than-normal font sizes
6. Data backup
   a. Backup storage for user personal data is required
7. Distribution
   a. App must be distributed on both the Apple App Store and Google Play Store

2.4. System Evolution

This section of the document describes the possible functional or non-functional changes to future versions of AdoraPlan.

1. Version 2 Changes

There are two changes to AdoraPlan for the version. A few ideas that could be implemented are real-time notifications and automated substitute finders.

1.1. Real-Time Notifications
- 1.1.1. Send real-time notifications to users for schedule changes, substitution requirements, or open events/time slots
- 1.1.2. System detects a schedule change, upcoming adoration slot, etc
- 1.1.3. System sends a push notification to the affected parishioners
- 1.1.4. Parishioners receive notification and can take necessary actions

1.2. Automated Substitute Finders
- 1.2.1. Allows users to automatically find a substitute when they can't make it to their adoration slot
- 1.2.2. Parionser marks their time as needing a sub
- 1.2.3. System sends notifications to all other parishioners
- 1.2.4. An available parishioner accepts the request
- 1.2.5. System updates the schedule and notifies both parties

2. Version 3 and beyond Changes

For the Version Three adjustments of AdoraPlan, a few suggestions that should be implemented are personal barcode scanners and multilingual support.

2.1. Personal Barcode Scanners
- 2.1.1. Provide a function that provides a barcode linked to a doorlock scanner for security purposes
- 2.1.2. The system will be linked to a door lock that uses barcodes to identify authorized personal
- 2.1.3. Each user will be assigned a barcode and can scan their code via the app to unlock the chapel door
- 2.1.4. Increase security and identification of authorized adoration participants

2.2. Multipangual Support
- 2.2.1. Provide an option to switch to a Spanish version of the app because St. Pius X has a large Hispanic community
- 2.2.2. User selects the preferred language
- 2.2.3. System translates the interface and content into Spanish
- 2.2.4. User can then interact with the app in their preferred language

2.5. Document Outline

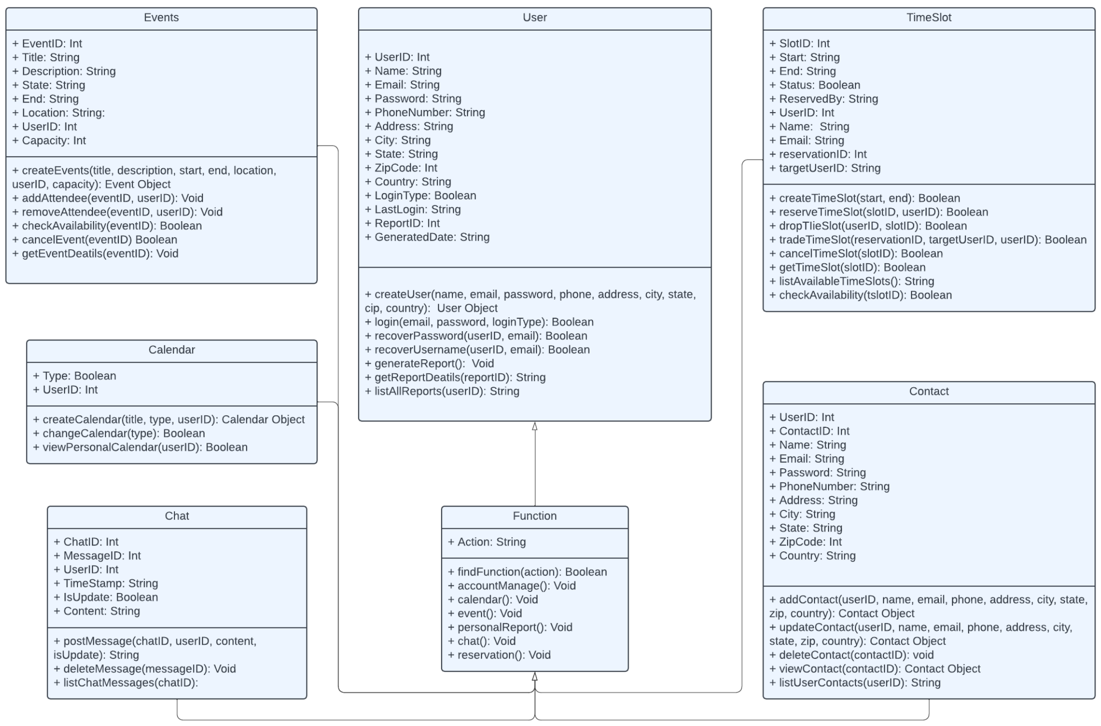
This section of the System's Specifications document outlines the development of AdoraPlan, the Adoration Scheduler App, and is organized into several sections. Below is an overview of the roadmap for achieving the project development objectives.

3. Structural Model

3.1. Model Introduction

The following section summarizes the class diagrams and how their metadata will interact. Using the Unified Modeling Language (UML) class models, the overview will depict the system's architecture with Class Diagrams. It emphasizes the structure of the class declarations and outlines the class methods for the Metadata section. It offers a more complete explanation of the operations and logic for class behaviors.

3.2. Class Diagrams


Events
+ EventID: Int
+ Title: String
+ Description: String
+ State: String
+ End: String
+ Location: String:
+ UserID: Int
+ Capacity: Int
+ createEvents(title, description, start, end, location, userID, capacity): Event Object
+ addAttendee(eventID, userID): Void
+ removeAttendee(eventID, userID): Void
+ checkAvailability(eventID): Boolean
+ cancelEvent(eventID) Boolean
+ getEventDeatils(eventID): Void
User
+ UserID: Int
+ Name: String
+ Email: String
+ Password: String
+ PhoneNumber: String
+ Address: String
+ City: String
+ State: String
+ ZipCode: Int
+ Country: String
+ LoginType: Boolean
+ LastLogin: String
+ ReportID: Int
+ GeneratedDate: String
+ createUser(name, email, password, phone, address, city, state, cip, country): User Object
+ login(email, password, loginType): Boolean
+ recoverPassword(userID, email): Boolean
+ recoverUsername(userID, email): Boolean
+ generateReport(): Void
+ getReportDeatils(reportID): String
+ listAllReports(userID): String
TimeSlot
+ SlotID: Int
+ Start: String
+ End: String
+ Status: Boolean
+ ReservedBy: String
+ UserID: Int
+ Name: String
+ Email: String
+ reservationID: Int
+ targetUserID: String
+ createTimeSlot(start, end): Boolean
+ reserveTimeSlot(slotID, userID): Boolean
+ dropTlieSlot(userID, slotID): Boolean
+ tradeTimeSlot(reservationID, targetUserID, userID): Boolean
+ cancelTimeSlot(slotID): Boolean
+ getTimeSlot(slotID): Boolean
+ listAvailableTimeSlots(): String
+ checkAvailability(tslotID): Boolean
Calendar
+ Type: Boolean
+ UserID: Int
+ createCalendar(title, type, userID): Calendar Object
+ changeCalendar(type): Boolean
+ viewPersonalCalendar(userID): Boolean
Contact
+ UserID: Int
+ ContactID: Int
+ Name: String
+ Email: String
+ Password: String
+ PhoneNumber: String
+ Address: String
+ City: String
+ State: String
+ ZipCode: Int
+ Country: String
+ addContact(userID, name, email, phone, address, city, state, zip, country): Contact Object
+ updateContact(userID, name, email, phone, address, city, state, zip, country): Contact Object
+ deleteContact(contactID): void
+ viewContact(contactID): Contact Object
+ listUserContacts(userID): String
Chat
+ ChatID: Int
+ MessageID: Int
+ UserID: Int
+ TimeStamp: String
+ IsUpdate: Boolean
+ Content: String
+ postMessage(chatID, userID, content, isUpdate): String
+ deleteMessage(messageID): Void
+ listChatMessages(chatID):
Function
+ Action: String
+ findFunction(action): Boolean
+ accountManage(): Void
+ calendar(): Void
+ event(): Void
+ personalReport(): Void
+ chat(): Void
+ reservation(): Void

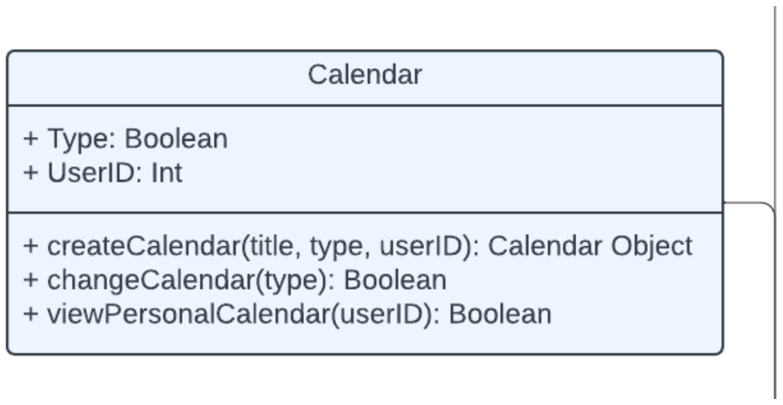
AdoraPlan’s Class Diagram Drawn In LucidChart

3.3. Metadata

This section of the document further describes each class's data attributes and operations and provides a brief overview of each class’s description. Below is the overview of each class in alphabetical order:

1. Calendar - page
2. Chat - page
3. Contact - page
4. Events - page
5. Function - page
6. TimeSlot - page
7. User - page

3.3.1 Calendar


Calendar
+ Type: Boolean
+ UserID: Int
+ createCalendar(title, type, userID): Calendar Object
+ changeCalendar(type): Boolean
+ viewPersonalCalendar(userID): Boolean

Description: Outlines the functions and properties of a Calendar object to create a calendar with the specified type

Visibility: Public

Is Abstract: No

Additional Information:

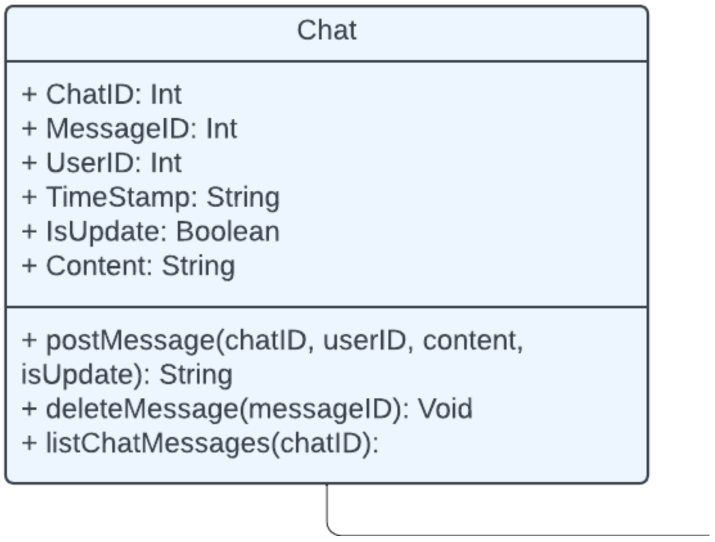
Attributes

Name	Description	Read Only	Multiplicity
Type	Type of Calendar User chooses to view	No	0
UserID	The user ID to identify what calendar to retrieve	No	0

Operations

Name	Description	Is Query	Is Polymorphic
createCalendar	Creates a calendar depending on what type is chosen for the user	Yes	No
changeCalendar	Changes the type of calendar the user wants to view	Yes	No
viewPersonalCalendar	Retrieves the user's personal calendar and displays it on the screen	Yes	No

3.3.2 Chat


Chat
+ ChatID: Int
+ MessageID: Int
+ UserID: Int
+ TimeStamp: String
+ IsUpdate: Boolean
+ Content: String
+ postMessage(chatID, userID, content, isUpdate): String
+ deleteMessage(messageID): Void
+ listChatMessages(chatID):

Description: Outlines the functionality of a chat system, including methods for posting, deleting, and listing messages within a chat

Visibility: Public

Is Abstract: No

Additional Information:

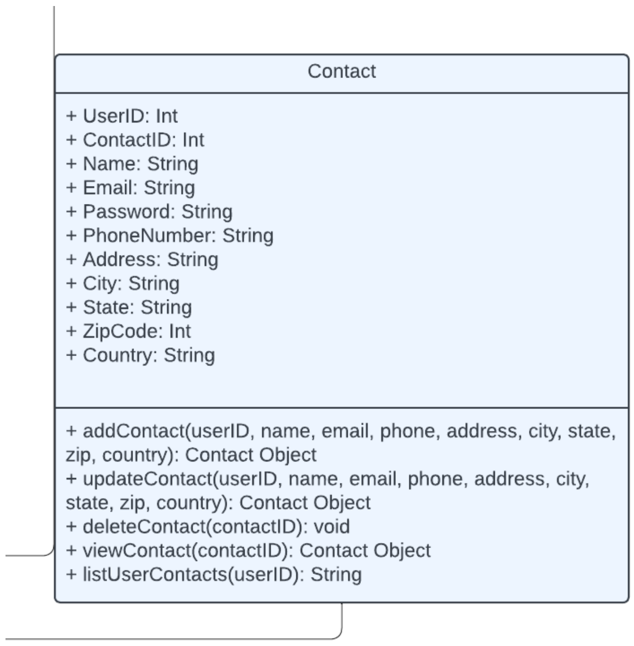
Attributes

Name	**Description**	**Read Only**	**Multiplicity**
ChatID	ID number of the chat	No	0
MessageID	ID number of the message	No	0
UserID	ID number of the user sending the message	No	0
TimeStamp	A time the message was sent	No	0
Content	String message of the user input	No	0
IsUpdate	Boolean, whether it is a message or an update notification	No	0

Operations

Name	**Description**	**Is Query**	**Is Polymorphic**
postMessage	Uploads the message to the chat	Yes	No
deleteMessage	Deletes the message from the chat	Yes	No
listChatMessages	Lists all the messages from the chat history	Yes	No

3.3.3 Contact


Contact
+ UserID: Int
+ ContactID: Int
+ Name: String
+ Email: String
+ Password: String
+ PhoneNumber: String
+ Address: String
+ City: String
+ State: String
+ ZipCode: Int
+ Country: String
+ addContact(userID, name, email, phone, address, city, state, zip, country): Contact Object
+ updateContact(userID, name, email, phone, address, city, state, zip, country): Contact Object
+ deleteContact(contactID): void
+ viewContact(contactID): Contact Object
+ listUserContacts(userID): String

Description: Outlines a contact management system that includes adding, updating, deleting, viewing, and listing user contacts

Visibility: Public

Is Abstract: No

Additional Information:

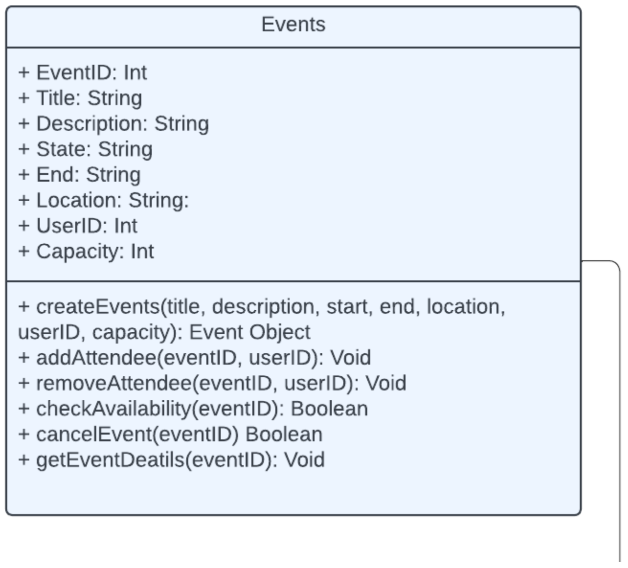
Attributes

Name	**Description**	**Read Only**	**Multiplicity**
UserID	ID number to identify the user	Yes	0
ContactID	ID number to identify the contact	Yes	0
Name	Name of the user	No	0
Email	Email of the user	No	0
Password	User’s password	No	0
PhoneNumber	User’s number	No	0
Address	User’s home address	No	0
City	The home city of the user	No	0
State	State of the user	No	0
ZipCode	User’s zip code	No	0
Country	User’s country of residency	No	0

Operations

Name	Description	Is Query	Is Polymorphic
addContact	The user creates their contact	Yes	No
updateContact	User updates the information associated with their personal profile	Yes	No
deleteContact	User deletes the information in their profile	Yes	No
viewContact	Displays the user’s profile when the user clicks on their account	Yes	No
listUserContacts	Displays the user’s contact information when another person clicks on their profile	Yes	No

3.3.4 Events


Events
+ EventID: Int
+ Title: String
+ Description: String
+ State: String
+ End: String
+ Location: String:
+ UserID: Int
+ Capacity: Int
+ createEvents(title, description, start, end, location, userID, capacity): Event Object
+ addAttendee(eventID, userID): Void
+ removeAttendee(eventID, userID): Void
+ checkAvailability(eventID): Boolean
+ cancelEvent(eventID) Boolean
+ getEventDeatils(eventID): Void

Description: Outlines an event management system that includes event creation, cancelation, addition, and event information retrievals

Visibility: Public

Is Abstract: No

Additional Information:

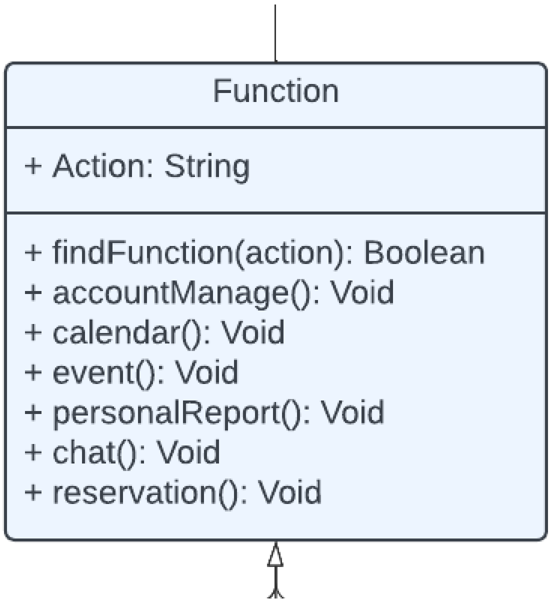
Attributes

Name	**Description**	**Read Only**	**Multiplicity**
EventID	Event ID number	Yes	0
Title	Title of the event	No	0
Description	A brief description of what the event is	No	0
State	The start time of the event	No	0
End	The end time of the event	No	0
Location	Location of where the event is taking place	No	0
UserID	ID number of the user who decided to volunteer for the event	Yes	0
Capacity	Number of volunteers needed	No	0

Operations

Name	Description	Is Query	Is Polymorphic
createEvent	Create the event and publish it on the general calendar using the information the user enters into the system,	Yes	No
addAttendee	Another user decides to volunteer for the event, so it updates the number of volunteers for the event and adds it to the respective user's personal calendar	Yes	No
removeAttendee	The attendee cancels the event, so the system removes the attendee from the list of volunteers.	Yes	No
checkAvailability	Returns the amount of open volunteer spaces for an event	Yes	No
cancelEvent	Cancels the entire event, removes all volunteers, and removes it from the general calendar	Yes	No
getEventDetails	Retrieves all the event information for a particular event when a user clicks on an event and displays it on the interface	Yes	No

3.3.5 Function



Description: Outlines a function management system that oversees what function the user chooses and redirects the system to the correct class to execute the function abilities

Visibility: Public

Is Abstract: No

Additional Information:

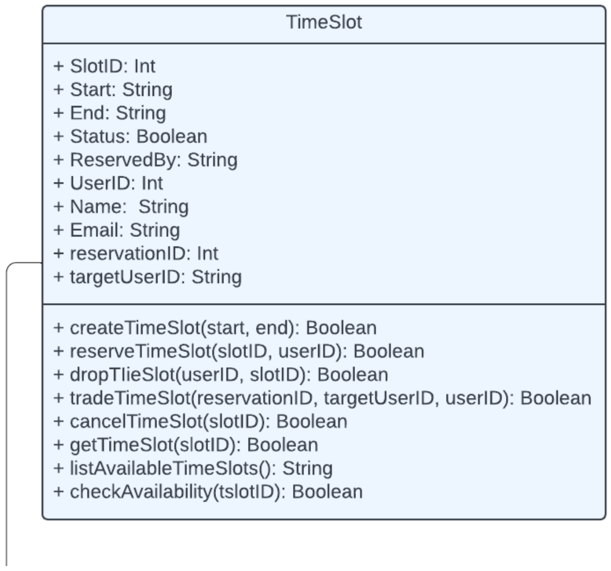
Attributes

Name	Description	Read Only	Multiplicity
Action	The name of the action the user chose to perform	Yes	0

Operations

Name	Description	Is Query	Is Polymorphic
findFunction	Displays the functions of the app for the user to choose	Yes	No
accountManage	Redirects the user to the account management class	Yes	No
Calendar	Redirects the user to the calendar class	Yes	No
Event	Redirects the users to the event management class	Yes	No
personalReport	Generates a personal report for the user	Yes	No
Chat	Redirects the user to the chat	Yes	No
reservation	Redirects user to the time-slot reservation class	Yes	No

3.3.6 TimeSlot


TimeSlot
+ SlotID: Int
+ Start: String
+ End: String
+ Status: Boolean
+ ReservedBy: String
+ UserID: Int
+ Name: String
+ Email: String
+ reservationID: Int
+ targetUserID: String
+ createTimeSlot(start, end): Boolean
+ reserveTimeSlot(slotID, userID): Boolean
+ dropTlieSlot(userID, slotID): Boolean
+ tradeTimeSlot(reservationID, targetUserID, userID): Boolean
+ cancelTimeSlot(slotID): Boolean
+ getTimeSlot(slotID): Boolean
+ listAvailableTimeSlots(): String
+ checkAvailability(tslotID): Boolean

Description: Outlines a contact management system that includes adding, updating, deleting, viewing, and listing user contacts

Visibility: Public

Is Abstract: No

Additional Information:

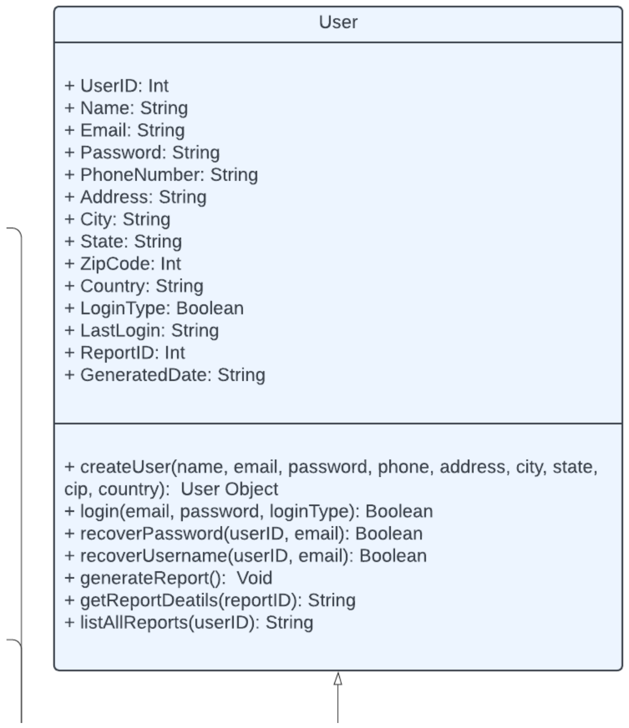
Attributes

Name	**Description**	**Read Only**	**Multiplicity**
SlotID	ID number of the time slot	Yes	0
Start	Start time of the time slot	Yes	0
End	End time of the time slot	Yes	0
Status	Whether or not the time slot has been reserved/open	No	0
ReservedBy	The user the timeslot was reserved by	Yes	0
UserID	ID number of the user	Yes	0
Name	Name of the user	Yes	0
Email	Email of the user	Yes	0
reservationID	ID number of the reservation	Yes	0
targetUserID	ID number of the user to trade slots with	Yes	0

Operations

Name	Description	Is Query	Is Polymorphic
createTimeSlot	Allows the user to create a new time slot	Yes	No
reserveTimeSlot	Allows users to reserve a time slot	Yes	No
dropTlieSlot	Allows users to drop a time slot	Yes	No
tradeTimeSlot	Allows the user to trade time slots with another user	Yes	No
cancelTimeSlot	Allows user to cancel all their time slot reservations	Yes	No
getTimeSlot	Displays the user's timeslot reservation	Yes	No
listAvailibility	Lists all the open timeslots	Yes	No
checkAvabilability	Checks to see what timeslots are availible	Yes	No

3.3.7 User


User
+ UserID: Int
+ Name: String
+ Email: String
+ Password: String
+ PhoneNumber: String
+ Address: String
+ City: String
+ State: String
+ ZipCode: Int
+ Country: String
+ LoginType: Boolean
+ LastLogin: String
+ ReportID: Int
+ GeneratedDate: String
+ createUser(name, email, password, phone, address, city, state, cip, country): User Object
+ login(email, password, loginType): Boolean
+ recoverPassword(userID, email): Boolean
+ recoverUsername(userID, email): Boolean
+ generateReport(): Void
+ getReportDeatils(reportID): String
+ listAllReports(userID): String

Description: Outlines a contact management system that includes adding, updating, deleting, viewing, and listing user contacts

Visibility: Public

Is Abstract: No

Additional Information:

Attributes

Name	**Description**	**Read Only**	**Multiplicity**
UserID	ID number of the user	Yes	0
Name	User’s name	Yes	0
Email	User’s email	Yes	0
Password	User’s password	Yes	0
PhoneNumber	User’s phone number (must be a valid number)	Yes	0
Address	User’s Address	Yes	0
City	City of residence	Yes	0
State	State of residence	Yes	0
ZipCode	User’s residence zip code	Yes	0
Country	Country of residence	Yes	0
LoginType	Whether or not the user is existing or new	No	0

LastLogin	The last login date of the user	Yes	0
ReportID	ID number of the report	Yes	0
GeneratedDate	Date when the report was generated	No	0

Operations

Name	**Description**	**Is Query**	**Is Polymorphic**
createUser	Creates a new user profile for the system	Yes	No
Login	Asks for the login information of the user and attempts to validate and log them in	Yes	No
recoverPassword	Sends a link to email/phone number to reset password	Yes	No
recoverUsername	Sends a link to email/phone number to reset username	Yes	No
generateReport	Generates an informal report of all user interactions for the systems administrator periodically	Yes	No
getReportDetails	Retrieves/logs all user interactions and stores the information in the database	Yes	No
listAllReports	Lists all the previous reports of user interactions	Yes	No

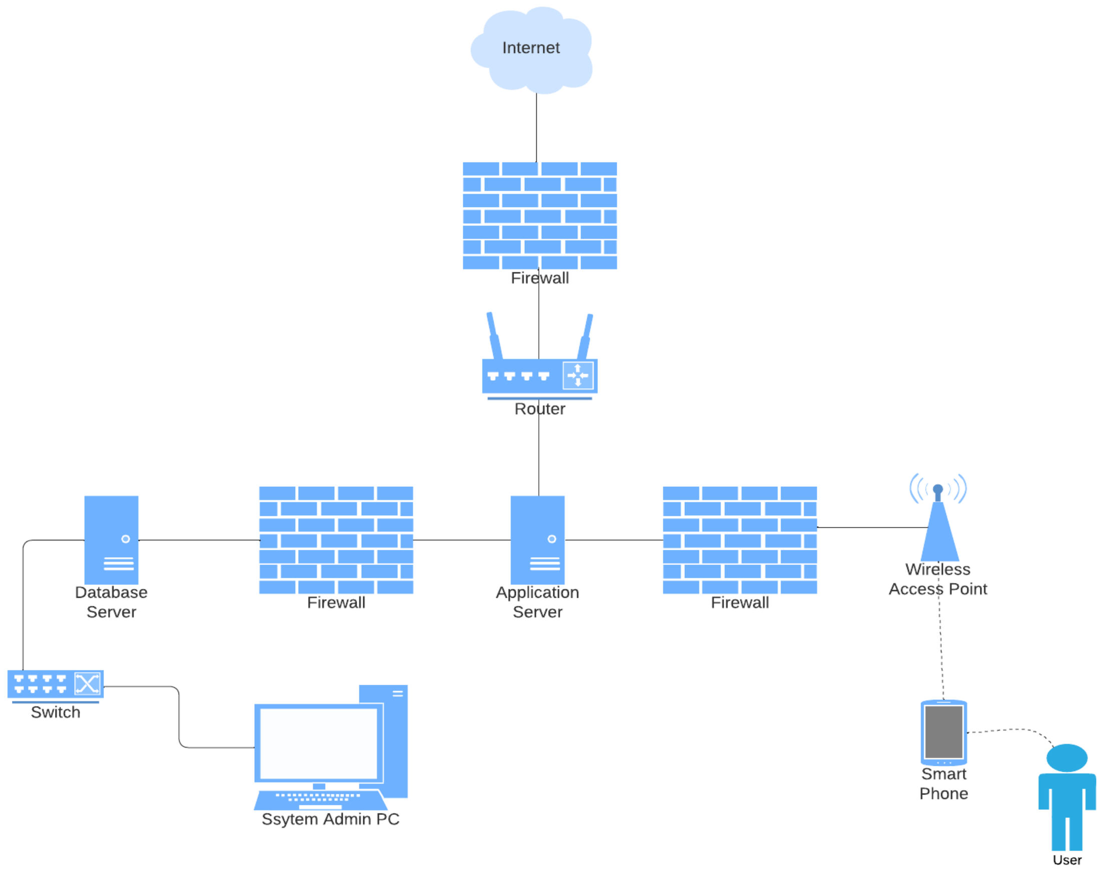
4. Architecture Design

4.1. Architecture Overview

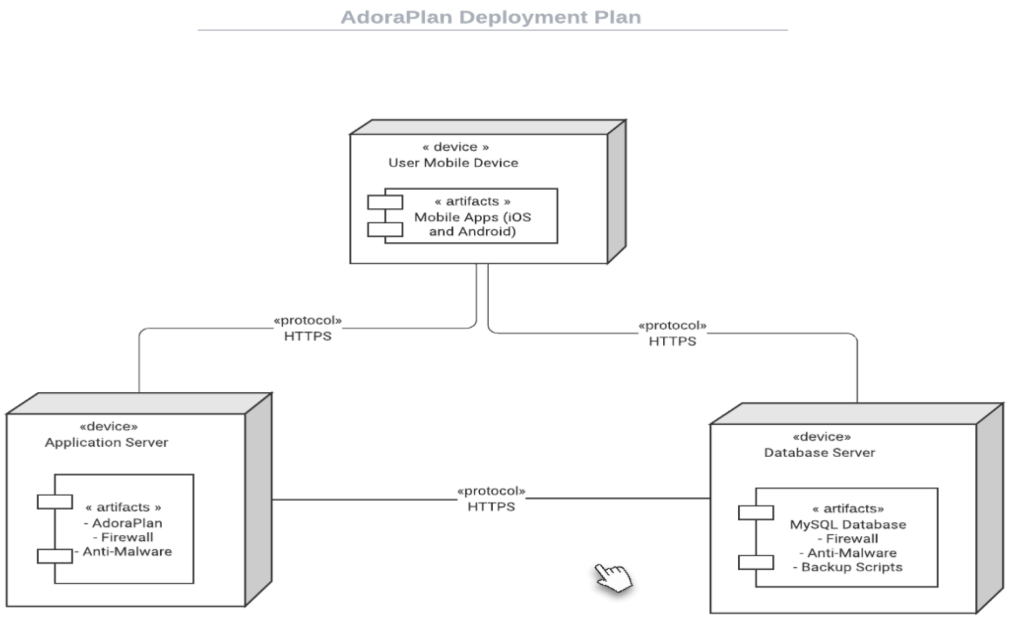
This section of the document provides a comprehensive summary of the architecture of AdoraPlan. It will detail the system's infrastructure model, hardware, software requirements, and security plan. This section will provide insight into the material and logical components of the systems and the necessary hardware and software to support the application. AdoraPlan will be a 3-tier client-server system. This will include a presentation, supplication logic, and data tier. Furthermore, the misfeature will consist of mobile clients, an application server, and a database server. To support this application, it is recommended that two servers be purchased: one for the application and another for the database.

4.2. Infrastructure Model.

4.2.1. Deployment Diagram 1 – Architecture Overview



4.2.2. Deployment Diagram 2 – Nodes and Artifacts



4.3. Hardware and Software Requirements

4.3.1. Hardware Components

To implement AdoraPlan’s system architecture, we need the following components:

1. Application Server: server to host application logic
2. Database Server: server dedicated to the database
3. Client Mobile Devices: Users can access the app using smartphones or tablets

4.3.2. Required Software Components

The software needed to implement the system includes:

1. Operating System: Linux for servers
2. Database MAangemnet System: MySQL 8.0 or higher
3. Mobile Development Platforms: Xcode for iOS and Android Studio for Android
4. Security Software: SSL certificates, firewall, and anti-malware tools

4.4. Security Plan

4.4.1. Security Overview

AdoraPlan handles sensitive user data, such as names, email, phone numbers, and personal scheduling information, which makes it a target for cyber threats. Key risks include:

1. Unauthorized access (e.g., account hijacking, admin panel intrusion)
2. Data breaches (e.g., unencrypted data in transit or at rest)
3. System availability failures (e.g., DDoS attacks, server outages)
4. Insider threats or negligent users
5. Non-compliance with data privacy regulations (e.g., GDPR)

The security strategy addresses application, network, physical, and user-level risks through layered defenses, encryption, access control, monitoring, and secure development practices.

4.4.2. Security Plan

The plan below is organized to cover potential threats to network, physical, application, and user security. This security plan aims to address various threats through specific measures and outlines the components, threats, and corresponding controls.

Component	Threat	Control
Application Server	Unauthorized access to backend or admin routes	- Enforce Two-Factor Authentification (2FA) for admin accounts - Use OAuth 2.0 or similar secure authentication protocols - Conduct regular penetration testing
	Server-side exploits or injection attacks	- Input validation and sanitation - Use frameworks that prevent CSRF/XSS - Harden server configuration (e.g. disable directory listings)
Database Server	Data breach through unauthorized access	- Encrypt sensitive data at rest using AES-256-Encrypt data in transit with SSL/TLS - Use role-based access control (RBAC)
	SQL injection or data leakage	- Use prepared statements and ORM tools - Regularly patch database software - Monitor for unusual queries
User Accounts	Brute-force attacks, phishing, or compromised accounts	- Enforce strong password policies) min length special characters) - Rate-limit login attempts - Implement password reset notifications
	Account impersonation or takeover	- Session expiration after inactivity - Notify users of unusual login locations
Network	Eavesdropping, man-in-the-middle attacks	- HTTPS via SSL certificates - Secure APIs using JWTs or signed tokens - Enable HTTP security headers

	Unauthorized network intrusion	- Use firewalls and IDS/IPS (Intrusion Detection/Prevention Systems) - Secure server ports and services (close unused ones
	DDoS attacks	- Use rate limiting and CAPTCHAs - Host on cloud providers that support DDoS mitigation (e.g., AWS Shield, Cloudflare)
Backup System	Data loss due to hardware failure, ransomware, or accidental deletion	- Daily automated backups stored off-site - Test backup recovery regularly - Encrypt backup data and protect it with access controls
Physical Security	Theft or unauthorized access to devices	- Keep database and application servers in secure server rooms - Restrict physical access to authorized personnel only
Logging & Monitoring	Delayed detection of threats or breaches	- Maintain audit logs of access and changes - Use log analysis tools or SIEM systems (e.g., Splunk, ELK) - Configure real-time alerts for suspicious activity
Development Practices	Introduction of insecure code or dependencies	- Conduct code reviews and automated vulnerability scanning (e.g., Snyk, SonarQube) - Keep dependencies up to date using package managers with security advisories
User Privacy Compliance	Violation of data protection regulations (e.g., GDPR, FLSA)	- Include user consent mechanisms for data collection - Allow data access and deletion requests - Publish a clear privacy policy

Additional Measures:

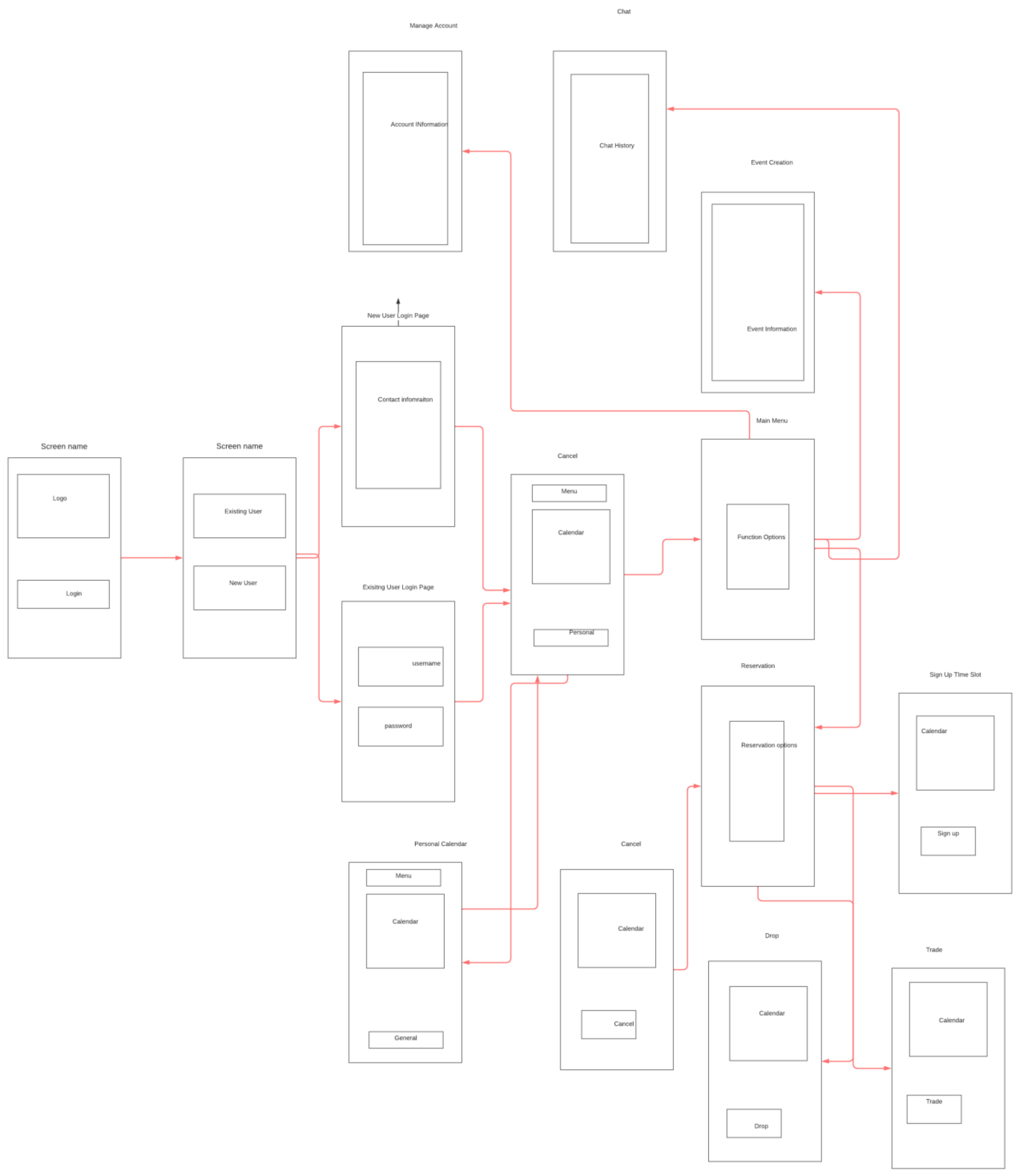
1. Security Awareness Training: Educate all technical staff and volunteers about common cyber threats (phishing, social engineering).
2. Secure Development Lifecycle (SDL): Incorporate security checkpoints at each development stage.
3. Disaster Recovery Plan: Prepare response procedures for data loss, system compromise, or outages.
4. Disaster Recovery Plan: Prepare response procedures for data loss, system compromise, or outages.
5. Legal Safeguards: Ensure that all third-party services and hosting platforms comply with applicable laws.

5. <u>User-Interface</u>

5.1. <u>User-Interface Requirements and Constraints</u>

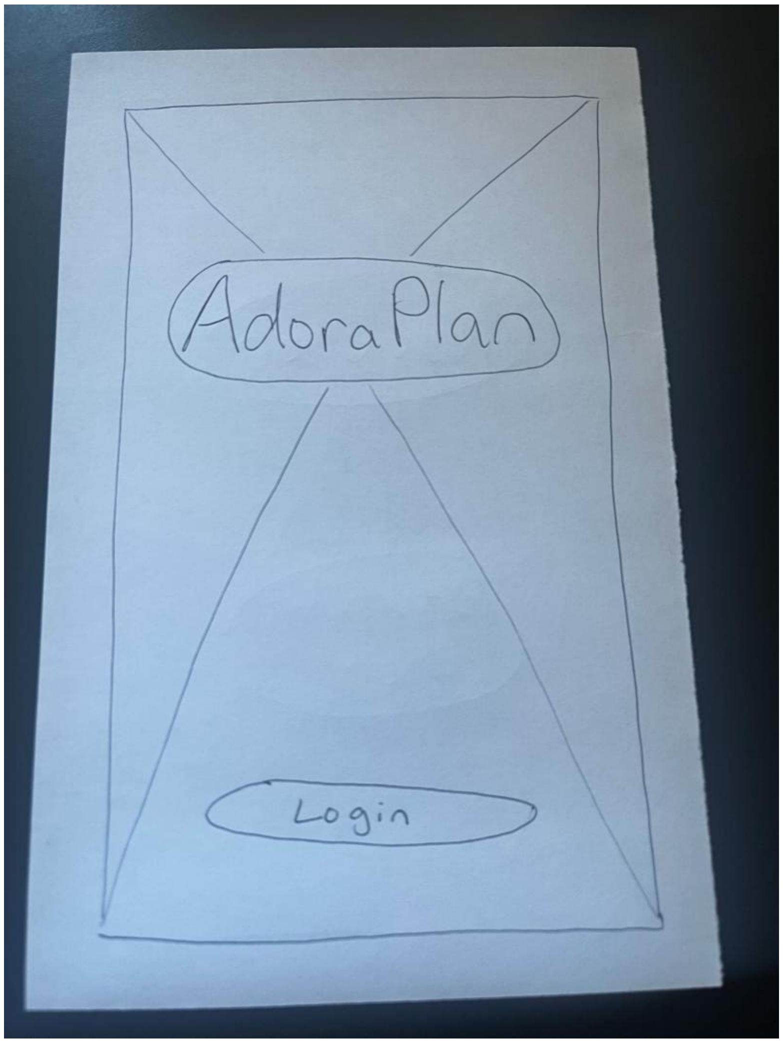
The user interface (UI) for AdoraPlan must be user-friendly and accessible to users of varying technical abilities. The primary users include young adults and elderly parishioners, who will use the app to sign up for adoration time slots and manage their commitments. If any problems arise, administrators will have access to and handle the schedule. Some fundamental guiding principles for UI design are clarity and simplicity. The app interface should facilitate quick access to actions and minimize the learning curve for new users. This section introduces the UI design elements that users will interact with and provides detailed diagrams and wireframes to give an overview of the user interface structure.

5.2. Window/Screen Navigation Diagram



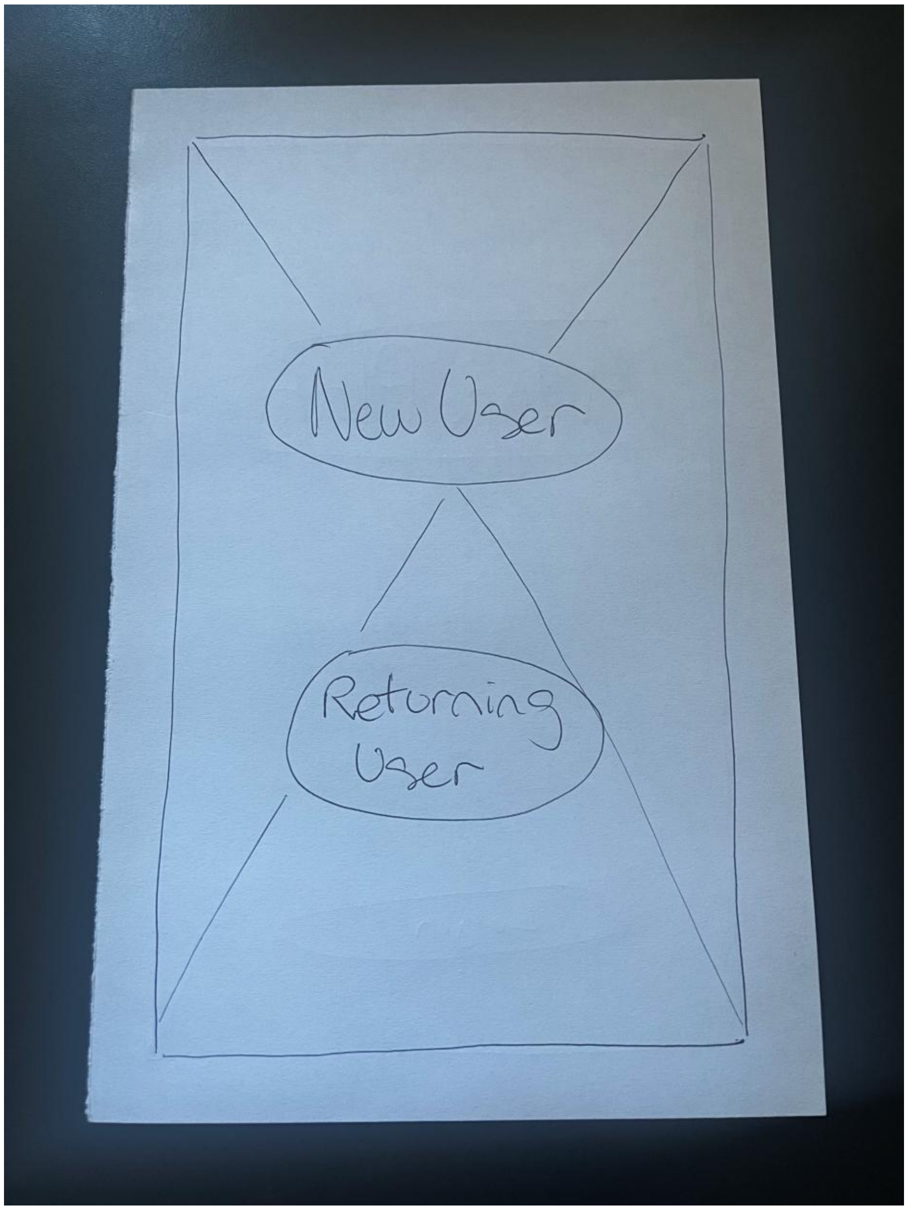
5.3. UI Wireframes

5.3.1. Welcome Page



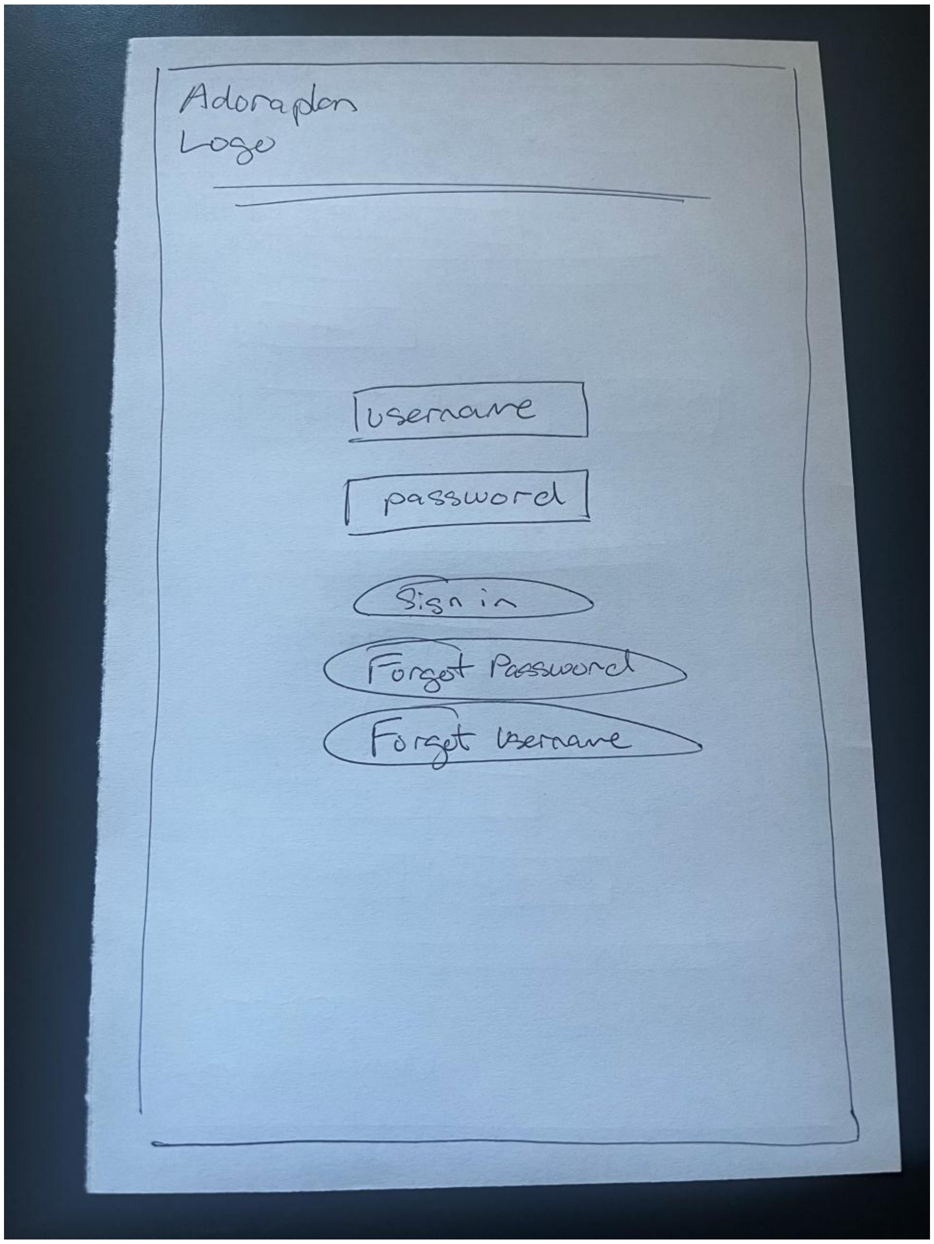
The first screen the user sees after downloading and opening the app. The front page prompts the user to sign in.

5.3.2. Sign-in Page



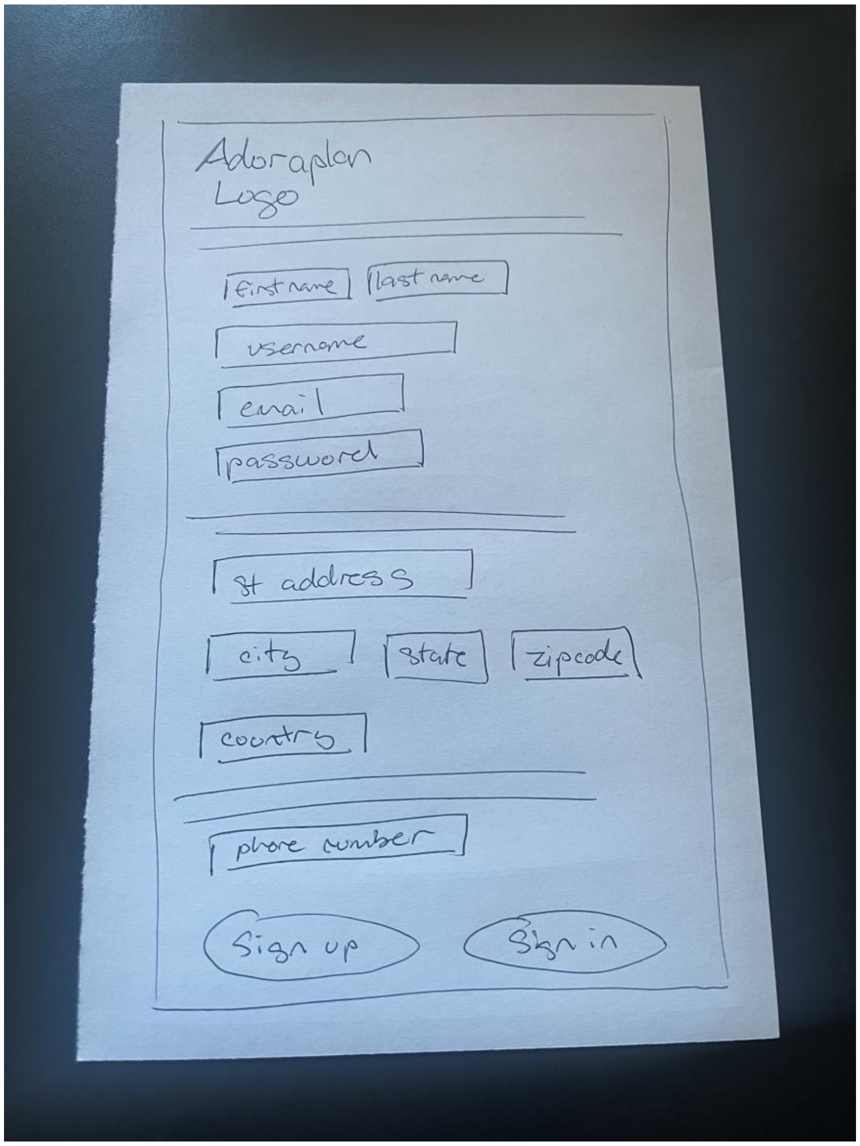
Prompts the user to select the sign-in option. If a new user is needed, the system will prompt the user to create an account and log in. If an existing user, the system will ask the user to enter their login information.

5.3.3. Existing User Login Page



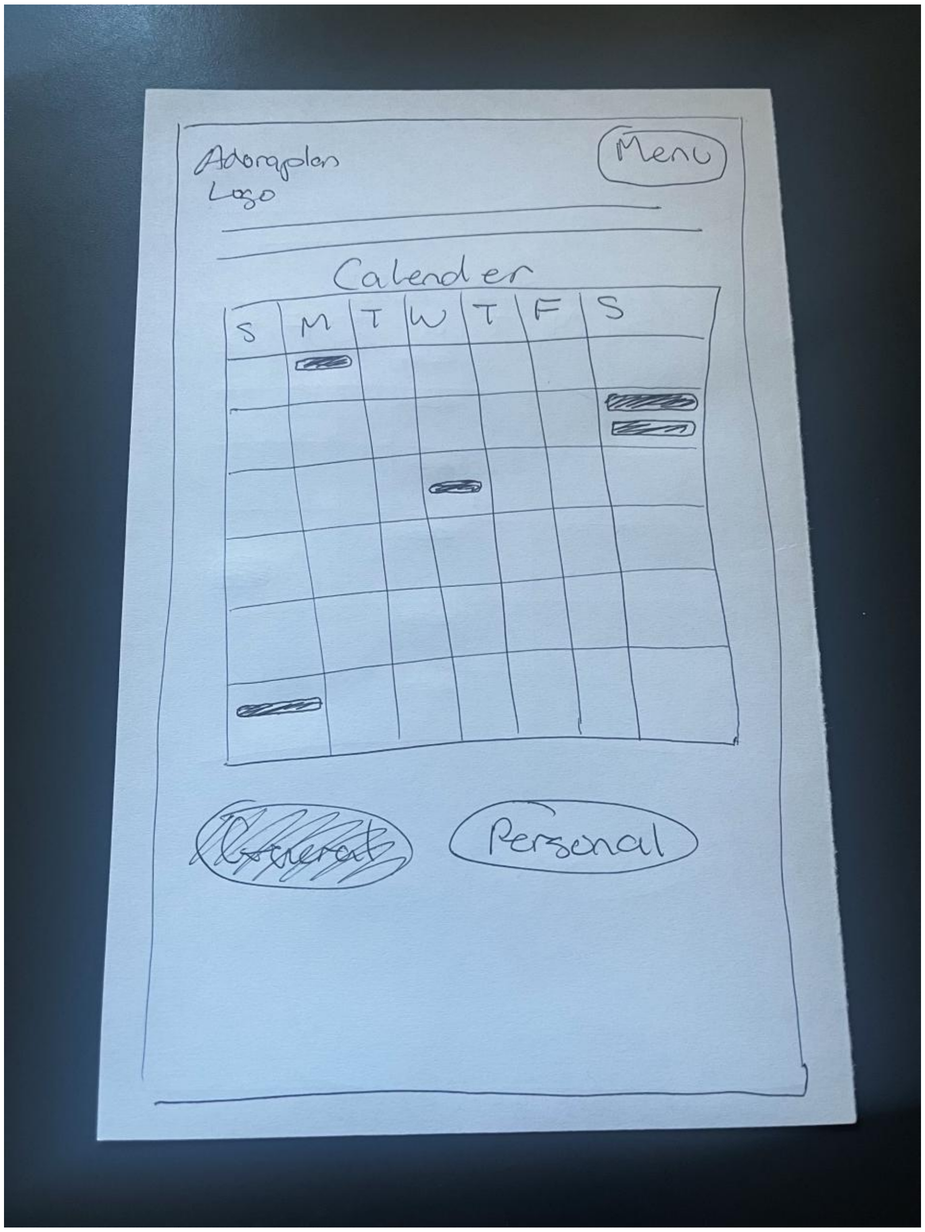
The system prompts the user for a valid username and password. It provides options to recover both the password and username if forgotten.

5.3.4. New User Login Page

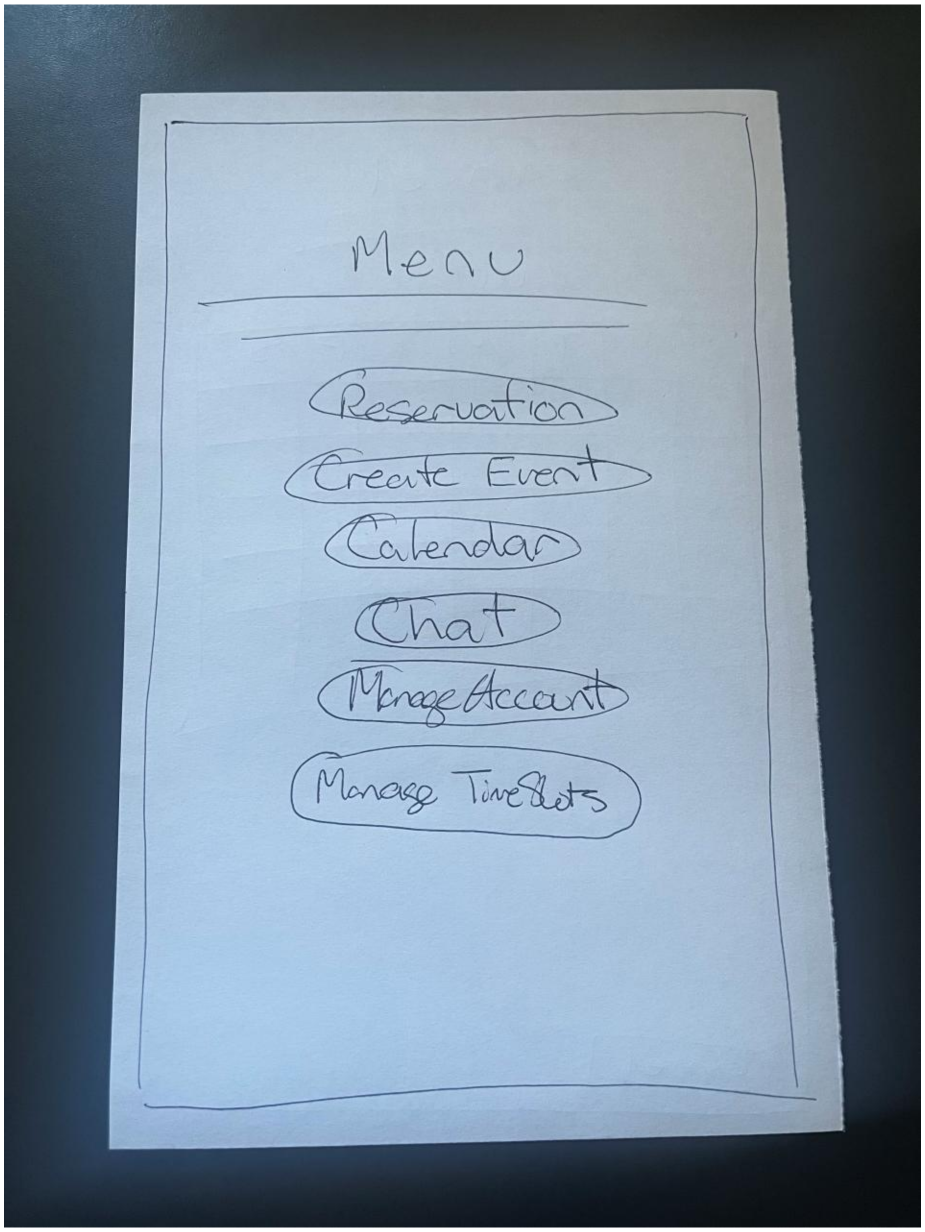


The system asks users to enter all personal information to create their profile. Options to sign up and sign in are listed below and can only be pressed when all required information is filled out.

5.3.5. Main Screen

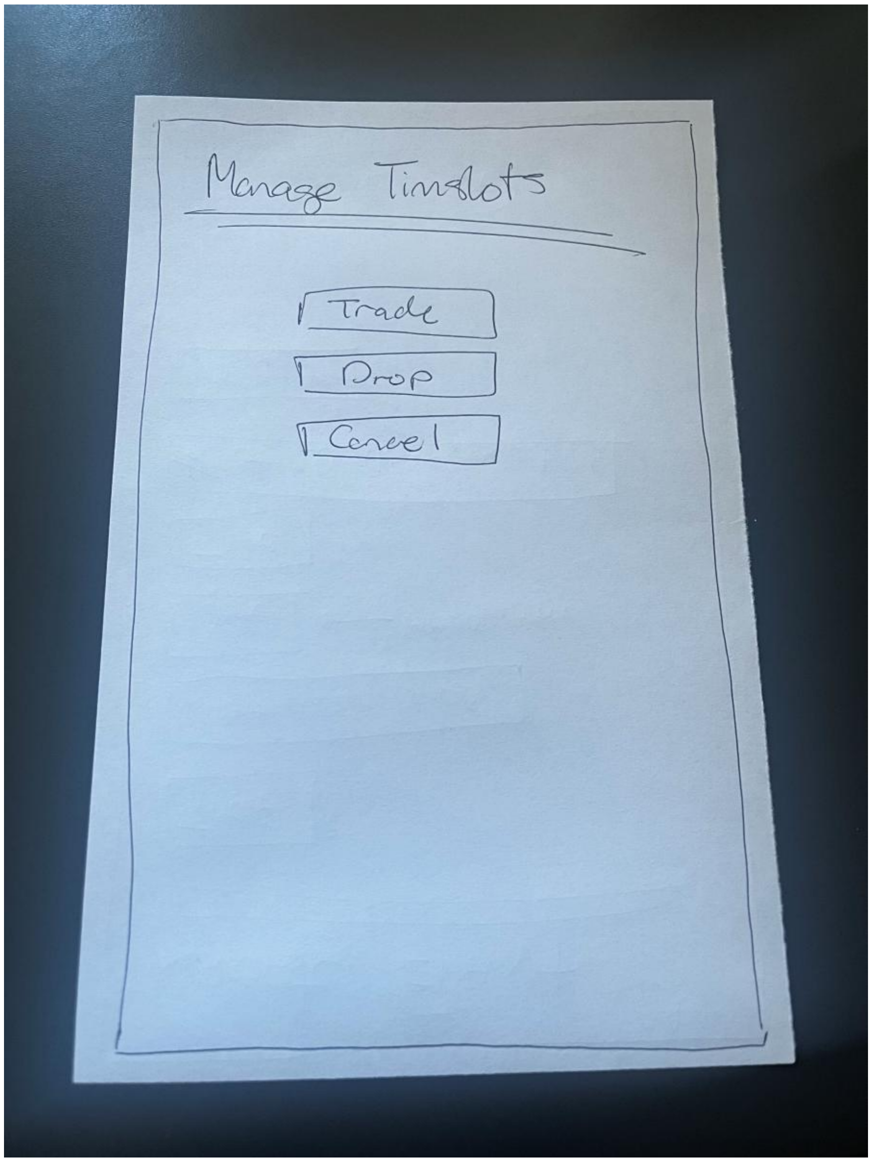


5.3.6. Main Menu



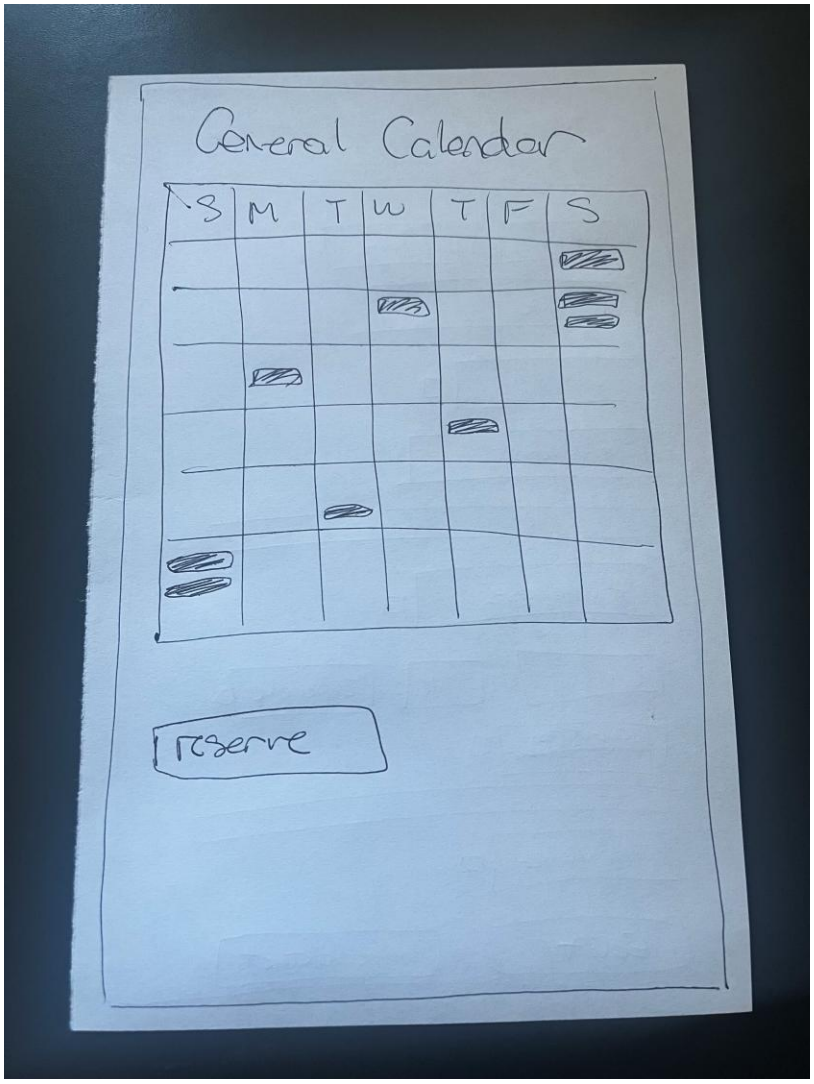
The Main Menu screen displays the other functionalities of the app and allows the user to choose which option to execute.

5.3.7. Reservation



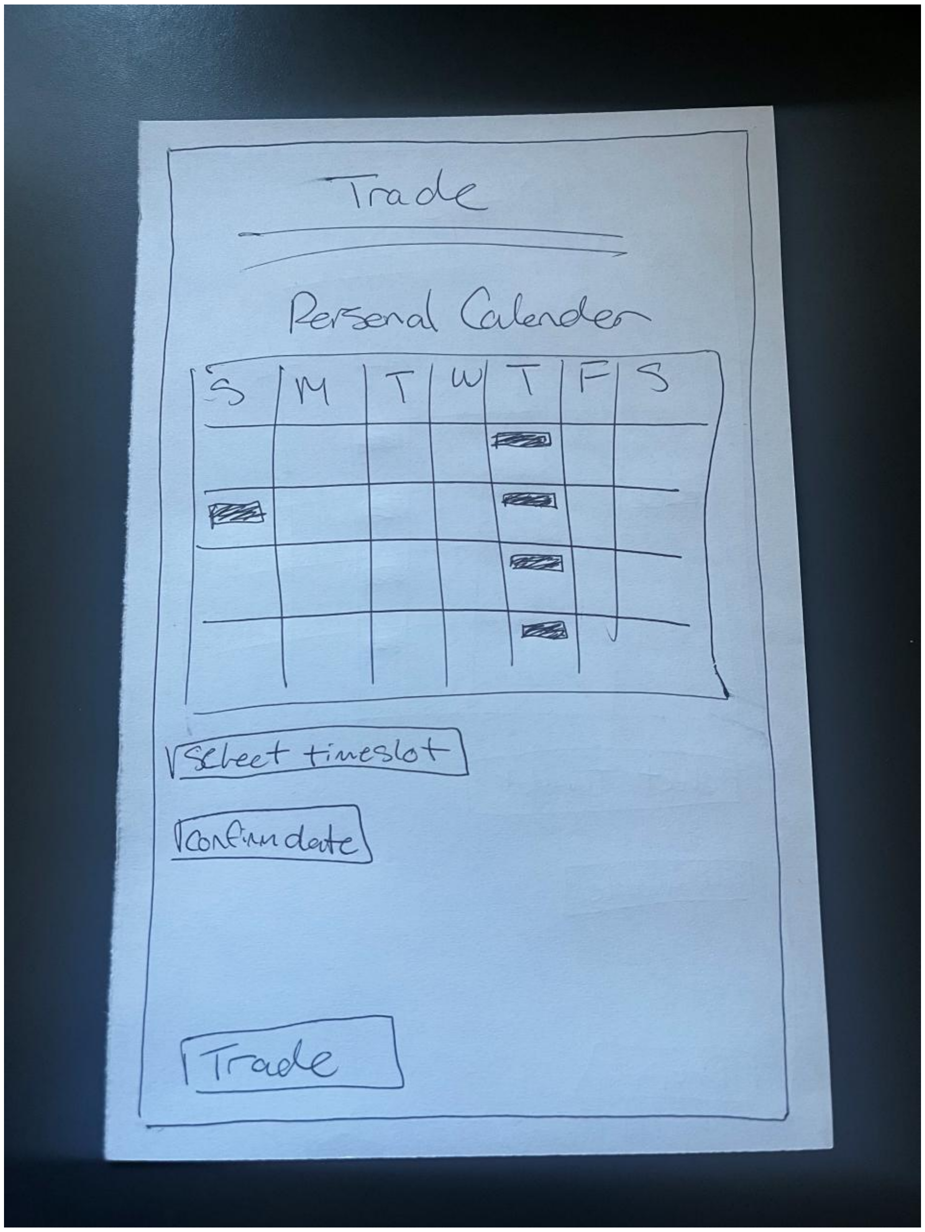
The display of the reservation screen allows the user to choose whether to trade, drop, or cancel their timeslot. They can also access the regular time-slot reservation for each option. I need to switch from a personal to a general calendar.

5.3.8. Sign-up for Time-Slot



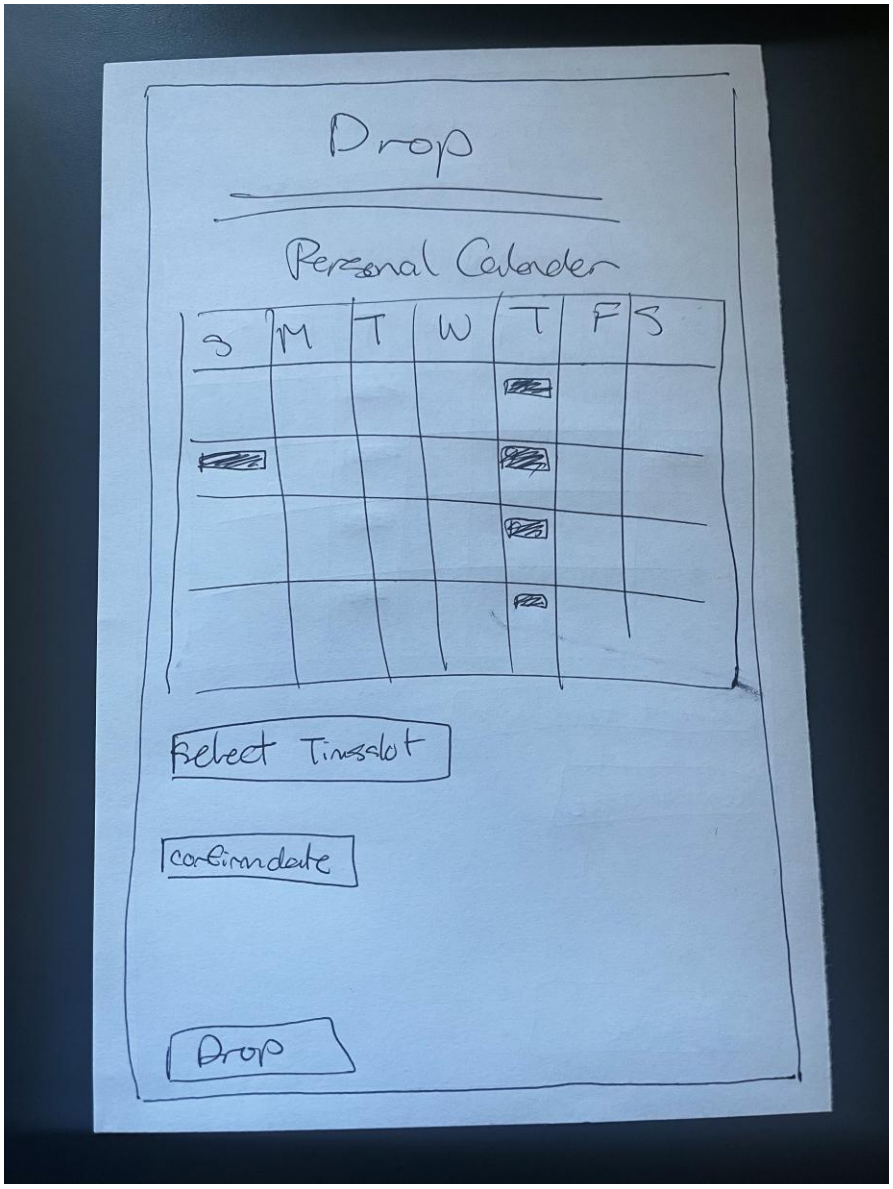
The user selects one of the dark time slots and clicks reserve to add it to their personal events list.

5.3.9. Trade



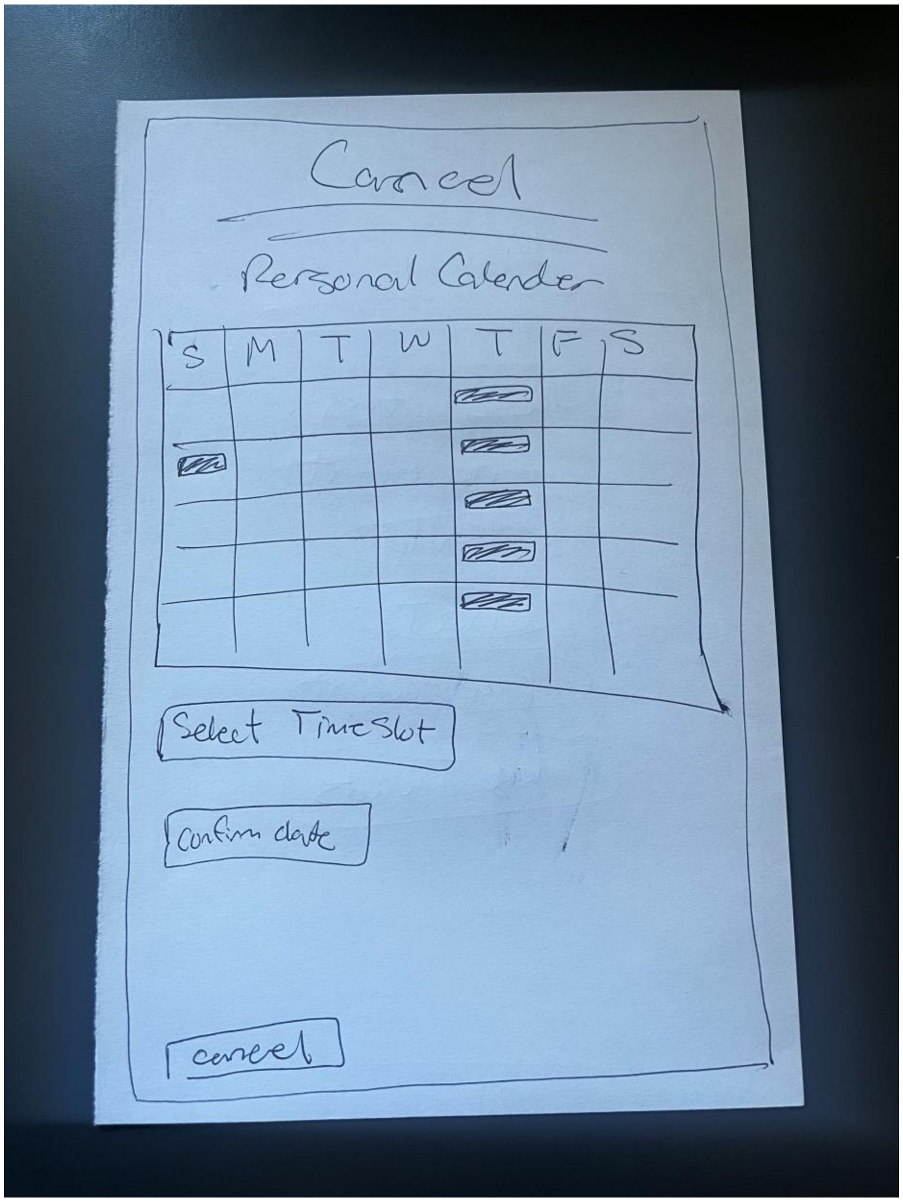
Users select the time slot they want to trade, confirm the date, and press trade. The trade button will notify the chatroom that the user wants to exchange this timeslot for another one. One of the other users will pick up the shift, and the timeslots will automatically trade.

5.3.10. Drop



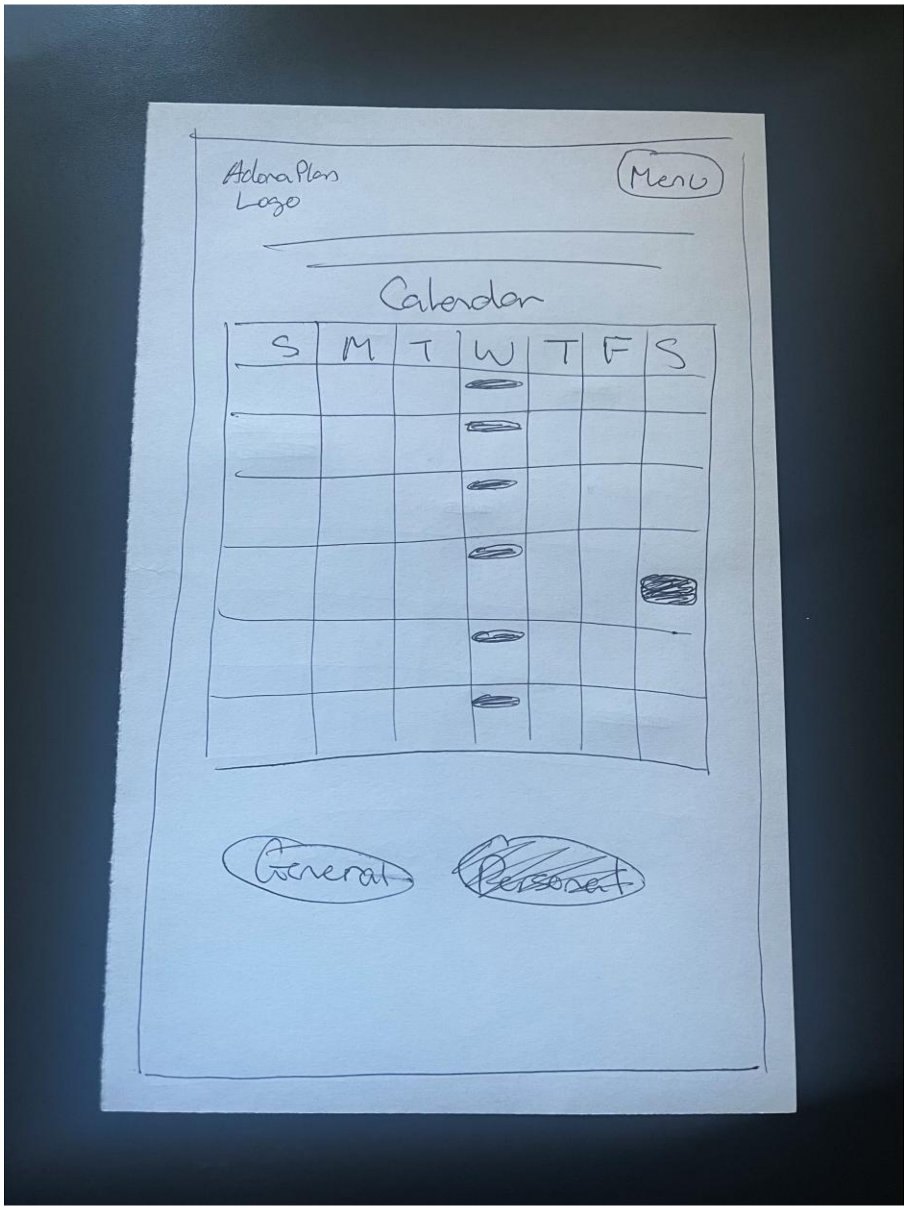
The user selects the timeslot they want to drop, confirms the date, clicks drop, and removes the user from that ONE of their adoration reservation.

5.3.11. Cancel



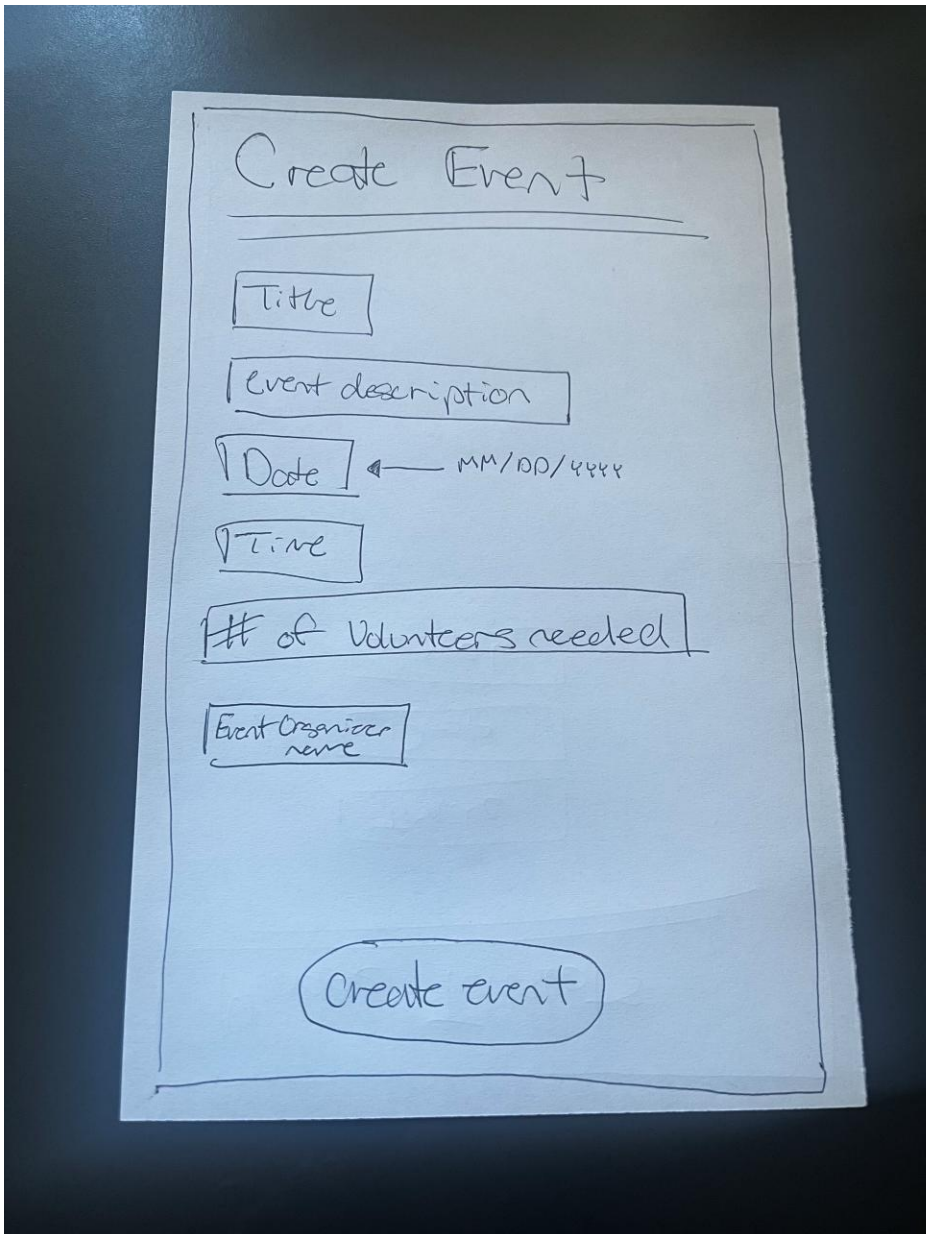
Users select the reservation they want to cancel, confirm the date, and click cancel, and the system removes them from ALL occurrences of that event.

5.3.12. Personal Calendar



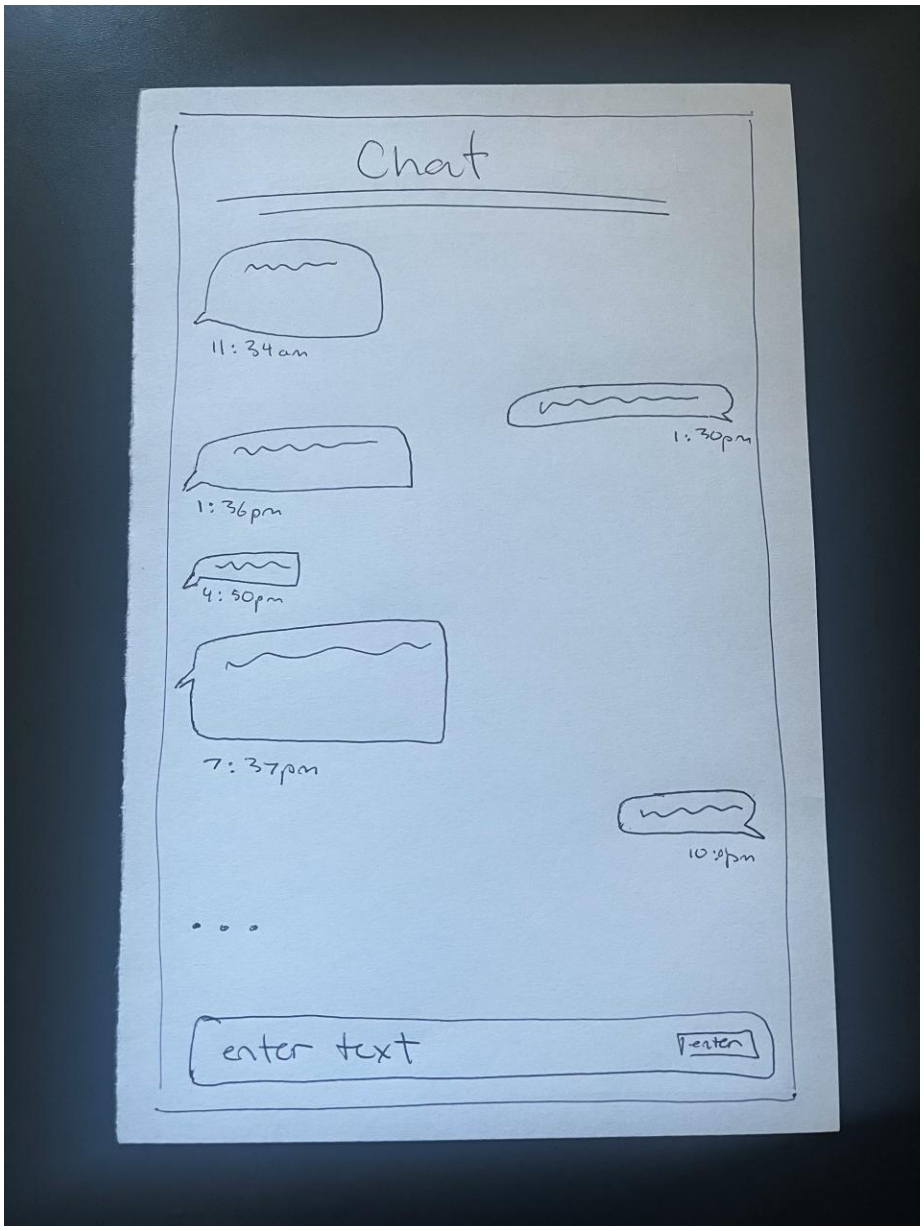
Users can switch from their general calendar to their calendar by clicking the personal button on the right.

5.3.13. Event Creation



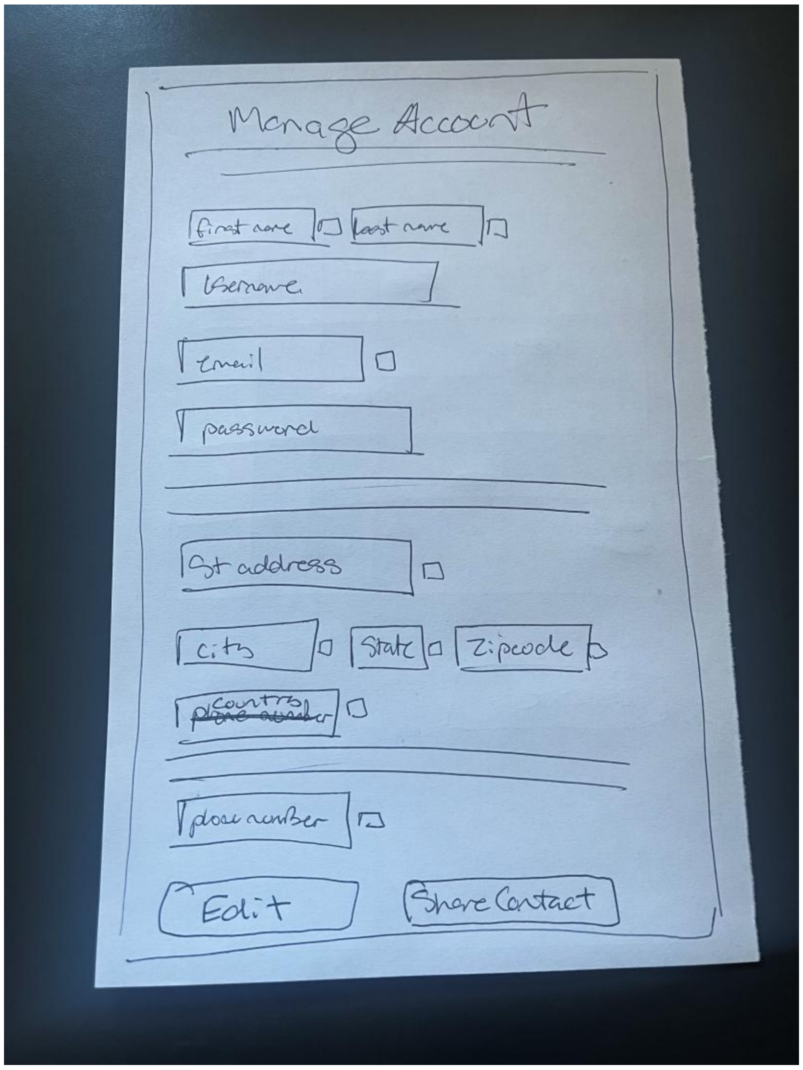
Create an Event page that asks the user for the event title, description, date, time, and number of volunteers. Once all the required information is entered, the user can click the Create event button, and the system will publish the event on the general calendar.

5.3.14. Chat



Chat interface between all the app users. Displays the text and time the message was sent. The user enters their message in the text box using the button and clicks “enter” when ready to send their message.

5.3.15. Manage Account



The users manage the account page. To edit the information, press the edit button, and it makes all the information customization. To save the changes, click the edit button again. To choose specific contacts to share., click the small square checkboxes on the right of each contact info. Press the share contact button again to make those things public on the application.

5.4. <u>Reports: "Formal Output" Design</u>

AdoraPlan does not utilize any formal report-generation software. The report generation it does is just an informal list of user interactions and logs for the systems administrator to keep track of.

6. Appendices

6.1. Glossary

F2A – two0factor authentication, a security method that uses two authentication methods to access an account

MySQL – “stands for “Structured Query Language.” SQL is the most common standardized language used to access databases. Depending on your programming environment, you might enter SQL directly (for example, to generate reports), embed SQL statements into code written in another language, or use a language-specific API that hides the SQL syntax.” (Oracle)

SSL certificates – “An SSL certificate is a digital certificate that authenticates a website's identity and enables an encrypted connection. SSL stands for Secure Sockets Layer, a security protocol that creates an encrypted link between a web server and a web browser.” (Kaspersky)

UML – “a standardized modeling language consisting of an integrated set of diagrams developed to help system and software developers specify, visualize, construct, and document the artifacts of software systems, as well as for business modeling and other non-software systems.” (Visual Paradigm)

6.2. References / Bibliography

Cameron, Andy, A.C. (2024, June 6th). W07 - Moving to Design. pptx

Department of Computer Science, Seattle Pacific University

https://canvas.spu.edu/courses/64100/modules/items/1070820

Cameron, Andy, A.C. (2024, June 6th). W08 - A Structural Data Modeling and UML Class Diagrams. pptx

Department of Computer Science, Seattle Pacific University

https://canvas.spu.edu/courses/64100/modules/items/1070821

Cameron, Andy, A.C. (2024, June 6th). W08 - B Physical Architecture Design. pptx

Department of Computer Science, Seattle Pacific University

https://canvas.spu.edu/courses/64100/modules/items/1070822

Cameron, Andy, A.C. (2024, June 6th). W08 - C UI Design. pptx

Department of Computer Science, Seattle Pacific University

https://canvas.spu.edu/courses/64100/modules/items/1070824

Cameron, Andy, A.C. (2024, June 6th). W08 - D Security. pptx

Department of Computer Science, Seattle Pacific University

https://canvas.spu.edu/courses/64100/modules/items/1070825

Kaspersky. (2023, April 19). *What is an SSL certificate – Definition and Explanation*. Usa.kaspersky.com.

https://usa.kaspersky.com/resource-center/definitions/what-is-a-ssl-certificate

Oracle. (2021). *What is MySQL?* Oracle.com. https://www.oracle.com/mysql/what-is-mysql/

Visual Paradigm. (2019). *What is Unified Modeling Language (UML)?* Visual-Paradigm.com. https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/