Design Report

NPCs

(Jennifer Duarte, Shambhawi Sharma, Inika Singh, Bridget Torres)

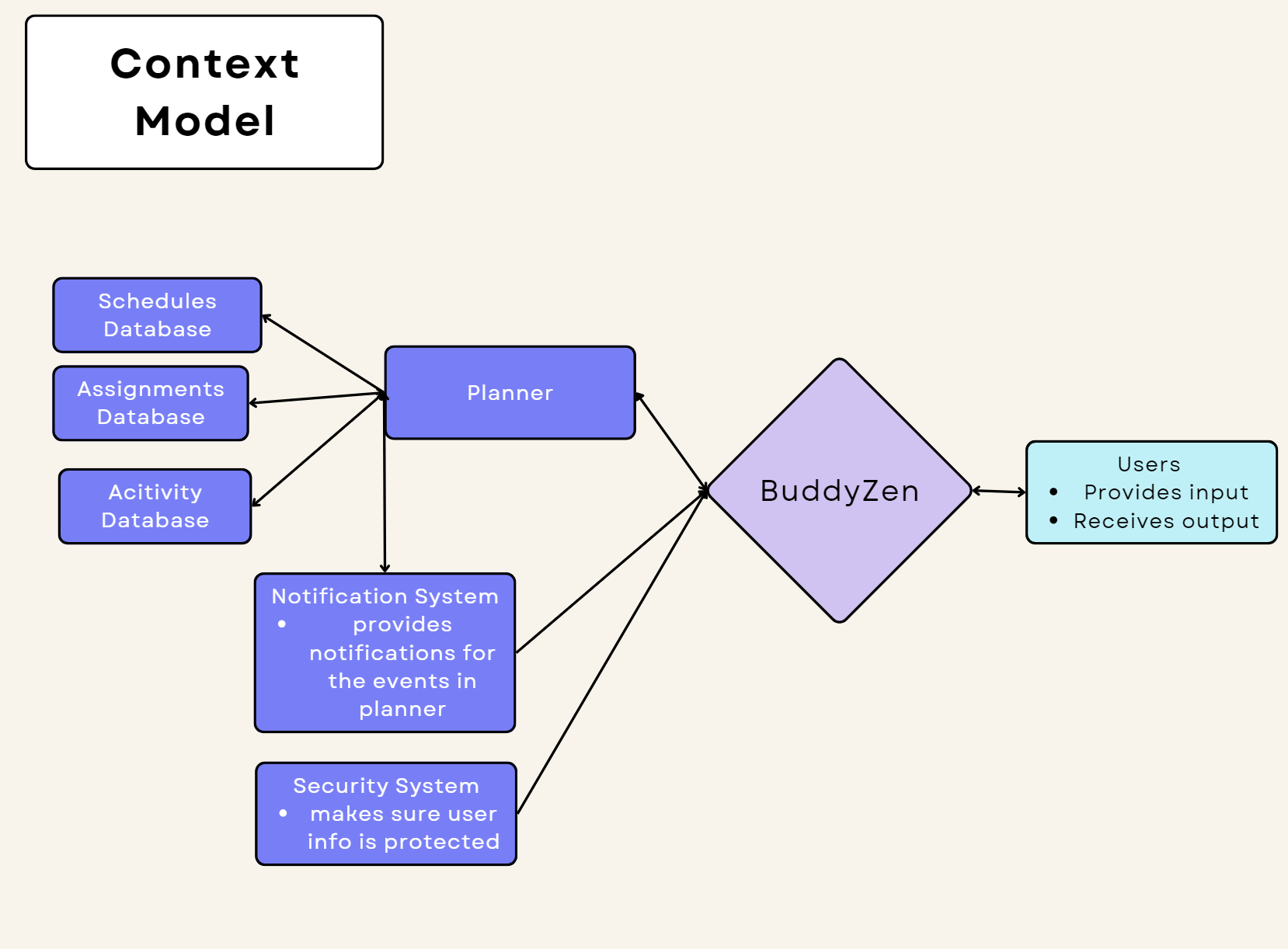
CS487 Software Engineering

11/9/2023

# 1. An Overall Summary

BuddyZen is reshaping the student experience with a helpful app designed to elevate organization and time management to new heights for greater efficiency. Designed while keeping in mind the demands of student life, BuddyZen introduces an advanced planner that integrates class schedules with start and end times and automatically adds assignment’s due dates for a comprehensive overview of the academic requirements for the student. In an effort to encourage students to prioritize their time, BuddyZen facilitates effortless prioritization of assignments since students can easily mark assignments once done, ensuring efficient time allocation. By providing consistent reminder notifications for upcoming due dates, BuddyZen ensures that students keep track of their academic obligations. Taking organization a step further, BuddyZen introduces intuitive features such as the ability to group different activities like club events into specific categories, resulting in a simplified and clutter-free approach to daily planning. BuddyZen provides itself as an ultimate companion for students who are seeking to have an organized and efficient academic experience.

2. Context Model and App Interaction



Above we share the context and system model for BuddyZen and it offers a description of how the application will interact within the broader system as it interacts with the various elements in its environment. The context model emphasizes BuddyZen’s core functionalities, including its ability to schedule events, set assignments in the planner, and have a reminder feature for events providing a clear illustration of how the program interacts with different components and creates a cohesive user experience. Likewise, the model explores the relationship and interaction between BuddyZen and the other components within the academic ecosystem. This overall outlines the information flow through the app’s integration with the notification system, planner layout, and security system. Therefore, this provides a transparent overview of how BuddyZen seamlessly integrates into the larger system. It highlights the central role of the app as a fundamental component in providing an efficient and effective time management system for students in their academic pursuits while providing security through the entire process of receiving and providing information.

3. User Navigation

In crafting the user interface (UI) sketches for BuddyZen, we have dedicated meticulous attention to creating an intuitive and visually appealing design that enhances the overall user experience. These sketches encapsulate the essence of BuddyZen's functionality, providing users with a seamless platform for managing schedules, tasks, and extracurricular activities.

### 3.1 Dashboard Overview

The UI sketch begins first and foremost with the login page that prompts the user to enter their login credentials. We have opted for a sky blue shade that is calming, along with our brand logo on the main page. Fields are provided below so that the user may enter their details and login (Figure 1).

Another primary part of our dashboard design is the main dashboard (Figure 3) (shown only when login details are saved), presenting an organized overview of the user's schedule. By pressing the three dots at the upper left of the main dashboard, there's a clean navigation bar with tabs for Calendar, Tasks, Activities, and Settings (Figure 2). The sidebar provides quick access to priority tasks and upcoming deadlines. The central area features a monthly calendar view. Clicking on the date brings us to the more detailed view which shows as a scrollable pop-up that contains a list of all past and upcoming schedules. Schedules that are past appear in red, immediately upcoming schedules show in green, and later schedules are all white. Arrow buttons are also provided on either side of the popup to help navigate between different dates quickly without needing to press the date itself (Figure 4).

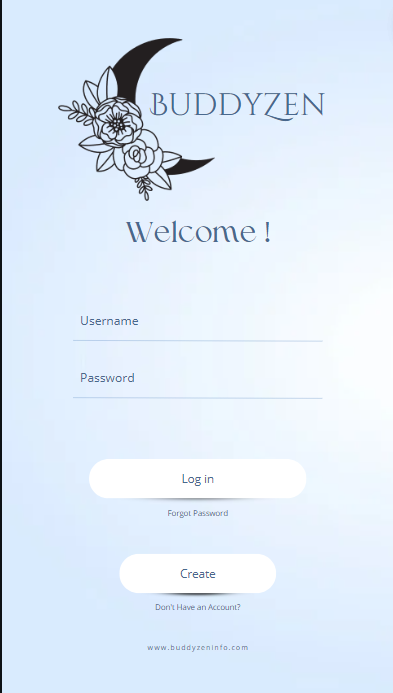


Figure 1: Login page for BuddyZen (design in Canva)

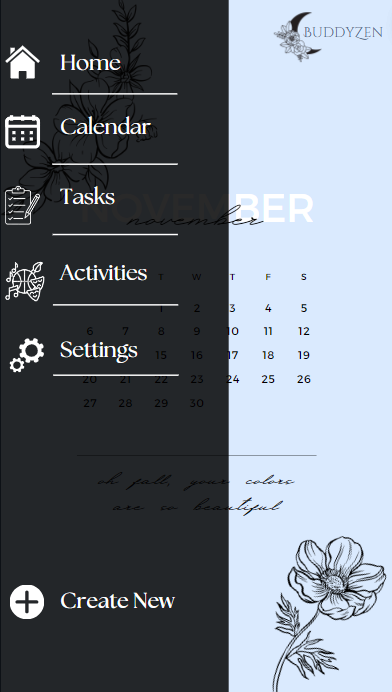


Figure 2: Sidebar View for BuddyZen (design in Canva)



Figure 3: Central Calendar View (Design in Canva)

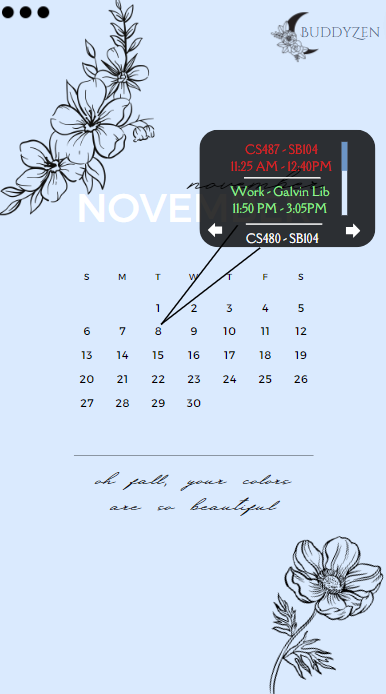


Figure 4: Scrollable event popup view (Design in Canva)

### 

### 

### 

### 3.2 Task Management

Within the sidebar, the UI sketch details a simple and intuitive task management interface. Users can add new tasks with a clear “Create New” button, and each task can be categorized by type (assignment, personal, club-related). Priority levels are easily adjustable, and a checkbox allows users to mark tasks as complete. The interface also supports drag-and-drop functionality for effortless task rearrangement (Figure 5).

### 3.3 Event Details

Clicking on a calendar event opens a detailed view. For classes, it shows the course name, location, and professor details (Figure 5). Assignment events display due dates and associated courses (Figure 6), while extracurricular events provide information on the club, location, and time. Users can quickly edit or delete events directly from this view, enhancing the overall user experience.

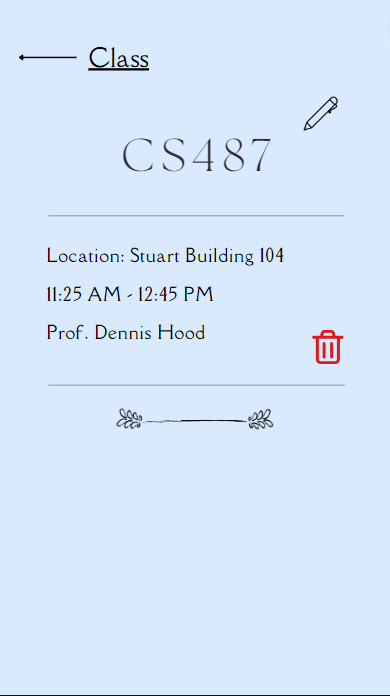
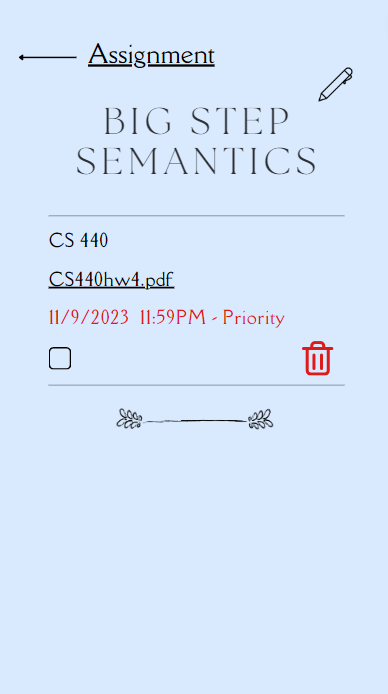


Figure 5: Detailed View for a class (Design in Canva)

  
Figure 6: Detailed View for assignment (Design in Canva)

### 

### 

### 

### 3.4 Notification Center

The UI sketch includes a Notification Center accessible from the top navigation bar. Users can view and manage reminders, with options to snooze or mark them as completed. The notifications are color-coded (red for high priority are almost due events, green for soon-to-be-due events, and white for others) for easy identification, maintaining consistency with the calendar events.

### 3.5 User Profile and Settings

In the Settings tab, users find their profile information and customization options. They can set preferences for notification timings, choose app themes, and manage account details. The UI sketch ensures a straightforward and visually appealing layout for a positive user experience (Figure 7).

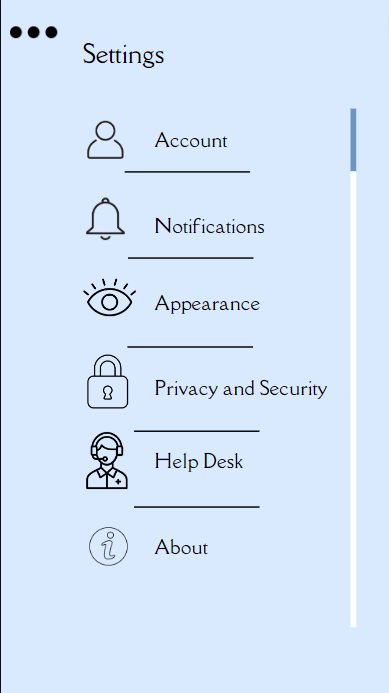


Figure 7: Settings View (Design in Canva)

### 3.6 Feedback Loop

Integrated into the UI is a subtle feedback button allowing users to share their experiences, report issues, or suggest improvements. This feature reinforces BuddyZen's commitment to continuous enhancement through user input.

### 3.7 Responsive Design

The UI sketch accounts for responsive design principles, ensuring a seamless experience across various devices. Whether on a desktop or mobile device, the interface adjusts intelligently, optimizing space and maintaining usability.

### 3.8 State Transition and Workflow Diagrams

First of all, the app BuddyZen has the login state, which requires the user to login. If the user enters the correct username and password, then the state transitions to Dashboard State. If either of them is incorrect, then the user remains on the login page until the correct username or password is provided. Note that the password can be changed in case the user forgets the password, in which situation the login state will enter into a paused state until a new password is set. As a result, the username and password can be changed under special circumstances, wherein the specific data attributes for the respective fields will be altered in the database. These new values will be expected when the user tries to login again.

The next state is the Dashboard state which consists of a calendar in the center. When the user clicks on a day in the calendar, a popup appears with the list of events on that day in order of timing. When the user clicks on a specific event, the state changes to Event Editing state/Detailed View state where the specific event can be viewed in detail, edited or deleted. It remains in the same state until the back button is clicked, upon which it goes back to the Dashboard state. When editing, it turns into an Edit state which lets the user edit the event details. As soon as the user saves changes, the state reverts back automatically to Dashboard state and automatically updates to the calendar. If an event is deleted, the state automatically reverts to Dashboard state and the changes are saved immediately.

The Dashboard also has a sidebar which is accessed via the three dots on the upper left hand corner. Upon clicking, the Dashboard state pauses to show the sidebar. Upon clicking Home in the sidebar, we are taken back to the Dashboard state. The Calendar button lists all events of the month in a list view regardless of the type (class, club or assignment), and displays the type immediately after the title of the event for clarity. The Tasks button filters out the other two types leaving us with only assignments, displaying the assignments in order of when they are due. The Activities button works similar to the Tasks button, except that it displays all club activities and classes. Upon clicking either the Calendar, the Tasks or the Activities button, the state changes to a List View state which filters out calendar events based on what was selected. Upon pressing the back button, the state changes back to Dashboard. Upon selecting any of the events in specific, the state transitions to Detailed View and works the same way as mentioned above.

The sidebar has two last buttons, the Settings button and the “create new” button. The settings button changes the state to the Settings state, which allows the user to change the settings of the app (as the name suggests). In case the username or password is changed, then the app reverts back to the Login state and the user must re-enter the changed username/password. This ties back into the “forgot password” circumstance, where the password is changed in the settings state and the user must re-enter their credentials to login. To return back to Dashboard state, the user must enter the three dots at the top left hand corner and select Home to go back to the Dashboard state.

The last button is the “create new” button. Upon pressing it, the state changes to the Add New Event state. The user is prompted to enter the specific details of each attribute for the new event, including but not limited to the name, time, location, instructor (if applicable), priority, notification setting, description, file attachments, and so on. After that, they are prompted to save changes. If the user cancels, then they are immediately reverted back to the Dashboard state with no updates made. If the user saves, the state changes to Dashboard state and the new event is sorted based on the time and added to the calendar on the Home page.

The workflow and state transition diagrams are given by Figure 8.

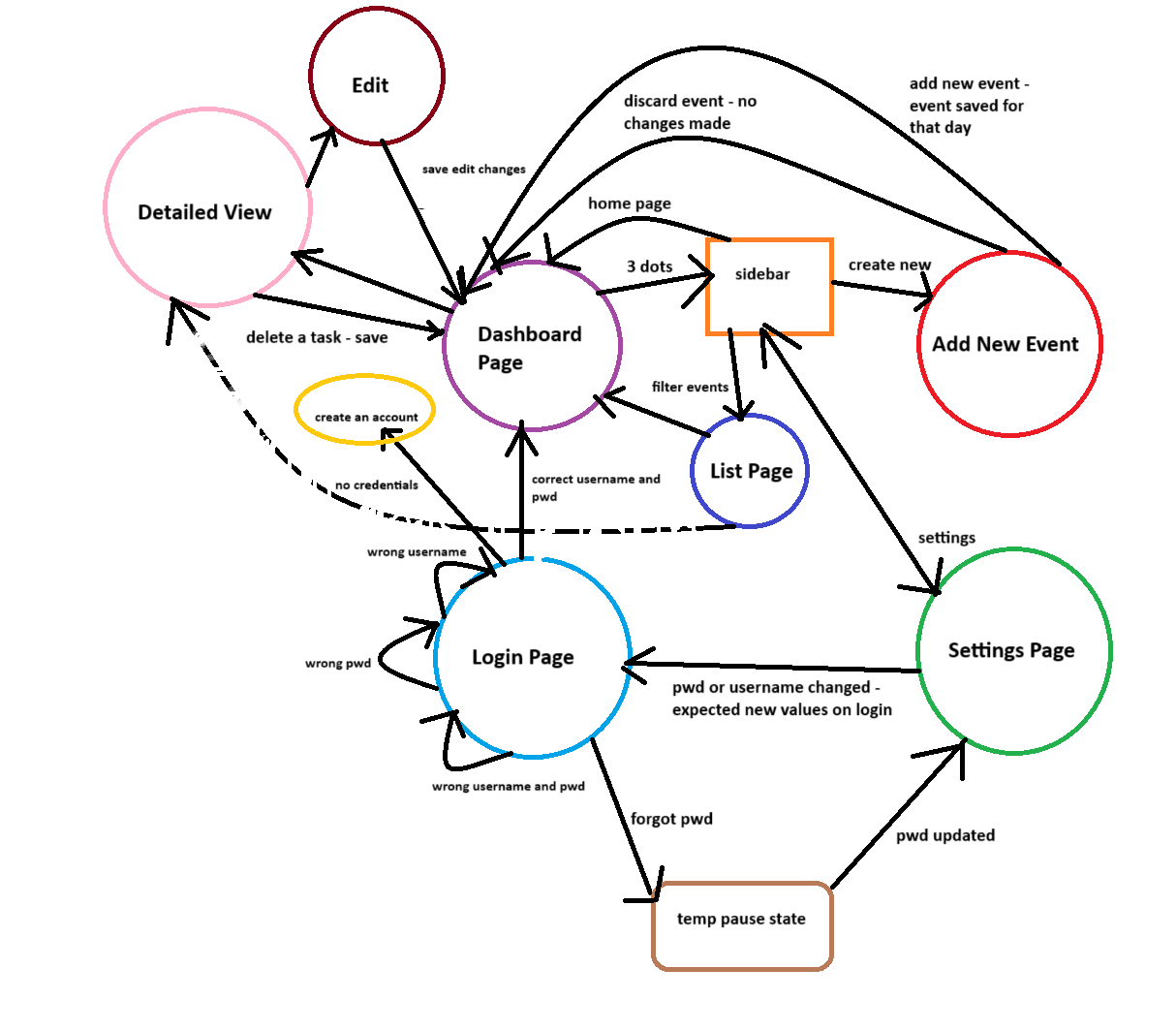


Figure 8: Workflow and State transition diagrams for BuddyZen

4. Algorithms

Below is the rudimentary pseudocode for BuddyZen that envisions our transition state models and workflow diagrams roughly. Note that upon implementation in a dedicated software environment (like Android Studio, Swift, Flutter, etc or a database provider like AWS, Azure, MySQL, etc) much of this pseudocode is subject to change depending on the functionality requirements.

class User {

var username

var password

var events

var settings

function login() {

# Logic to authenticate the user

}

function changePassword(newPassword) {

# Logic to change user's password

}

}

class Event {

var name

var time

var location

var instructor

var priority

var notificationSetting

var description

var fileAttachments

function editEvent() {

# Logic to edit event details

}

function deleteEvent() {

# Logic to delete the event

}

}

class Calendar {

var events

function viewEventsByDay(day) {

# Logic to filter and display events for a specific day

}

}

class Settings {

function changePassword(user, newPassword) {

# Logic to change the user's password

}

}

class AddNewEvent {

function createNewEvent(user, eventDetails) {

# Logic to create and save a new event

}

}

class BuddyZenApp {

var currentState

var currentUser

var currentEvent

var calendar

function transitionToLoginState() {

# Logic to set up the login state

}

function transitionToDashboardState() {

# Logic to set up the dashboard state

}

function transitionToEventEditingState() {

# Logic to set up the event editing state

}

function transitionToDetailedViewState() {

# Logic to set up the detailed view state

}

function transitionToListViewState() {

# Logic to set up the list view state

}

function transitionToSettingsState() {

# Logic to set up the settings state

}

function transitionToAddNewEventState() {

# Logic to set up the add new event state

}

function handleLogin(username, password) {

# Logic to handle the login process

}

function handlePasswordChange(newPassword) {

# Logic to handle password change

}

function handleDaySelection(day) {

# Logic to handle the selection of a day in the calendar

}

function handleEventSelection(event) {

# Logic to handle the selection of an event

}

function handleEventEditing() {

# Logic to handle the initiation of event editing

}

function handleEventDeletion() {

# Logic to handle event deletion

}

function handleSidebarClick() {

# Logic to handle the click on the sidebar

}

function handleSettingsChange() {

# Logic to handle changes in settings

}

function handleCreateNewEvent(eventDetails) {

# Logic to handle the creation of a new event

}

}

# Implementation of BuddyZenApp

app = new BuddyZenApp()

# Main function for the Dashboard state

function mainDashboardFunction() {

# Sample interactions in the Dashboard state

app.transitionToDashboardState()

app.handleDaySelection("2023-11-09")

# View event details and edit

app.handleEventSelection(event)

app.handleEventEditing()

event.editEvent("New Event Details")

# Save changes and revert to Dashboard

event.saveChanges()

app.transitionToDashboardState()

# Access sidebar and navigate to Settings

app.handleSidebarClick()

app.transitionToSettingsState()

# Change password in Settings

app.handlePasswordChange("newPassword")

# Revert to Login state after password change

app.transitionToLoginState()

app.handleLogin("exampleUser", "newPassword")

# Access sidebar and navigate to Create New Event

app.handleSidebarClick()

app.transitionToAddNewEventState()

# Enter details for a new event and save

newEventDetails = { "name": "New Event", "time": "2023-11-10", "location": "Location", "instructor": "Instructor", "priority": "High", "notificationSetting": "15 minutes before", "description": "Description", "fileAttachments": "Attachments" }

app.handleCreateNewEvent(newEventDetails)

}

# Call the main function for the Dashboard state

mainDashboardFunction()

5. Data Perspective

The data perspective of BuddyZen and the application's information architecture is meticulously designed to facilitate efficient organization and management of user schedules. The core entities include User, Event, and Calendar, each possessing distinct attributes and relationships. Users are identified uniquely with a `userId` and are associated with personalized settings, while events, ranging from classes to assignments, carry comprehensive details. The Calendar entity orchestrates the temporal organization of events, forming a vital link in the system. Object models, encompassing User, Event, and Calendar, provide a structured representation of the application's data entities. This data perspective illuminates how BuddyZen seamlessly handles user authentication, event creation, calendar organization, and settings management, culminating in a robust and user-centric information architecture.

Data Entities and Relationships:

* User
  + Attributes
    - `userId` (unique identifier)
    - `username`
    - `password`
    - `events` (list of associated events)
    - `settings` (user-specific settings)
* Event
  + Attributes
    - `eventId` (unique identifier)
    - `name`
    - `time`
    - `location`
    - `instructor`
    - `priority`
    - `notificationSetting`
    - `description`
    - `fileAttachments`
  + Associations
    - Many-to-One relationship with `User` (each event belongs to one user)
    - Many-to-Many relationship with `Calendar` (an event can be associated with multiple days)
* Calendar
  + Attributes
    - `calendarId` (unique identifier)
    - `events` (list of events for specific days)
  + Associations
    - Many-to-One relationship with `User` (each calendar belongs to one user)

Object Models

* User Object:

User {

userId: string

username: string

password: string

events: List<Event>

settings: Settings

}

* Event Object:

Event {

eventId: string

name: string

time: datetime

location: string

instructor: string

priority: string

notificationSetting: string

description: string

fileAttachments: List<string>

user: User

calendar: Calendar

}

* Calendar Object:

Calendar {

calendarId: string

events: List<Event>

user: User

}

Data Flow

1. Login and User Management
   1. Upon login, the system verifies the user's credentials and loads the user's data.
   2. User data, including events and settings, is fetched based on the logged-in user.
2. Event Management
   1. Users can create new events, and the system assigns a unique `eventId` to each.
   2. Events are associated with the user who created them and may be linked to specific days in the calendar.
3. Calendar Management
   1. The calendar associates with a user, allowing the system to organize events based on specific days.
   2. Events can be viewed and managed through the calendar.
4. Settings and Password Change
   1. User-specific settings, including password, are stored in the `Settings` object.
   2. When the password is changed, the system updates the user's password attribute in the `User` object.

Workflow

1. Login Workflow
   1. User enters credentials.
   2. System verifies the login and loads the user's data.
   3. User-specific data is fetched, including events and settings.
2. Event Creation Workflow
   1. User creates a new event with specific attributes.
   2. The system generates a unique `eventId` and associates the event with the user.
   3. The event is added to the user's list of events and linked to the appropriate calendar.
3. Calendar View Workflow
   1. Users can view their calendar, which displays events organized by days.
   2. Clicking on a day loads associated events using the `Calendar` object.
4. Settings and Password Change Workflow
   1. Users can access and modify their settings.
   2. Password changes trigger updates to the user's `password` attribute in the `User` object.

Data Storage

* Database: The system likely uses a database to persistently store user data, events, and settings.
  + Tables: User, Event, Calendar
  + Relations: Foreign keys link tables based on associations.

BuddyZen's information architecture revolves around users, events, and calendars. The object models and data flow highlight how these entities interact, providing a structured and organized approach to schedule management. The workflow and data perspective underscore the seamless integration of functionalities within the app, contributing to a user-friendly experience in schedule organization and task management.

6. Non-Functional Requirements

Multiple design approaches have been taken to meet the non-functional requirements for BuddyZen and make it an application that possesses quality and behaviors that favor the benefit of the user. Some of the non-functional requirements that helped the application become user-friendly are mentioned as follows.

### 6.1 Security

To ensure the security of the data the user inputs and is stored in the application, each user is required to have a login and password. Each password needs to be at least over 8 characters in length including upper and lower case alphabets, one or more digits, and at least 1 symbol. This strategy will reduce the risk of having common and easy passwords therefore reducing the risk of someone hacking into the system with a user’s login. Further, data encryption will be implemented to ensure the security of the user data as it is being transmitted and stored; providing confidentiality to the schedules, assignments, and activities.

### 6.2 Performance

The performance of BuddyZen is ensured by its response time. Whenever the user adds to the planner, they are instantaneously added to the calendar. The response time will be minimal for user interactions and provide for a smooth and actively responsive user experience. Furthermore, the app will be able to handle multiple users at once since users might be sharing their schedules or planning at the same time. This will happen without compromising this performance as the app can handle a large quantity of users at a given time, making scalability a significant part of the app.

### 6.3 Ease of Use

The User Interface is specifically designed to cater to the user and provide a consistent and intuitive approach to navigating through the app. The app has a calming background and the interface is straightforward allowing users to easily click through the buttons and make adjustments and modifications to their planner as they like. Furthermore, through settings, the user can modify the notifications to their personal preferences such as if they would like to receive notifications with sound or not or if they want to mute them altogether.

6.4 Reliability

BuddyZen will be functioning all 24 hours of the day providing the users to access their planners at any time of the day regardless of any time constraints. Furthermore, if the app faces some difficulties in hardware or software it will save the user information in case it crashes but otherwise, it will keep performing while fixing the issue in the backend part of the app.

6.5 Compatibility

BuddyZen will be able to function through major browsers such as Chrome or Safari to support a large audience and diverse user preferences. Further, the mobile responsiveness of BuddyZen will allow it to be accessed and functional on different smartphones or tablets with operating systems such as iOS or Android.

# 7 Risk Assessment and Exception Management

### 7.1 Identification of Potential Failures

A few potential failures that may arise during development and usage are possible design errors. The app consists of features such as a customizable calendar, task tracking, priority settings, and reminder notifications. In order to make it customizable for a user they must create an account. A design error that could arise from creating an account would be security concerns. In order to make the app reliable for the users there can be a priority on the user’s account information. Another design error that could arise would be the effectiveness of the notification reminders. It is noted that notification issues can arise if there aren't consistent notifications for different reminders to the user. This issue may arise during usage of the user when they are able to note if they were not notified of an important task, which can arise from the issue as well as lessen the reliability of the application.

### 7.2 Assessment of Risk Exposure

In order to assess risk exposure there should be different factors to take into consideration that may potentially cause problems. These factors include the overall functionality of the application, data security, the user experience, and the app's overall performance. Risk exposure that may occur in the functionality of the application would be when a user is trying to synchronize their calendar to the application. This action can cause issues such as inconsistencies in the synchronization of the calendar, introduction of delays, and issues with data modifications. Another risk exposure that may arise is the risk of having unauthorized access to users' data. This issue is likely to occur if there is no strong authentication mechanism and a lack of adoption of proactive and holistic approaches to security. Another risk exposure within the application is the user's experience in the application, since providing a bad experience to users causes users to become disinterested which leads to a loss in the application's user engagement.

### 7.3 Risk Mitigation Actions

Specific steps that will be taken in order to reduce the likelihood of potential risks in the application would consist of creating a user-friendly environment for the user interface. By making it user-friendly and making it easy for users to navigate through the application, it will not only fulfill the user’s needs but also allow to attract and retain users to the application. Another specific step that will be taken in order to minimize potential risks is by performing thorough performance testing on the application in order to be able to address possible issues during development, this would then minimize the amount of issues that may arise during usage for users. It is also important to note that in order to tackle the risk exposure of data security it should be taken into account that there should be regular security assessments being conducted but also be able to have strong login security processes like two-factor authentications.

### 7.4 Exception Management Mechanisms

Exception management means handling unexpected events that may arise and being able to adapt the program in order to make everything go back to normal. Now when it comes to mechanisms for exception management, there are specific methods that are able to handle those unexpected events. An exception management mechanism that would apply to this application would be invalid user inputs. The application consists of having users sign up in order to have an individually catered experience, and in order to make sure valid information is being given from the user, there should be an exception that would listen for those exceptions and then give the user a message stating to input proper information. Another exception management mechanism that would be beneficial for this application is user feedback. With user feedback, users are able to report issues that might have occurred in the app with information on diagnosis of the issue. This exception management mechanism will not only be able to improve the application but also be able to get information on issues that are arising through the application from the user's perspective.

# 8. H-C-I and C-C-I protocols

### 8.1 H-C-I protocols

For H-C-I protocols it should be necessary to implement a user interface where users are capable of adjusting their schedules or overall adjusting the system however they feel would fit. Since users are not only capable of changing their schedules, they are also capable of setting up their preferences for notification timings, app themes, and managing account details. This will not only be able to have the system gather the information from the user’s input but also be able to adjust itself as necessary from the information provided by the user from the user interface.

### 8.2 C-C-I protocols

For C-C-I protocols it should be necessary to have the frontend and backend of the system be able to communicate and interact with one another in order to process and manage information being retrieved from the user. In the case of a database such as Azure, we aim to implement this using either Azure functions (using AJAX calls and RESTful APIs) or Java Applets to allow for function calls. Database interactions is a C-C-I protocol that enables users to be able to include their schedules and the notification settings in which the system is able to retrieve that information in order to implement that date to update its calendar and notification system.

# 9. Changes and Additions

In this deliverable we emphasize more of BuddyZen’s features in the summary. We provide a context and system model showing how elements interact with each other within the application. We include user interface sketches that show different menus within the application. The deliverable details a state transition with three main states as well as a workflow diagram. This deliverable provides a more comprehensive view of the application's internal workings, it describes the algorithmic perspective using pseudo code for critical aspects of the application. We also go further into how data in the application is organized, stored, and changed. From the last deliverable, we describe design approaches to satisfy the non-functional requirements: security, performance, ease of use, reliability, and compatibility. We introduce potential design errors like notification effectiveness as well as accounts impacting security with data provided. We also introduce risk exposure and mention a user having a poor experience with the application if synchronization issues occur. Additionally, mitigation steps which include an emphasis on a user-friendly interface and extensive performance testing. The deliverable highlights H-C-I protocols which are the user setting their preferences in the application and C-C-I protocols which are frontend and backend communicating.

# 10. Remaining work

Having established the application requirements in the first deliverable as well as detailed design and planning of the application. The remaining work needed to be done to create the final working prototype would be the development of the application which would consist of coding. Once the development is complete the next step would be to test the application to identify errors or inconsistencies. We would need to make sure that each of the features of the application works together seamlessly. Then we would gather feedback from test users to implement any necessary changes or improvements to the application based on the feedback. We would also need to take security measures when it comes to user data and make sure it's protected. We would need to make sure the application is performing well, not slow or unresponsive during peak usage. Finally, the application would get launched and be continuously updated for bugs or user feedback.