

Time algorithm manipulate

Live a More Balanced Life.

G-20 The Priority Project

Tyler Farkas 130673230 & Thomas Shapton 130754710

# Table of Contents

Title Page

Table of Contents

Requirements

Introduction: Problem, Purpose, Scope, Perspective, Constraints, and Non Functional Requirements……………………………………………………………………………………………………………..3-4

Use Cases: Signup, Login, Add, Edit, Check…………………………………………………………………..5-9

Use Case Diagram & Relationships Among Use Cases………………………………………………….10

Team Organization & Time Management…………………………………………………………………….11-13

Analysis

Introduction………………………………………………………………………………………………………………….14

Identifying Concrete Objects…………………………………………………………………………………………15-21

Entity Objects

Boundary Objects

Control Objects

Use Case Sequence Diagrams………………………………………………………………………………………..22-24

Design

Introduction…………………………………………………………………………………………………………………..25-26

System Overview……………………………………………………………………………………………………………27-29

System Architecture: Overview, Subsystem Diagram, Cohesion & Coupling, Detailed

Subsystem Responsibilities………………………………………………………………………………..30-37

Data Design: Data Flow Diagram, Data Dictionary…………………………………………………………..38-40

Interface Design

Appendices

Introduction

Problem

With all the distractions in the world it is difficult to feel in control of your day. Whether you’re on your phone texting a friend, playing an application, thinking about loved ones, there are always thoughts on your mind. What am I doing next? What has to be done? How much time do I have leftover? Can I afford to sacrifice time in one place to replace it in another? Students are told early in their university careers to create a daily schedule. It is an amazing way to organize your day and understand what tasks need to be done. The problem with a schedule is it takes time to create and lacks a clear overall picture of your situation. Individuals don’t know how much free time they have, and other significant information. All this time adds up. It takes at least 15 minutes a day, and if you want to calculate other variables yourself it’s significantly more. That’s 105 minutes a week, which could be spent elsewhere. Time is the most valuable resource so why don’t we take control of it, and value it to what it is actually worth.

Purpose

An overall outline of the TAM system. Specifications of each section are included to assist individuals in understanding the scope of the project. Being able to use each phase to assist in the development of the system is a must. Time is limited and knowing how each phase is implemented is the most efficient way to develop a program.

Scope

TAM is an android application that is a day planner, week planner, and month planner. It keeps track of time and prioritizes more significant events to ensure enough time to be successful with all the events. It schedules the events for you to save time and ensure a healthy day with enough time for sleep, buffer time between events, and exercise.

Perspective

TAM is an application with a full-stack development process. The front-end platform allows the user to login to their account, enter events, edit events, and view events. It displays the information for each user to update their schedule accordingly. All information will be taken and stored in the back-end platform using MySQL in a database. To process this information the application and database will communicate via an apache server. As an application such as this user information privacy is critical. The database ensures information security for user events, login information, and their general schedule. The only way for information to be accessed is through the server. Thus in order to use the application an internet connection is necessary to access the server and update values.

Accessing the database is limited. Only when necessary will information be communicated with the server. We decided to update information on each start up rather than storing local values because of the sensitive information.

Constraints

Due to an internet connection being necessary, if a POST is made without a connection the information may be lost and the user will have to redo the request. If a connection is not made the user won’t be able to login thus stopping any POST requests that may occur.

If at any time the user loses their internet connection they will be logged out. In order to view any pages the user must be logged in. A message will be displayed letting the user know that information cannot be accessed until an application reset occurs.

Non Functional Requirements

The algorithm must be able to take the right events, create the most efficient schedule. This means using all time, prioritizing each event, and accounting for breaks. The elegance of the solution is the most significant aspect, as any person can create a schedule for themselves, and we want to keep editing events to a minimum so it must be as realistic as possible.

The user should be able to easily view their events in a pleasing manner. One quick glance is all it should take to save time, and show the user what there next step is.

Only necessary information is taken from the user, as it ensures efficiency, and user ambiguity. The user must feel secure when giving information. We make sure that the user controls the event name and importance.

Use Cases:

**Scenario #1**: Adding events

**Participating Actors:**

* Ronald Raegan

**Flow of Events:**

1. Ronald Opens main page
2. Ronald goes to Schedule Events
3. He goes to add event
4. He goes to add his work schedule
5. He fills out the details of his acting job
6. He then clicks add Event
7. The system checks if the edit value has any interference with other events
8. It then adds the event to the event archive
9. The system takes the info and sends it to the algorithm
10. Then takes the results to the algorithm and adds it to the users calendar archive
11. He goes to the schedule preview for the month
12. It then takes the users calendar archive to fill out the month schedule
13. He then returns to home page
14. He closes the application.

**Extensions:**

1. If the user made a mistake at any point in time, they can hit the back button to get to the previous page.
2. If the user doesn’t want to add an event they can hit the home button to cancel the process.

**Exit Condition:** The user filled out the required feels without errors, and hits the submit button.

**Scenario #2**: Sign Up

**Participating Actors:**

* Harry Potter

**Flow of Events:**

1. He opens the app
2. He chooses to create an account
3. He is put to the create account screen
4. He writes in a username and password
5. He clicks okay
6. The app then goes and checks if the username and password or okay to use
7. Returns okay and sends harry to the add an login page
8. He then exits

**Extensions:**

1. The user must input values with length greater than 6 and less than 20.
2. The user must replicate their password to assist them in remembering it.

**Exit Condition:** A message is shown to the user that the account is created successfully and the user exits or logs in.

**Scenario #3**: Login

**Participating Actors:**

* Harry Potter

**Flow of Events:**

1. He opens the app
2. He selects the username field
3. He writes in a username
4. He writes in his password
5. He clicks okay
6. The app then goes and checks if the username and password are correct
7. Returns okay and sends harry to the home page.
8. He then exits

**Extensions:**

1. A message is displayed if a login attempt fails.

**Exit Condition:** The user clicks the submit button and enters the program, or exits the program.

**Scenario name**: Check schedule

**Participating actor:**

* Marry Poppins

**Flow of Events:**

1. She opens main page
2. She clicks on calendar
3. The calendar directory is brought up
4. She clicks to view the week
5. The system goes to the list page
6. The system checks the users calendar archive
7. It pulls up all events happening in the current week
8. It then puts this information in list format on the list page
9. She then ask to see the day planner
10. The system removes all values that not are from the day selected
11. It then displays this new information
12. The user then changes the date they wish to see
13. The information not in this date is removed
14. It then checks if the user archive for events that fit in this new timeline
15. It adds in the new events
16. It displays the events
17. The user then ask to display the events in calendar layout
18. The display images are removed
19. The new calendar background is added
20. Then the list of events are added in
21. The user goes back to the homepage
22. The user exits the app

**Extensions:**

**Exit Condition:** Once the schedule is seen the user is finished viewing the updated schedule.

**Scenario name**: Edit schedule

**Participating actor:**

* Nicholas cage

**Flow of Events:**

1. Enters Homepage
2. Clicks Scheduling events
3. The page is switch to the scheduling event page
4. Then clicks edit events
5. Opens a page with a list of all events
6. Nick scrolls down to find work schedule for Face off and clicks it
7. A page of info on the work schedule is then displayed
8. He changes the work time
9. He clicks ok
10. The system checks if the edit value has any interference with other events
11. If not it deletes the user calendar archives
12. Then it writes over the part with the edit work info in the event archive
13. It plugs in the info from the event archives into the algorithm
14. The output from the algorithm is then inputted to the calendar archive
15. It then switches the page to the calendar page and displays a list of events for the month from the calendar archive
16. He then returns to the homepage and exits the application.

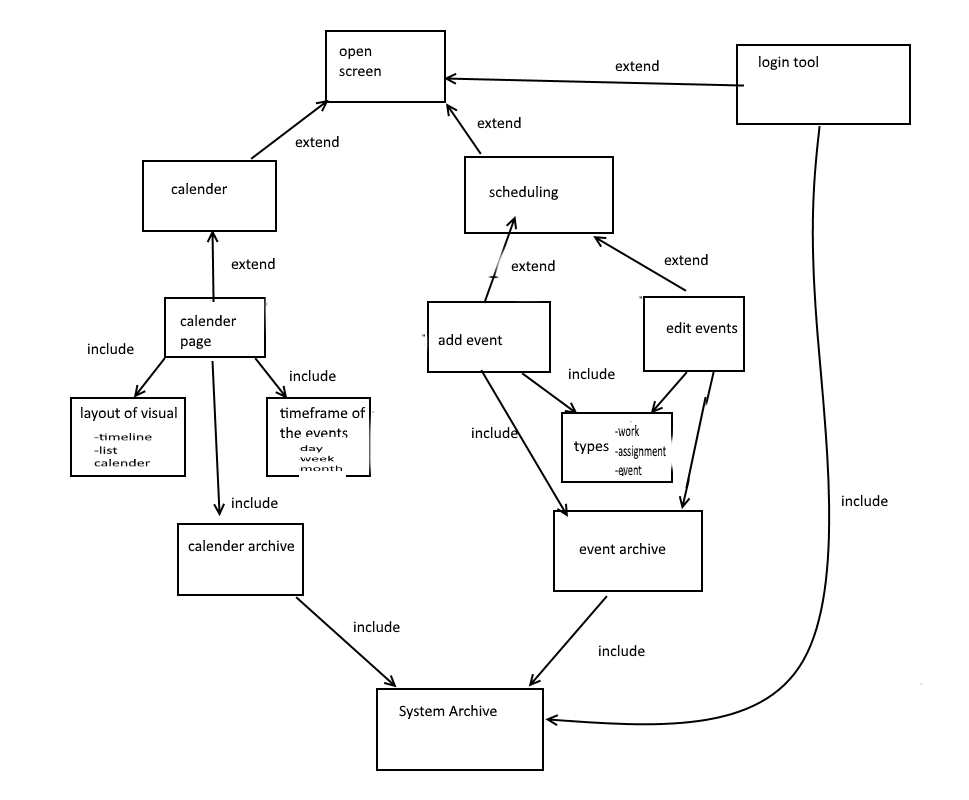
**Extensions:**

**Exit Condition:** Once the event is edited, the event is changed accordingly and a message is displayed to the user.

Use Case Diagram



Identifying Relationships Among Use Cases



Team Organization

As group 20 consisting of Thomas Shapton and Tyler Farkas we plan to create a functional Day planner. Our requirements given by our instructor Dr. C. Hoàng we will create a more interesting spin on the average device of a day planner. Using such languages as HTML, JavaScript, and PHP we will design an object oriented project with great detail. We will also be using MySQL to act as our database and Xampp to run a local server for testing purposes. Our ideal end product is a perfect algorithm to plan the users schedule thus extensive testing is necessary. This creates emphases on the Design and Implementation phase. The team must work together to design effective code.

With two people we have the luxury of not having too many voices. Therefore a democratic process will be the most efficient when making decisions on designing the project. However, having dueling schedules makes it hard to work on the project as a team. The solution to that is to have phase leaders. A phase leader will take on most of the work for a single phase therefore whichever team member has the lighter schedule will be the phase leader.

The trick to our project is certain builds will require a lot more time than others and tracking time will be essential to our project completion. The two biggest parts is a login system and a way for the user to add an event. We will split these two parts between us and use a divide and conquer method. Taking on this project one step at a time is beneficial in order to focus on the bigger tasks of the assignment and not get caught up with minor complex pieces. Implementing the skill matrix will assist us in getting the most out of each individual as we don’t have a large enough group to afford wasting talent.

Time Management

Project Timeline:

*Meetings are kept track of by the appropriate phase leader.*

* **Saturday, October 31st:** Meeting to decide Work Division, and **Requirement Phase**.
* **Tuesday, November 3rd:** Meeting to discuss goals for **OO Architectural Design Phase**.
* **Friday, November 6th:** **Track progress** of OO Analysis Phase Completion.
* **Tuesday, November 10th:** Meeting to discuss builds **OO Design Phases**.
* **Friday, November 13th:** **Track Progress** of OO Design Phase.
* **Tuesday, November 17th:** Meeting to determine **Final Version of Phases** and start of Implementation Phase.
* **Friday, November 20th:** **Review Completed Report**, Make Revisions when applicable.
* **Tuesday, November 24th:** Meeting to track builds **Implementation Phase**.
* **Friday, November 27th:** Share progress with group partners, **Integrate code**.
* **Tuesday, December 1st:** **Track Progress** and Integrate Code.
* **Friday, December 4th:** Final Project touch ups. **Finalize Completed Report**.
* **Monday, December 7th:** Finalize Code and **Package Project**.

Skill Matrix

|  |  |  |
| --- | --- | --- |
| Participant  TASK | Thomas | Tyler |
| Requirement Phase | Secondary Responsibility | Primary Responsibility  Secondary Responsibility |
| Architectural Design Phase | Primary Responsibility  Secondary Responsibility | Secondary Responsibility |
| Home Page Interface Build | Primary Responsibility  Secondary Responsibility | Secondary Responsibility  Interest |
| Login Archive Build | Secondary Responsibility | Interest |
| Event Interface Build | Interest | Primary Responsibility  Interest |
| Algorithm Implementation Build | Primary Responsibility | Secondary Responsibility |
| Maintenance Phase | Primary Responsibility | Primary Responsibility |
| Software Testing | Secondary Responsibility  Interest | Secondary Responsibility |
| Documentation | Primary Responsibility | Primary Responsibility |
| Server Side Implementation | Primary Responsibility | Secondary Responsibility |
| Communications | Secondary Responsibility | Primary Responsibility |
| User Interface Design | Primary Responsibility | Secondary Responsibility  Interest |
|  |  |  |
|  |  |  |

Analysis

Introduction

This section describes the different Entity objects, Boundary objects, and Control objects in our system. It explains how each individual object is incorporated into each use case. Each use case is looked at individually and lists of objects used are shown. The analysis phase is performed in order to look at the underlying processes of the system. In order to avoid having forgotten parts sequence diagrams are created to combat use cases.

Each use case is provided with a sequence diagram to fully explain how the application runs on an underlying aspect. Each communication between levels is shown for the use cases in the sequence diagram to ensure an easy development process.

All objects will be explained on a general framework level. Not all objects are listed because they are either irrelevant or do not assist in the explanation of the use cases.

Overview of Objects

|  |  |  |
| --- | --- | --- |
| Entity | Boundary | Control |
| * User * User Database * Event Database * Schedule Database | * Phone screen * Touch screen * keypad | * Login System * Add/edit scheduler * View schedule * Algorithm |

Use Case Sign Up

Entity objects for the Sign Up use case

User

* User will select the app and navigate to create an account. User will use the phones keypad to type in a username/password. User will then select submit account. Finally User selects home by clicking the screen and exits the app.

User Database

* Contains all user’s password, username and account numbers. It will be used by the Login Object to check for current user info and add new user info.

Control objects for the Sign Up use case

Login System

* Initiates when the user clicks submit. Reads the information from the text box on the screen. Then checks that the inputs fit the requirements to fill username and password. After it will check in the systems archives to make sure the username has never been used by another account. It will then add the information to the archives and move the screen to display the home screen.

Boundary objects for the Sign Up use case

Touch Screen

* The user will be able to click on
  + A “Create Account button” to open a screen and create an account
  + Textboxes are used as input to fill in a password and username
  + A submit button is clicked to confirm and submit an inputted password with the Login control object

Phone Screen

* Displays the create account information to the User. Information consists of filling out a password, username and restrictions displayed accordingly.

Keypad

* Used to type in a password or username

Use Case Login

Entity objects for the Login use case

User

* The User types a password and username, then clicks submit to activate the Login object.

User Database

* Contains all user’s password, username and account numbers. Accessed by the Login Object to check if submitted username and password exist within it.

Control objects for the Login use case

Login

* Initiates when the user clicks submit. Reads the information from the text box on screen to make sure the User filled out the required fields. If so it will check if the given username and password exist in the User Archive. If applicable it sends the screen to the Users homepage.

Boundary objects for the Login use case

Touch Screen

* The user will be able to click on:
  + A “Submit” to call the login object to confirm both username and password
  + Textboxes to fill in a password and username

Phone Screen

* Used to display information to the User where they can click to Submit.

Keypad

* Allows the user to type in a password and username.

**Note: The following cases all happen after the login case and will initiate at the home screen of the User.**

Use Case Add Events

Entity objects for the Adding events use case

User

* User selects schedule event button on the screen. Now they can choose if they would like to add an event. They will be responsible for selecting what type of event they wish to add. Following the selection they will fill out the required information for that event architecture type. Once the User is finished a submit button will be pressed and the event will be added accordingly. After they can select the homepage button or add another event.

Event Archives

* The first check by the add/edit scheduler object, checks if an event doesn’t interfere with existing events. Then is written to by the Add/edit scheduler with a new event.

Schedule Archives

* Cleared by the Algorithm then rewritten with new inputs.

Control objects for the Adding events use case

Add/edit scheduler

* Initiates when the User clicks add event. Determines what type of event the user wishes to add and what the user inputted. Displays a page with all the necessary inputs from the user to describe the event. Then waits until the User selects add for the specific event. Firstly the scheduler will go through all the inputs to make sure each field is filled out correctly. If so it will find the users event archives and fill in the added event. Then it will initiate the Algorithm. Once the algorithm is done it will change the screen to the calendar screen.

Algorithm

* Clears out the original Schedule archives. Then will read from the Event archive and output the results to the Schedule archive. Then it notifies that it’s done.

Boundary objects for the Adding events use case

Touch Screen

* The user will be able to click on
  + A button to navigate to the schedule event page
  + A button to navigate to the add event page
  + A list which the user can click and select from to pick the type of event
  + A button to go an create event
  + A few dialog or list sections the user can click on to define an event
  + A button to add event Add/edit scheduler

Phone Screen

* Labels on buttons for the user to be able to navigate from page to page. Labels next to text boxes and list bars for the user to know what to fill in.

Keypad

* Allows User to fill in textboxes with necessary information belonging to the event being added.

Use Case Check schedule

Entity objects for the Check schedule use case

User

* The user will select calendar on the screen. Then they will be responsible for choosing what the layout should be for the event (time frame: day, week, or month – display: calendar, list, or timeline). Then the events are displayed in the format they selected.

Control objects for the Check schedule use case

View schedule

* When the user clicks on the display schedule this control object is called. It starts with pulling up all events with dates coming in the time frame that the user has requested. Each event it pulls up will be displayed with the format the user has requested (calendar, list, time line).

Boundary objects for the Check schedule use case

Touch Screen

* The user will be able to click on
  + A button to navigate to the calendar page.
  + Two list that contain type of display and time frame of display.
  + A button to display schedule which initiates the View schedule control object.

Phone Screen

* Labels on buttons for the User to navigate from page to page. Labels next to text boxes and list bars for the user to know what needs filling. Once the View schedule control object has given the necessary information the users schedule is displayed

Use Case Edit schedule

Entity objects for the Edit schedule use case

User

* The User will then select schedule event on screen. Now they can select to edit an event. They then can search through a list of all events with a scroll bar and a search box. After the User can select an event by clicking on it in the list. They can then refill the information in the event and clicks submit.

Event Archives

* The first check by the Add/edit scheduler object to check if an event doesn’t interfere with existing events. Then is written to by the Add/edit scheduler with a new event

Schedule Archives

* This is read by the view schedule object to find all to be events. This will be cleared by the Algorithm then rewritten with new inputs.

Control objects for the Edit schedule use case

View schedule

* When the User clicks on the edit event page this control object is called. It starts by pulling up all events with dates coming after the current time from the Users schedule archives. Each event it pulls up it displays with a title and time.

Edit scheduler

* Initiates when the user clicks on an event to edit. Pull up the editable information from the Users event archives from the event and display it in an editable way. It waits then until the user says his finish editing. It will first go through all inputs to make sure they are filled out correctly. If so it will then find the users event archives. It will find the edited event and replace the information. Then it will initiate the Algorithm. When the algorithm is done it will change the screen to the calendar screen.

Algorithm

* It gets called by Add/edit scheduler. When it is called it clears the user’s schedule archive. It then pulls up the user’s event archives. It will use this and other values it has in its program to design a schedule. It will write the results in the user’s schedule archive.

Boundary objects for the Edit schedule use case

Touch Screen

* The user will be able to click on
  + A button to navigate to the schedule event page.
  + A button to navigate to the edit event page.
  + A list of coming up events which the user can click and select from to pick an event to edit.
  + A few dialog or list sections the user can click on to define an event.
  + A button to edit event which initiates the Add/edit scheduler.

Phone Screen

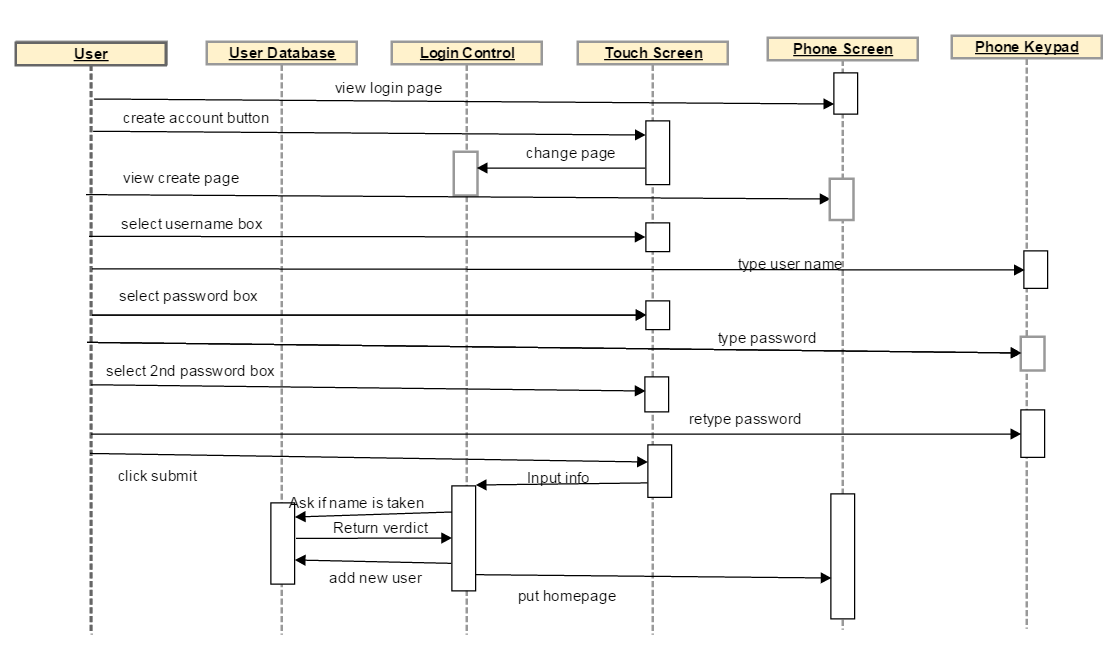
* Labels for the user to navigate from page to page. Labels next to text boxes and list bars for the user to fill in. There will be a list of all events the user will be able to edit given from the View schedule control object. Also all information in the edit event page will be filled out with information that has already been filled out for that event from the View schedule control object.

Keypad

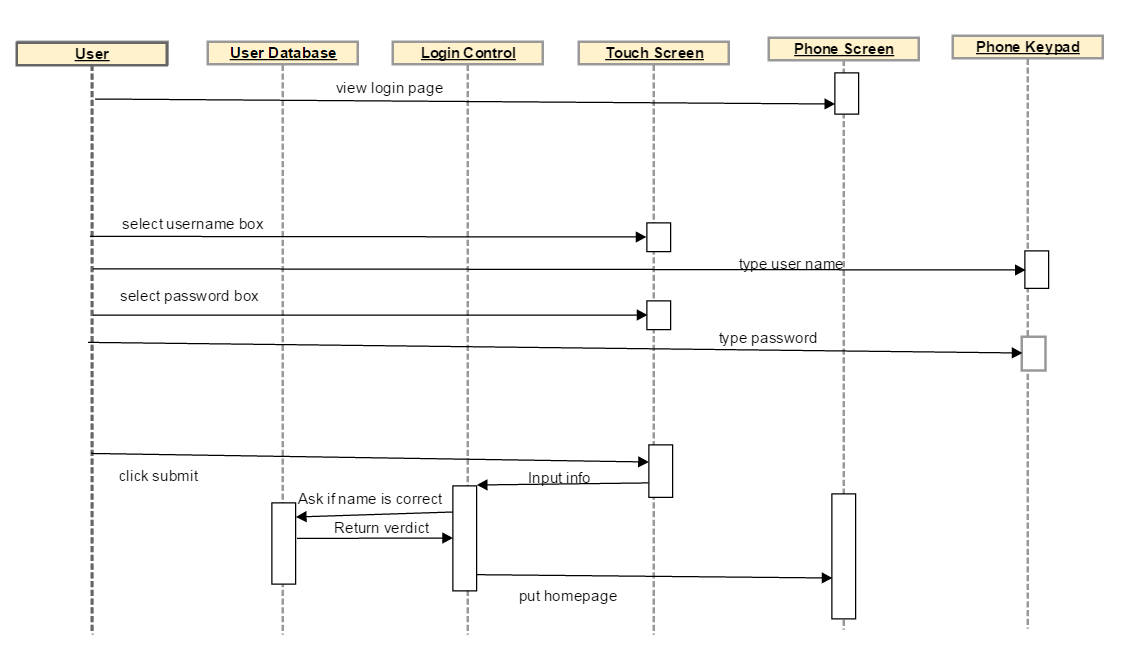
* In order to fill in some textboxes with necessary information belonging to the event being added.

Use Case Sequence Diagrams

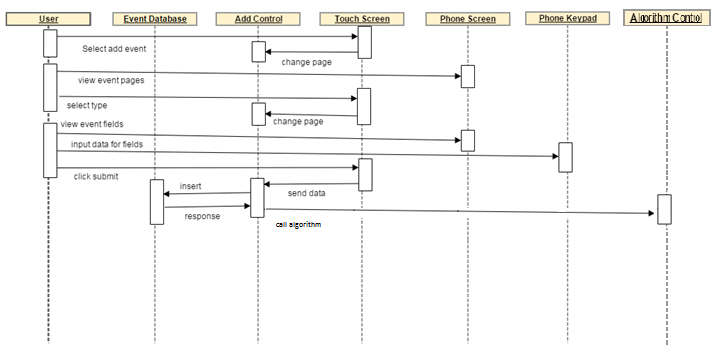
Use Case Sign Up



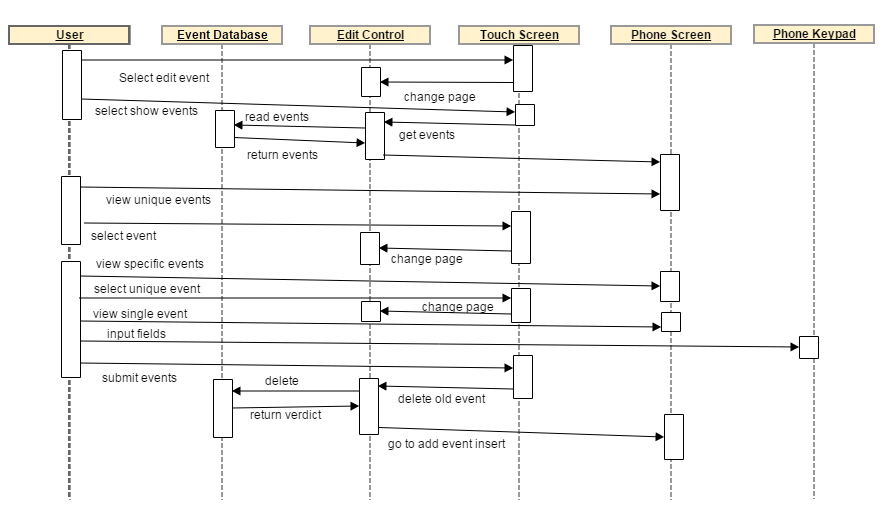
Use Case Login



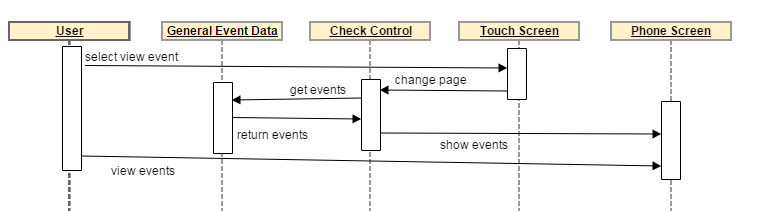
Use Case Add Event



Use Case Edit Event



Use Case Check Event



Design

Introduction

PURPOSE

The reason for this implementation is to give an overview for the client regarding the deep technical aspects of the application. General outlines were previously mentioned for the client, but as the software development team the design document provides a ton of clarity. Once the design document is created, building the application is as easy as following an Ikea instruction manual. The benefit of the design document provides the documentation for teams to decrease the amount of communication needed.

OVERVIEW

Overview is provided in the table of contents. A more in depth look is provided below at each subtopic to provide a full understanding of the applications design document.

**System Overview:** Provides documentation on the entities of the client side, server side, documentation, and conventions used in our application. The client side is literal application that is stored on the phone. The server side deals with the apache server, and MySQL. Further details will be described below. Documentation will be the customs we used to ensure symmetry and enough understanding to describe the performance of code. The convention section will talk about the coding conventions used such as camelCase.

**System Architecture:** This section talks about the major subsystems, their relationships, and provides a diagram to explain them. It will give a thorough analysis on the subsystems do, their responsibilities, and a brief description on how they go about doing them. It takes a look at the use of cohesion and coupling in our program, the dependencies each subsystem has on one another, and the working of a united whole. Each subsystem is working as its own individual system to provide a cohesive unit. Furthermore there is a brief description on how the hardware is mapped to the software and the major policy decisions that went into the development of this application. Major policy decisions are what we felt were the true goals of the applications performance.

**Data Design:** A Diagram and description of the flow of data throughout the system. All data that is transferred is explained in detail through a Data Dictionary. Each arrow represents a flow of data, in which gets processed and delivered to a database. The data design is an essential tool to development as it acts as a manual. Having an overview of all the data that is being transferred gives a true understanding of how the application runs, and what is going on in the back-end of the system. With all the detail provided errors are easily found out, and new content is easily added.

**Interface Design:** This subsystem is the way the user interacts with the application. It communicates with each package to decide what is presented at the given time. With this communication provides fluidity to the program. Further details provided in its individual section to understand how this is done, and what it looks like.

System Overview

TAM runs on the industry standard software. We wanted a structured program with as few flaws as possible. Programs that are thoroughly tested benefit the user experience as hiccups are rarely seen and if they are plenty of information is given online regarding errors that others have witnessed. With all the content out, picking up the language was relatively easy which allowed us to build this application under the time constraint provided. User experience was the greatest benefactor in our decision and we knew we could build a stable application using the technologies we picked.

Client Overview

On the client side of the application we used the cordova JQuery Framework. This provided us with simple elegant DOM manipulation, to update the application as necessary. The JQuery Mobile application provided a framework to use that will keep the users attention on a simple interface design.

HTML was used to communicate with the JQuery mobile framework. Form inputs were used for user fields, buttons were used for browsing, and CSS and JS were used for updating data, and the drawing element to the screen.

We decided on using a navbar to browse between interface pages as it adds elegance and simplicity to the way it was programmed. It benefited communication between partners because we decided on this method early; consistencies were easily maintained for the interface.

In order for user friendliness we incorporated a well known plugin to allow the user to pick a date. This plugin is called Datebox and provides a great looking frame, with many utilities to automatically look after assertions, programming to the lowest common denominator, and elegantly providing the appropriate format for our system to read in.

Server Overview

The large portion of our application is dealing with the server, and manipulating the information to be stored in the appropriate way, and easily read out in the right way. This was a difficult task as we wanted to keep in mind memory efficiency, and program speed.

We used the JQuery framework’s AJAX (Asynchronous Javascript and XML) method to communicate with the server and access the appropriate PHP file. Using the POST command of this method information was able to be communicated to the file and returned back. This strategy gave fluidity and elegance to the design.

Once the information was passed to the PHP file, the information was sorted and a connection to MySQL was made. If this connection was successful either a SELECT, DELETE, or INSERT command was given with the appropriate information, and the command performed as expected.

Once finished an echo was transferred back to the Javascript file with the information the subsystem needed to perform its actions.

Documentations

Documentation was kept simple. For methods & functions, we decided on PRE conditions, POST conditions, and a description what the function does. This greatly enhances code readability, and reusability as each group member was easily able to differentiate and understand what each function did.

For documenting general code, a general outline was given, taking the person and walking them through to understand what each part of the code is doing. Documentation such as this wasn’t the best implementation but it greatly increased understanding of processes and loose code.

Variables were documented as well, if initialized the variable type was given, and a brief description given on what it does. This further enhances readability, and reduces the need for communication.

Conventions

In our coding practices we used a number of conventions to maintain consistency throughout the code.

The first convention we used was for variable names, we decided on using the underscore method. This separated each word with an underscore for easy readability and understanding. This method formalized expectations and influenced consistency with the group. It benefited the program by giving a layer of clarity to each variable name which benefited us in the error recovery process. Errors were easily found out and understood due to the showcase of variable names.

Naming collisions when working with a team are bound to happen, using the underscore method greatly reduces the risk as it is easier to read and mark down which were used. Furthermore the greatest addition this convention added is that it made each person of the development team think and rationalize on what each word meant, and only the significant words with great meaning were used. Code was shortened due to the lack of of’s, the’s and other additions to names.

System Architecture

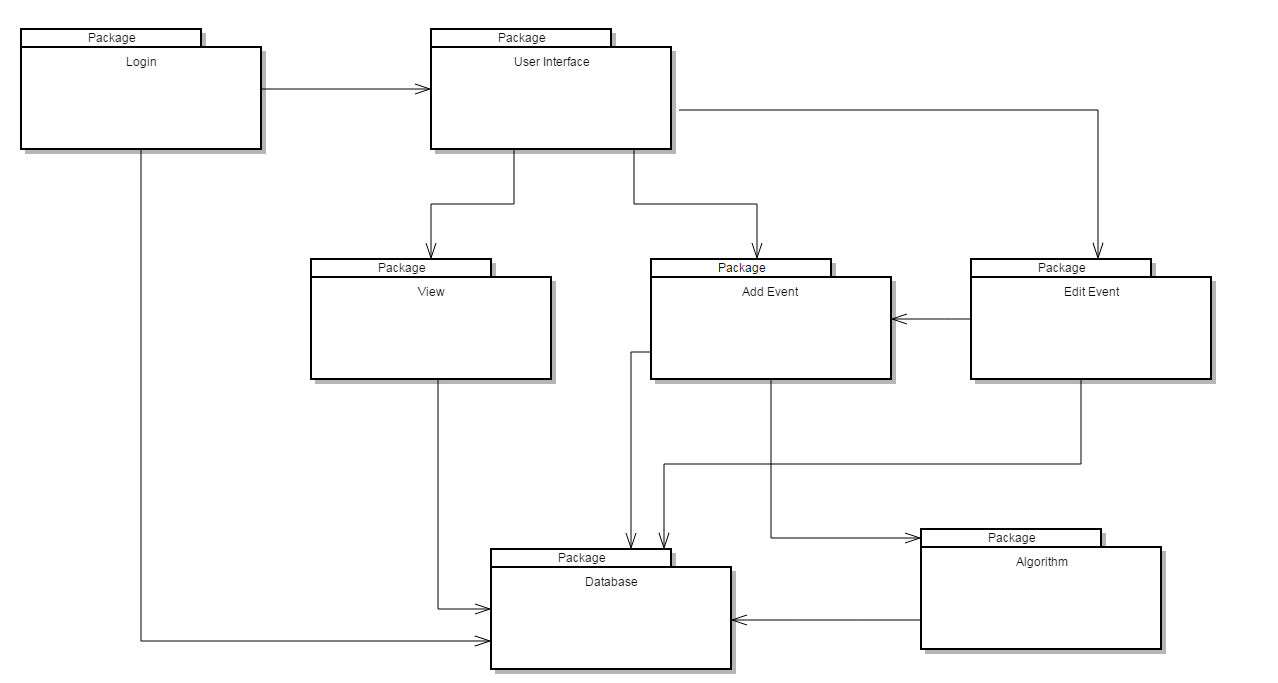
High Level Overview

Our system is composed of 7 different major subsystems that outline how the application performs.

1. Add Event
2. Edit Event
3. Algorithm
4. Login
5. View
6. Database
7. User Interface

Below is the subsystem diagram that outlines the relationship between each major subsystem. This shows the general flow of information.

Subsystem Diagram



As the diagram suggests the user starts at the Login subsystem, which communicates with the database to authorize the account or create the account. The User Interface is shown, where the user can select one of three options. The interface interacts with the rest of the system which further interact with the database. Each other subsystem uses the database to read and write information. The edit event uses the add event subsystem to add the new event once the old one is deleted and edited. The algorithm is run after each add, and the view takes that schedule from the database to view.

Cohesion and Coupling Analysis

Cohesion

Cohesion is the act of a system to be uniform, to form a united whole. The subsystem diagram is an initial overview as to how our system implemented cohesion. As you can see each specific task is done separately. Each major subsystem had functions to perform a specific task. Some functions performed a variety of tasks; this is due to many reasons such as code security, and time constraints. Each subsystem will be explained in greater detail below, as to how cohesion was implemented.

**Add Event:** In order for this subsystem to run, it used 5 different functions. Each function performed a different task. Event\_Page is used to keep track of the current page, used for easy page manipulation. The display function, displayed the appropriate fields depending on the type of event. The switch\_time function changed whether it was a single day event or week event. The submit\_event function ran when the user pressed the submit button, this assigned the priority and called the add\_event function. The add\_event function performed a POST using JQuery to insert the event. All functions used each other to form a cohesive unit and perform a single task, to add an event.

**Edit Event:** Similarly edit event used 5 functions to run. It had the same Event\_Page function, A display event function which showed all the events by reading them in from a POST, organize them and add them to the list. This function violated cohesion as it did multiple tasks but due to variable security and time constraints it had to be done. Display\_event called Single\_name function which shows all the events with the same name for the user to edit. Once a specific button is clicked the onclick links to the single\_event function. This displays the event in the appropriate field, updating all the values. Once the submit button is clicked the old event is deleted, and the add\_event function is called on the new fields. Each function performs a single task, some are larger tasks then others, but cohesion remains intact.

**Algorithm:** Responsible for creating a schedule by reading in all entries from the database. Taking the entries and associated values through a variety of functions that is our own special formula to deliver an exceptional articulate algorithm that displays the user’s schedule. Satisfies cohesion because it is one unified result from many different functions.

**Login:** Each function performs its own separate task, to fulfill the cohesive arrangement. There are a multitude of functions both in PHP, and JS, to check for account names, get\_user names, all doing its own separate functionality.

**View:** Responsible for reading in the schedule the algorithm subsystem creates, and displaying it in a chart for the user to see. Each function performs its own tasks to create a display for the user.

**Database:** Each call to the database is one function that either reads or writes depending on the MySQL call. It is evident that the specific functions are cohesive.

**User Interface:** This subsystem is responsible for all UI changes from each subsystem. Since each UI block is individual it is highly cohesive as each does its own activity.

Coupling

Coupling is how much the subsystems are interconnected. Below we will give a brief explanation of any coupling that occurred in the subsystem and if so how it didn’t harm the applications functionality.

**Add Event:** Interconnected with the user interface. Otherwise functionality is not shared between other subsystems.

**Edit Event:** Interconnected with the user interface. Edit event uses the functionality of Add Event to add an event once it has been edited. Since our program is sequential this saves coding and reuses code to make it more efficient. In the case where our program is not sequential it would be detrimental to the system and coupling could not exist in this case. It is dependent on giving add event the appropriate format and information.

**Algorithm:** Communicated with Add\_Event. After the event is added it is called and performed. The dependency it has is a correct implementation of adding the event. The database must have the events in the appropriate order or else the algorithm will not work effectively.

**Login:** All other subsystems are dependent on login. If the user is not logged in appropriately a logged out page is shown instead of the appropriate page. If the login page doesn’t work correctly it will halt the system and thus crash the application.

**View:** Dependent on the database subsystem similar to the other subsystems. Dependent on the algorithm to be working correctly as it needs a schedule to view. Otherwise limited coupling, and performs its necessary tasks.

**Database:**  Every other subsystem above relies on the database to be working correctly. Since they are all manipulating data in some way it is necessary. The user won’t be able to see any errors at this level because it is hidden deep down.

**User Interface:** All other subsystems except database relies on the user interface to communicate with the user. If the appropriate information is not displayed and accounted for the user cannot continue to use the application.

Detailed Subsystem Responsibilities

Add Event:

The add event subsystem looks after the adding event interface. It starts via the HTML file displaying the information applicable to adding an event. The user fills the information in and a javascript file registers when the user selects the submit button. This is the start of the responsibilities for the Add Event.

**Responsibilities:**

* It is responsible for taking the inputted information from the appropriate fields and converting it into a specific format allowing easy accessibility in the text file. This is done by the add event subsystem running a POST to the php file which inserts the event into the database subsystem.
* Assertions are made to make sure information inputted in fields is acceptable. Depending on the information inputted a function differentiates and sends it with the POST.
* After the event is inserted, the algorithm subsystem is called.

Edit Event:

The edit event looks after the edit event interface. It reads in all entries, stores them and adds appropriate links for the user to click. The user selects the appropriate event registered by a javascript file and the edit event HTML file initializes. The user edits the appropriate fields, and the javascript file registers the user clicking the submit button and the edit event subsystem communicates with the add event subsystem. The add event function is run and appropriate events are inserted.

**Responsibilities:**

* The edit event subsystem is responsible for reading in all entries and displaying the user events with each unique event.
* It is then responsible to show these events as options to the user. Once selected another list is shown for all the events with the same name.
* The edit event is responsible for taking the input from the fields and waits for the submit button to be clicked.
* The old file is deleted and replaced by adding the new fields with appropriate assertions.
* Once clicked, the add event function is called in the add event subsystem to insert the new fields.

Algorithm:

The Algorithm will be the link between the Event Archive and the Schedule Archive. It will have no interface and no accessibility by the user. It will be a subsystem of both the edit and add event archive systems. It will be a PHP coded system that will use 3 functions from other classes. One will be a public function in the Schedule Class that will clear the Schedule archive. The second will be a read function in the Event class that will allow it to use the user’s events as parameter for its function. The last one will be again for the Schedule Class that will allow the results of the algorithm to be written to the Schedule Archive.

**Responsibilities:**

* Clears the old Schedule Archive of the user
* Reads through the Event Archive using a function that will read out events in a specific way to get desired results
* Uses a function to read the constantly changing Schedule Archive which it will use for future event scheduling
* Uses a function to write to the Schedule Archive

Login:

The Login system will handle all responsibilities when a user enters the app. It is written in a mix of JavaScript and PHP. The JavaScript top level functions will be hooked up to the buttons. One for the Create Account Button and one for the Login Function. It will then transfer needed information in the private subclasses which run on PHP. Since the User Archives need to be kept in secrecy everything writing and reading to it will be handled in PHP.

**Responsibilities:**

* It will pull out the given username and password given by the user in a text box
* Determines whether given username and password is appropriate
* When Creating an account
  + Determines if username is taken by another account and if so it will send an appropriate error handle (the user will be asked for a different username).
  + Generates an account number associated with the user
  + Adds in new account to User Archives with users username, password, and account number
  + Creates two files for the user’s Event Archives and Schedule Archives using the users account number to designate them
* When Login into an account
  + Checks if the username and password given is in the User Archives (if not found the user will get an error message to type in a different password). If found it will return users account number

View Schedule:

This System will handle displaying the user’s schedule. The user will use a webpage made from HTML in order to operate it. It will be initiated by a button in the HTML file that will call the top level of the system which is written in JavaScript. This will pull the necessary info from the HTML file given by the user to determine what Subclass function will be used to display the next screen. That will then call its subclasses that is written in PHP. The PHP functions are written to read out the user’s Schedule Archives which is in the server files hence PHP. It will then return that information on the schedule to write out a new page for display.

* It will read out what the user inputted for the time frame that the user wishes to see
* It will read out the user’s Schedule events in order to display the information requested
* It can be able to display the information in a calendar, timeline, or list layout

Databases:

This subsystem handles all the stored data. It communicates with all other subsystems using the software XAMPP to run an apache server and MySQL. MySQL calls are called with a php file whether it is inserts, deletions and selects.

* It is responsible for storing events in one table. Events are differentiated by the user’s username and the single day char. If it is M,T,W,R,F,S, or U it is a week event, if it is a D it is a day event, and finally C for a study event.
* Furthermore it is responsible for storing all usernames and login. This connects with the Login subsystem to insert the new account information or to check old account information.

User Interfaces:

The user interface is an incorporation of all interfaces of subsystems. It is necessary for the user to browse the application. It uses the JQuery Mobile framework to create the actual application. Furthermore it uses the DateBox plugin to record the dates in a user friendly way.

* The user interface subsystem is responsible for reading in all inputs from different fields for the user.
* It is responsible for browsing the application via page links.
* It is responsible for keeping a fluid mobility throughout the application. It does this by incorporating the back and home buttons and using a nav bar javascript file to change between page interfaces.

Mapping Subsystems to Hardware:

We will need the user to navigate the pages using a touch pad to click buttons or mouse.

Form inputs will need a keyboard to enter the information.

The database that stores the event, schedule, and user information will be placed in memory of a server hard drive.

Major Policy Decisions:

**Control Flow:** Control flows are outlined in the prior pictures. A text description is the following.

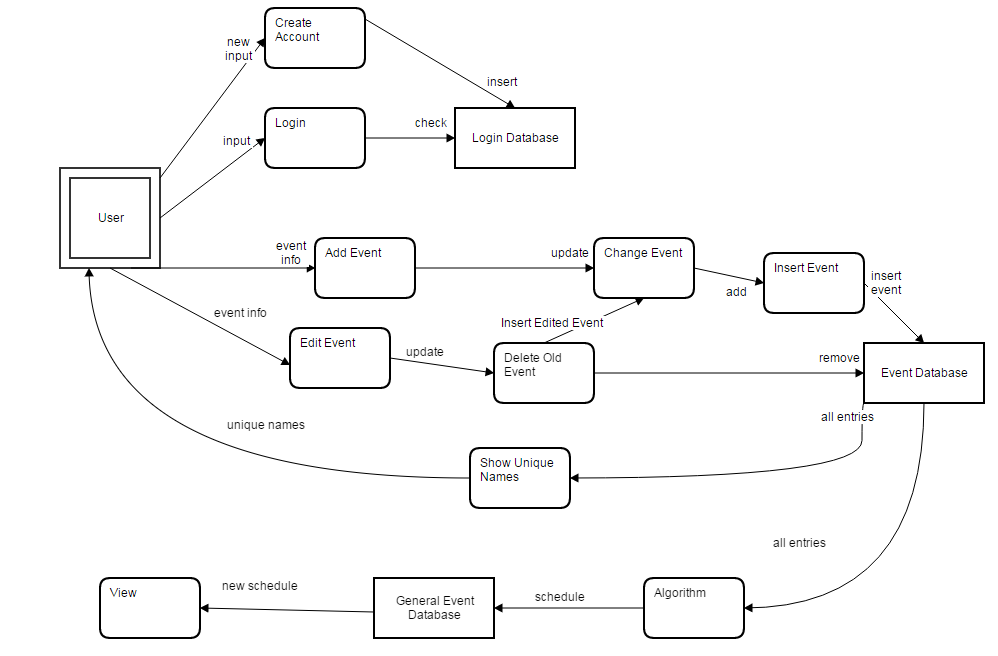
1. User opens application, fills in the account credentials, credentials are sent through PHP to check the SQL database if the login is there. If it is allow user to continue.
2. User has the choice to either view the events or add/edit events.
3. If the user views, a current list is taken from the SQL database and represented via HTML as a list, calendar, and day.
4. If the user selects add, an event form is shown, and waits for a submit button. A PHP script runs and the event is added to the database.
5. Edit takes the list view, asks the user to select an event to edit. Once selected the fields are shown like the add page. Information is then changed. The PHP script is called and searches the table for the old event, and changes the updated event.
6. After the ADD/EDIT the algorithm is called in PHP. It updates the schedule and gives an updated version to the user.

**Access Control:** Control is monitored through the login process. Credentials must go through an APACHE server and be validated. Only once validated can the user enter the process.

**Data Storage:** Using an APACHE server and MySQL a database is created with appropriate tables. One table is used for events, and the other for login credentials.

Data Design

Data Flow Diagram



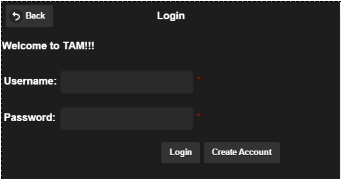
Data Dictionary

|  |  |  |
| --- | --- | --- |
| Name of Data Element | Description | Narrative |
| New input | Username  Password  Repeat Password | Name the user requests.  Password the user requests  A repeat of password. |
| input | Username  Password | Name the user has.  Password the user has. |
| check | Username  Password | Name the user has.  Password the user has. |
| insert | Username  Password  USER\_# | Name the user has.  Password the user has.  Auto Generated Number. |
| Event info | Name  WeekorDay  Day of Week  Dress Code  Importance | Event Name  Is it a Weekly event or a specific day. If weekly input start and end time of each day. If specific input the date, start and end times.  Dress Code – formal or casual accounts for buffer time. If formal buffer time is an hour, if casual buffer time is 30mins.  Importance assigns priority, if Necessary priority is 0, if want priority is 4, if whenever priority is 6. |
| update | Event type | Depending on event type clicked, priority is set to 3 for work or school, 1 for appointment, 0 for vacation, and 5 for exercise. |
| Insert event, add, remove | EVENT\_NAME  PRIORITY  TIME\_END  TIME\_START  DAY  BUFF\_BEGIN  BUFF\_END  USER\_# | Name of event.  Priority value of event 0-6.  End and Start time of event, can be day if applicable, if not date set to 0000-00-00.  Day char value can be M,T,W,R,F,S,U which acts a general day of the week or C for a specific day and D for a study day.  BUFF\_BEGIN and END are either 30mins or 1hour and act as buffer time.  USER\_# is the # assigned to the username to differentiate events. |
| All Entries | Events | All events in database with corresponding USER\_#. |
| Unique Names | uniqueNames[] | Array of values with unique event names. |
| Schedule & New Schedule | Events | Events listed in order of priority, and values for a day. Incorporates sleep time, general time, and specific day time. |

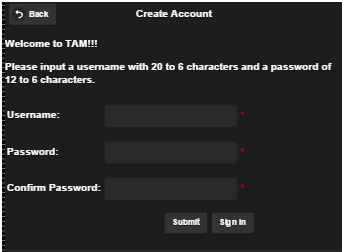
Interface Design

Login Page:

In order to login, you must have a pre-existing account page. The credentials username and password must be length 6-20. If the credentials match the criteria of an account the user number is returned and the user is able to login.

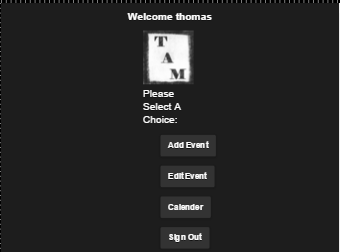


Create Account Page:

Assertions are on each field, the user inputs the values into the fields. The username must be length 6-20, and the passwords must be greater than 6 in length and be the same characters. If the passwords match a sign is shown to the user in which they may select the submit account which checks if the username isn’t already taken and creates the account. If the username is taken a message is displayed to the user to change their username. If the user selects the sign in page it takes them back to the main login page.

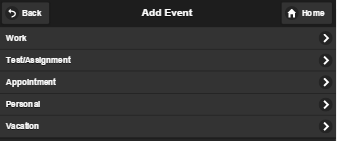
Home page:

Once you login the user is taken to the home page. The home page is where you browse to the utilities of the application. To add an event you click the add event button, edit event to edit, and to display the events in a list or calendar format the calendar button is selected. If the user is finished the sign out button is selected.



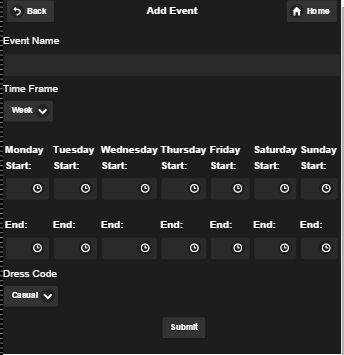
Add Event Page:

When the user selects the Add Event page the screen is switched to the Add event page. A list of options is shown in which they represent the type of events to input.



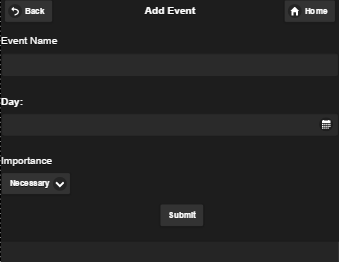
Work:

Below is the work page interface. For a general work event, the time frame is asked for, if it’s multiple days of the week or just a day. The event name is asked to store, and search for. The dress code is asked for to represent the buffer times between events.



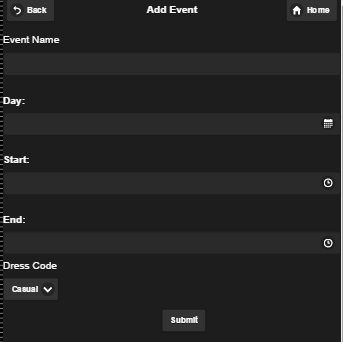
Test/Assignment:

If the user selects the Test/Assignment option this page is shown. The event name is asked for, how important this event is to the user, and the end date for this assignment.

****

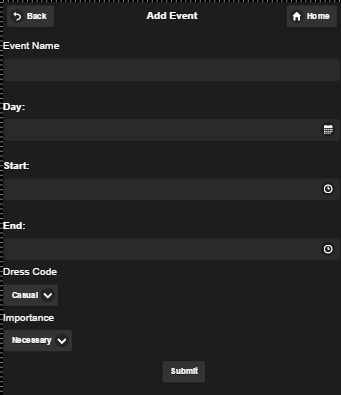
Appointment:

If the appointment option is selected the picture below is presented. The name, date of the appointment, start times, and end times are asked for. Depending on the type of appointment, such as hockey or an interview more time is needed to prepare and move to the next event. Casual or Formal will be selected in each event.

****

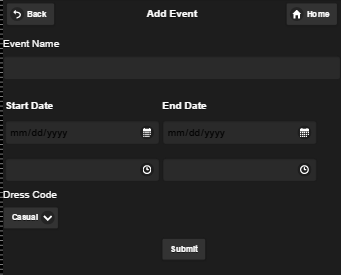
Personal:

The personal page is the same as an appointment except the importance of the event is not fixed. Depending on how important the event is it can either be overwritten or overwrite a different event. If you go for a coffee date it can be skipped for an assignment, but if you have a family birthday party it will take preference over the assignment.

****

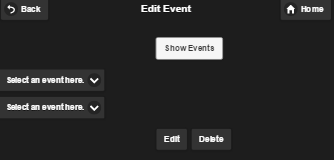
**Vacation:**

The vacation page asks for the event name, the start date and time of the vacation and end date and time for the vacation. If you are away on these days, the vacation takes priority over everything else and the days are taken up by vacation time.

****

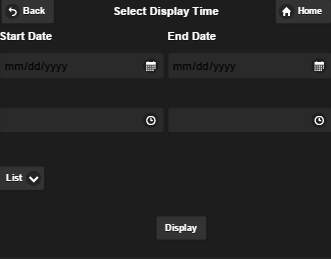
Edit Event:

If the edit event page is selected by the user the picture below is presented. To see the unique events the user has the show events button is selected. The events are shown in the first drop down box and once that is selected. The individual days of each event is shown. The user can either press edit to edit the event, or delete to delete it.

****

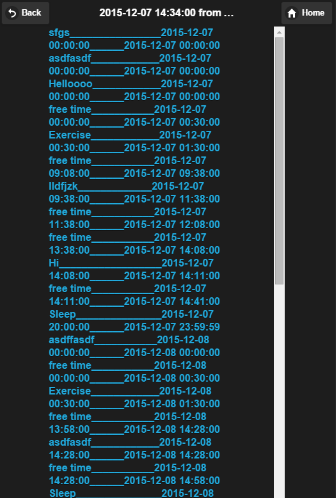
Calendar:

Below the calendar page is presented. It asks for the start date and time in which you want to view, then asks for the end date and time. Once finished the user selects either the list option or calendar option from the dropdown box and the user hits display to the inputted fields.

****

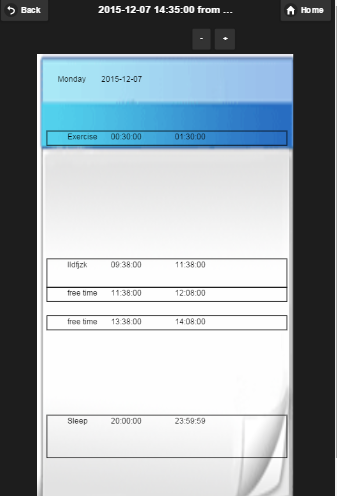
List View:

If the list option is selected, each event is listed off with its information in the schedule. Day by day each event is read off. The Event name, start and end times are shown.

****

Calendar View:

The calendar page is presented if the calendar option is clicked. It is a more visually appealing option to view your daily schedule, day by day. To see the next day the + button is clicked. To view the prior page the – button is selected.

****