Software Engineering
Software Requirements Specification (SRS) Document

<Awesome To-Do List>

<9/15/2022>

<Final Version>

<By: Henry Huerta, Yinru Li, Sara Gable>

<I HAVE ABIDED BY THE UNCG *Academic Integrity Policy* ON THIS ASSIGNMENT. (Henry Huerta, Yinru Li, Sara Gable)>

Table of Contents

1. Introduction

1.1 Purpose:

The goal of our project is to create an web application that is able to take users task information and store it in the form of a to-do list to make memorizing said tasks a thing of the past. To allow the user to become more productive throughout their day, this program will act as a helper to keep things off of their mind and on their screen. The weather information will be displayed on the homepage while they are also able to enter the tasks. There will also be a joke displayed below the list to start the users day off with a smile. Additionally, relaxing music will be (optionally) played in the background when users plan for their daily tasks.

1.2 Document conventions:

The purpose of this Software Requirements Document is to describe the client-view and developer-view requirements for the Awesome To-Do List web application.

On the client-side there will be a window displaying local weather information.Underneath that there is the date of the current day. And underneath that there is an input box allowing for user input of the tasks that they want to add to the to-do list. At the very bottom there will be a joke section where the user clicks on the “Get a Joke” button to display a new joke. The system will be made to serve those with busy lives who can’t be bothered to memorize menial tasks such as remembering to drink water and take out the dog. Busy parents, athletes with rigorous schedules, private business owners, and just overall people with busy schedules would be the ones to benefit from our web app providing a smooth and easy to use To-Do list experience.

On the developer-side the app has a database for user login information including their username, hashed password, and the account ID. There will be an api for the weather of the user’s location as well as for the jokes and music. There will be a function for changing the background image on the website matching the users preference. The performance of the website will be made to function smoothly and have high response times, as well as storing the information given in the most efficient way possible, allowing the data to be retrieved as fast as possible for the user.

1.3 Definitions, Acronyms, and Abbreviations

.Net	A set of software technologies from Microsoft for connection information, people, and computer systems.
ATDL	An abbreviation for Awesome To-Do List. This is the name of the system that is being built.
JS	An abbreviation for JavaScript. We will be using this language to build the ATDL.
DB	An abbreviation for Database.

1.4 Intended audience:

The introduction and general description in this SRS inform the users about what we are hoping to accomplish through this project, giving a general overview of the things they can do with our application. The technical, functional, and non-functional requirements are intended for our development team to have specific ideas about what exactly we need to build for the users.

1.5 Project Scope:

The primary goal of our application is to allow our clients to clear their minds by putting their tasks in our app; to be rid of memorizing a bunch of to-dos, so our users can focus on completing their tasks more efficiently. Also we look forward to helping our users start their day in an eased mood, so our application adds jokes and relaxing music as additional features. For additional helpfulness, the application also provides local weather information, so even planning for that will be handled. Generally speaking, we want to help the users prepare for a productive day.

1.6 Technology Challenges:

We will build the application primarily using HTML, CSS, and JavaScript, as well as more convenient and advanced tools like node.js, express, JQuery, etc, so it is important that every team member has basic knowledge about how those tools work.

At this point, the biggest challenge for us is to retrieve API data and update it to only a portion of the webpage.

1.7 References:

2. General Description

2.1 Product perspective:

These days, many people have all sorts of tasks to perform every day, and it becomes difficult to keep track of these to-do's in mind. So we thought a to-do list application will help people manage their day in a clear, reasonable, and efficient way.

2.2 Product features:

Primarily, the application lists what a user has to do in a day. Users can add tasks onto the to-do list; when a task is completed, users can cross it out. If a user no longer wants to see a task, completed or not, they are able to delete it easily.

The application has some very helpful side features. First, the application has a reserved space to show the current weather at your location, so you'll know if it's safe to go out or not. Also, the application displays a joke each day and plays soothing background music, so the users can have a bit of ease while planning for a busy day.

2.3 User class and characteristics:

Our application only expects our users to be able to open a webpage in a browser. Any person who is intelligent enough to realize the importance of making a plan before doing things will be able to use the application right away.

2.4 Operating environment:

The application is a web-application that runs on web browsers such as Chrome, Edge, Safari, etc. Any devices that have access to a web browser can use the application.

2.5 Constraints:

The user must input their own tasks because it is so individualized. Also, because it is online, the user must have access to the internet.

2.6 Assumptions and dependencies:

We assume that the user will know some bit about how computers and websites work. We also assume that users will have access to the internet and are able to stay connected throughout their days.

3. Functional Requirements

Functional requirements

3.1 Primary

(1) Li & Sara - The application will allow users to add tasks to a to-do list.
(2) Henry - The application will locally save the to-do list for user revisit.
(3) Li & Sara - The application will delete a task from the todo list if the users wish to delete it by clicking a “cross” button next to the task.
(4) Li - Users will be able to switch between up to three lists whilst saving their differing data each time

3.2 Secondary:

(5) Li - The application will display weather information. Weather information includes the city name, temperature, and a brief weather description.
(6) Li - The application will play relaxing music if the users click on the play button. The music is selected from a few music files built-in to the website.
(7) Li - The application will display a joke each day in a box at the bottom of the webpage.

2. Technical Requirements

Operating System & Compatibility
Interface requirements

2.1.1 User Interfaces

(1) The to-do list will be on the right of the webpage.
(2) An input box is at the top of the list, users enter a task here, and there is a “plus” button right next to the input for adding the task to the todo list.
(3) Beside the list will be the weather information.
(4) Below the weather will be a box where a piece of joke lies. The joke box will update when clicked by the user.
(5) A play button(s) will be placed at the bottom left corner of the webpage. Users can play music by clicking on the button(s).

2.1.2 Hardware Interfaces

The application is simply a HTTPS website, so any devices that have access to internet service and web browsers will be able to use the application.

2.1.3 Communications Interfaces

All that is required is a valid email for an account as well as reminders.

2.1.4 Software Interfaces

We will need to make our project compatible with the varied API's necessary for the ideal completion. We will also need the lists to be compatible with how tasks are received so that they may be processed in a simple and efficient way.

3. Non-Functional Requirements

Performance requirements

- (1) The list should be able to take as much information as the user needs to input within the limitations of guest or user account
- (2) The user should access the information by logging in to their account from the device they made the list on
- (3) The weather information should be displayed instantly to the user
- (4) The music should be played without lag and instantly when the button is clicked by the user
- (5) The joke should be displayed instantly as the user refreshes the page or click to receive a new one

Security requirements

A user must use the same device to fetch the to-do entries he inputted. If logged in on a different device, the todo entries made on the other device won't display.

Software quality attributes

5.4.1. Availability - Will be available to all online guests or users
5.4.2. Correctness - Tasks will be inserted by the user
5.4.3. Maintainability - Application will be clean and easy to update
5.4.4. Reusability - Code will be concise so parts may be reused if needed
5.4.5. Portability - Application will be available on all common devices

Process Requirements

5.5.1. Development Process Used - Teamwork
5.5.2. Time Constraints - Before deadlines given by our wonderful professor Sunny
5.5.3. Cost and Delivery Date - As free as possible

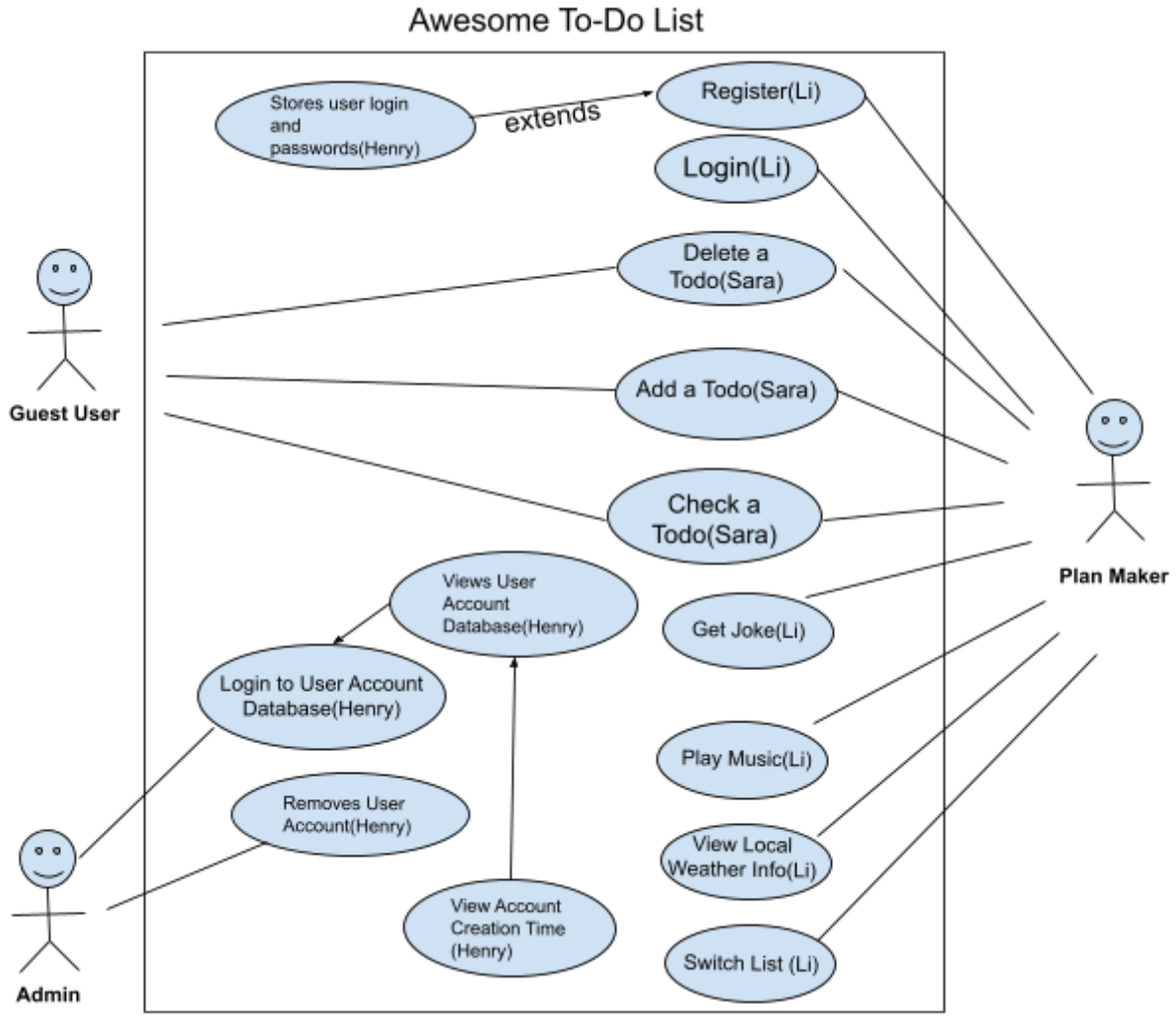
Other requirements

All music will be royalty-free, and that should be the only possible legal roadbump.

All SRS/SRD should be:

- **Correct:** A method of analysis that ensures that the software meets the requirements identified.
- **Unambiguous:** There is only one interpretation of what the software will be used for and it is communicated in a common language.
- **Complete:** There is a representation for all requirements for functionality, performance, design constraints, attributes, or external interfaces.
- **Consistent:** Must be in agreement with other documentation, including a systems requirements specification and other documents.
- **Ranked for Importance and/or Stability:** Since all requirements are not of equal weight, you should employ a method to appropriately rank requirements.
- **Verifiable:** Use measurable elements and defined terminology to avoid ambiguity.
- **Modifiable:** A well-defined organizational structure of the SRS document that avoids redundancies can allow easy adaptation.
- **Traceable:** Ability to trace back to the origin of development and move forward to the documents produced from the SRS.
- **Legible and Professionally Presented**: Must use a consistent font and style. Must have proper formatting of tables and charts. Must be grammatically correct. Use active tense and concise sentences.

Use-Case Model



Brief Description:

- Add a Todo
  The user should be able to type a task in the input box and add it to the todo list.

- Remove a Todo
  The user should be able to delete a task on the todo list.

- Check a Todo
  The user should be able to mark a todo done.

- Switch List
  The user should be able to switch to a different list by clicking on the one of dots next to the list. There should be 3 different lists.

- View Local Weather Information
  The user should be able to view local weather information on the homepage.

- Play Music
  The user should be able to play relaxing music by clicking on the “play” icon.

- Get a Joke
  The user should be able to display a piece of joke every day by clicking on the “get a joke” button.

- Remove User Account
  The admin should be able to remove a user account from the database.

- View Account Creation Time
  The admin should be able to look up the creation time of an account.

- View User Account Database
  The admin should be able to view the user account database that stores usernames, passwords, and account IDs.

- Store User Login Information
  The user account database should be able to store usernames, passwords, and account IDs when a new account is created.

- Login
  The user should be able to log into the application from the login page.

- Register
  The new user should be able to register for a new user account on the registration page.

Scenarios:

A. Add A Todo

I. Initial Assumption: User is logged in and can access the to-do list application, or the user chooses to use the app as a guest user.
II. Normal: The user inputs a task and presses “enter” to add it onto the list.
III. What Can Go Wrong: The user types something wrong and adds the wrong task to the list; if so, the user can simply delete the wrong todo entry and try again.
IV. Other Activities: N/A
V. System State On Completion: The todo list displays the new todo entry.

B. Remove A Todo

I. Initial Assumption: User is logged in, he can access the to-do list application and has added todo entries on the list, or the user chooses to use the app as a guest user.
II. Normal: The user deletes an todo entry by clicking on the “trashcan” icon next to it.

III. What Can Go Wrong: User removes the wrong item; if so, the user can type it and add it again.
IV. Other Activities: N/A
V. System State On Completion: The todo entry disappears on the todo list.

C. Check a Todo

I. Initial Assumption: User is logged in and has already added things to the to-do list, or the user chooses to use the app as a guest user.
II. Normal: The user clicks the checkmark to mark a todo entry that has been done.
III. What Can Go Wrong: Users may mark a task as done even though it has not been done; if so, the user can unmark it by clicking on the checkmark again.
IV. Other Activities: N/A
V. System State On Completion: The to-do list crosses off the todo entry with a line through, and the entry turns green indicating the todo is completed.

D. Get Joke

I. Initial Assumption: User is logged in and can access the to-do list application.
II. Normal: By pressing a button, the user generates a joke.
III. What Can Go Wrong: The joke API fails to respond; if so the application will display a local default joke, “Diarrhea is genetic. It runs in your jeans.”
IV. Other Activities: N/A
V. System State On Completion: the webpage displays a joke in a box below the todo list.

E: Play Music

I. Initial Assumption: User is logged in and can access the to-do list application.
II. Normal: The user clicks on the “phonograph” icon, and the application plays relaxing music on the homepage.
III. What Can Go Wrong: the application fails to read the music files. The users should be able to contact the developers for a fix.
IV. Other Activities: May turn music on or off at will: click on the phonograph to turn it on, click on it again to turn it off.
V. System State On Completion: Relaxing music plays on the homepage as the user creates his todo list for the day.

F: View Local Weather Info

I. Initial Assumption: User is logged in and allows the web-app to know his location.
II. Normal: The user is able to see, in a box at the top left corner of the homepage, the weather information in the location in where the user is and the current date.
III. What Can Go Wrong: The application fails to provide the weather information of the user’s exact location, however, it is usually about the areas not far from the user. Or the user refuses to give the web-app his location, then no weather information for him.
IV. Other Activities: N/A
V. System State On Completion: In a box at the top left corner of the homepage, the application displays the city name in where the user is, the weather icon that describes the weather, the temperature(°F) in the city, and the current date.

G: Login to User Account Database Page

I. Initial Assumption: The admin remembers the admin code.
   Normal: The admin clicks on the “lock” icon to access the admin login page, then he will enter the admin code secret to the admin alone, then the admin will access the user account database UI connected to the MongoDB database which stores all passwords and usernames.
   What Can Go Wrong: The admin forgot the admin code, or the admin code leaks to other individuals.
   Other Activities: N/A
   System State On Completion: The admin can see all usernames and passwords.

I: Removes User Account

I. Initial Assumption: Admin is logged in on the application administration page.
II. Normal: Admin can input the account ID to delete the intended user's account.
III. What Can Go Wrong: the admin could delete the wrong account.
IV. Other Activities: N/A
V. System State On Completion: The user account database no longer holds the intended users account.

J: Views User Account Database

I. Initial Assumption: Admin wishes to view the information stored in the User Account Database (UAB). And the admin has logged into the application administration page.
II. Normal: Admin views the UI connecting to the MongoDB database of all user account information.
III. What Can Go Wrong: N/A
IV. Other Activities: the admin is allowed to view the user account information and delete a user account because he’s the authority.
V. System State On Completion: the application displays the MongoDB user account database to the administrator.

K: Stores User Login Information

I. Initial Assumption: The user account database is working properly.
II. Normal: the user account database stores the username, the password, the account ID when a user account is created.
III. What Can Go Wrong: The MongoDB database fails to store user account information. The user should be able to contact the developer for a fix.
IV. Other Activities: N/A
V. System State On Completion: the user account information(username, user email, password) that is created from the application’s sign up page is stored in the user account database as a line of MongoDB data.

L: Login

I. Initial Assumption: The user has a registered account to login to the application. The user account is saved in the database.
II. Normal: The user will enter username and password to log into his account.
III. What Can Go Wrong: The user enters username and password that don't match. The user should be able to contact the developer for password reset.
IV. Other Activities: N/A
V. System State On Completion: The user is logged in, and he can use the application.

M: Register

I. Initial Assumption: The user accesses the login page, and he wants to register for a user account.
II. Normal: The user gets to a registration page where he enters a username, password, and his email address. Then an account is created.
III. What Can Go Wrong: The user enters an email address that doesn't belong to him. The application should send verification code to the email address, and the user needs to input the correct code to complete the registration. Also, the username might have been used, the application should check for and avoid duplicates.
IV. Other Activities: N/A
V. System State On Completion: the application transfers the user to the registration page, after the user finishes the registration process, the application sends the user account information to the user account database for storage.

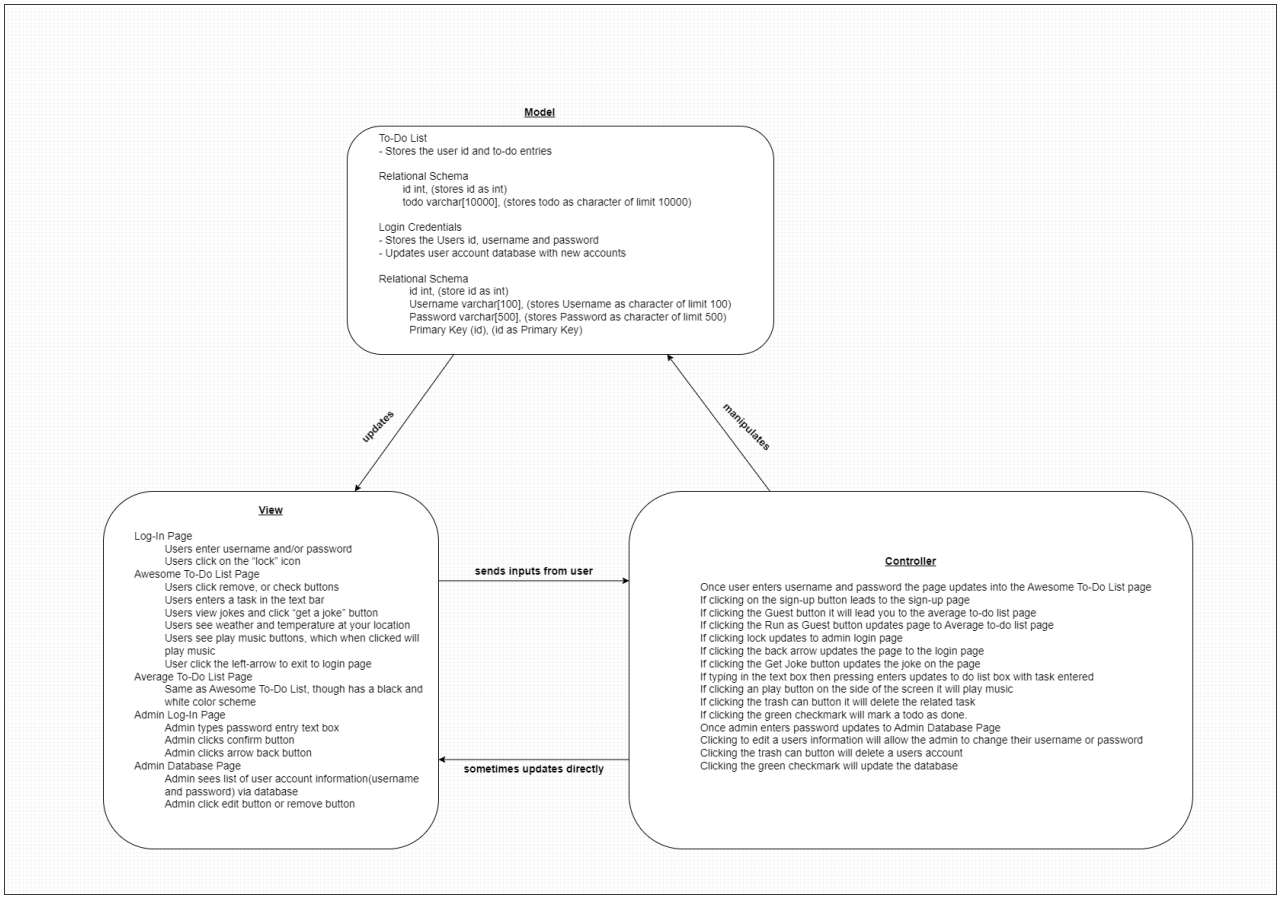
N: Switch Lists

VI. Initial Assumption: The user is logged in.
VII. Normal: The user clicks on one of the dots next to the list to switch to a different list.
VIII. What Can Go Wrong: N/A
IX. Other Activities: N/A
X. System State On Completion: the application switches to a different independent list.

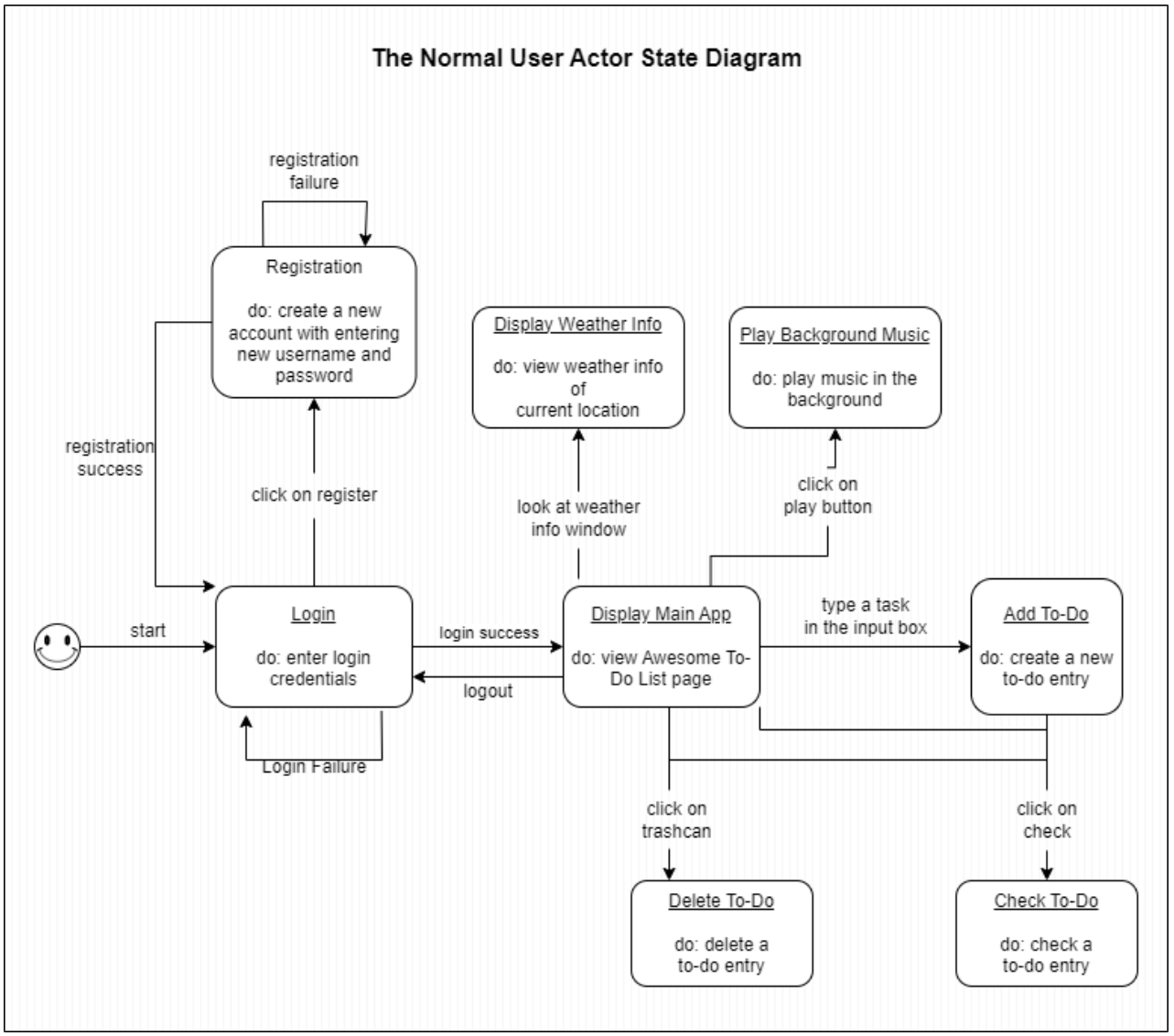
O: Check Creation Time

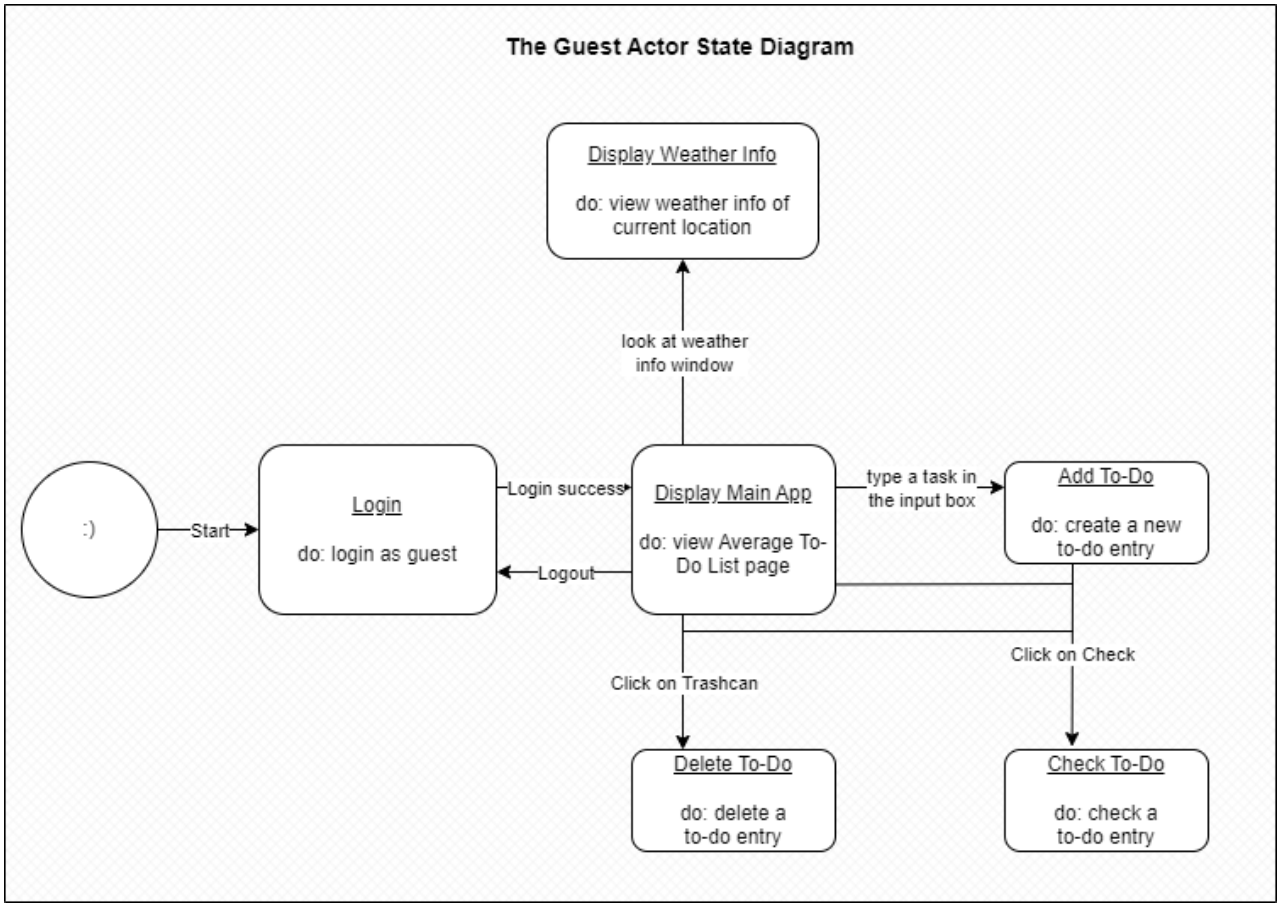
XI. Initial Assumption: The Admin is logged in.
XII. Normal: The admin enters an account ID to the yellow input box, enter clicks on the "clock" icon to read the creation time of the account in a pop-up window.
XIII. What Can Go Wrong: N/A
XIV. Other Activities: N/A
XV. System State On Completion: a pop-up window displays the creation time of the account.

Software Architecture

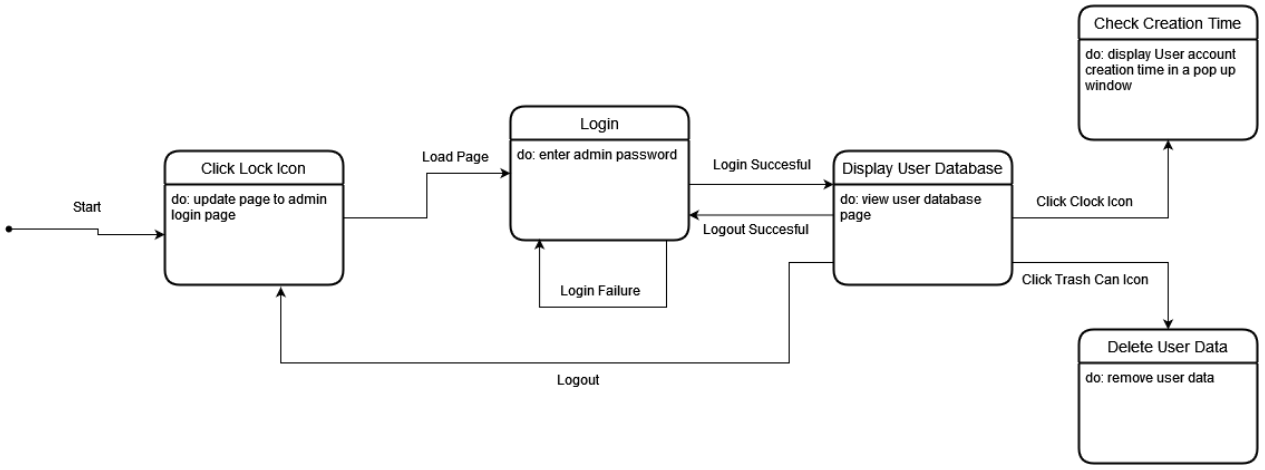


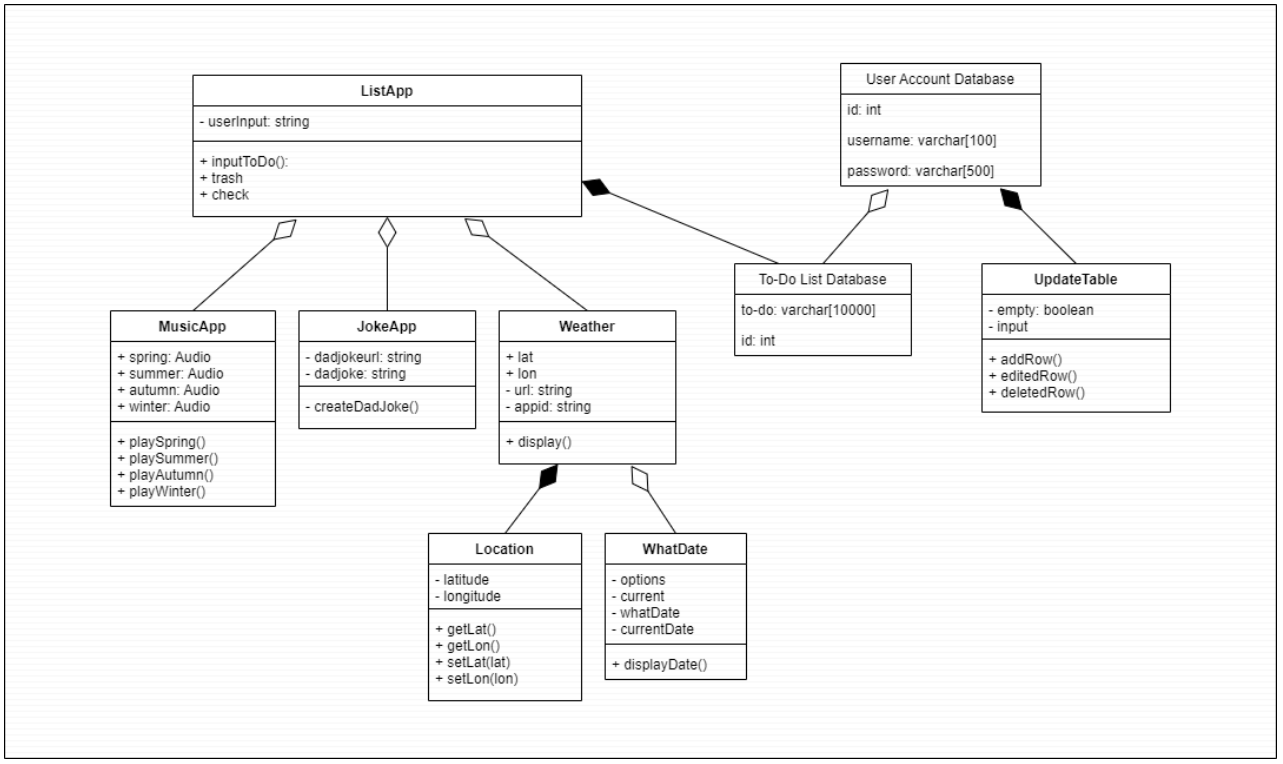
Software Design





Admin State Machine Diagram

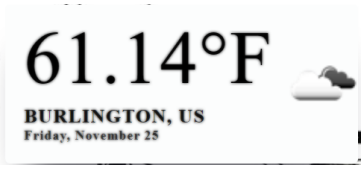



ListApp
- userInput: string
+ inputToDo():
+ trash
+ check
User Account Database
id: int
username: varchar[100]
password: varchar[500]
MusicApp
+ spring: Audio
+ summer: Audio
+ autumn: Audio
+ winter: Audio
+ playSpring()
+ playSummer()
+ playAutumn()
+ playWinter()
JokeApp
- dadjokeurl: string
- dadjoke: string
- createDadJoke()
Weather
+ lat
+ lon
- url: string
- appid: string
+ display()
To-Do List Database
to-do: varchar[10000]
id: int
UpdateTable
- empty: boolean
- input
+ addRow()
+ editedRow()
+ deletedRow()
Location
- latitude
- longitude
+ getLat()
+ getLon()
+ setLat(lat)
+ setLon(lon)
WhatDate
- options
- current
- whatDate
- currentDate
+ displayDate()

Final Presentation Scenarios

<-----Guest----->
Guest: Login use case:
Guest user clicks on "Run as Guest" on the login page.

Guest: Display Weather Info use case:
Guest user views the local weather info(Current date, weather icon, and temperature).



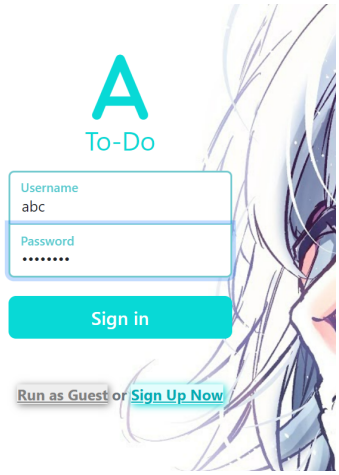
Guest: Input Todos use case:
Guest user types a todo in the input box and hits "enter" to add it to the list.

Guest: Delete Todos use case:
Guest user clicks on the "trashcan" next to a todo entry to delete it from the list.

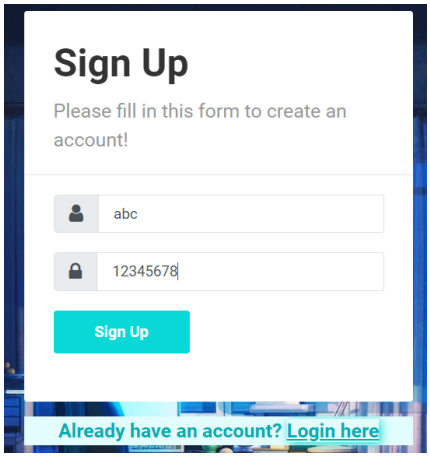
Guest: Check Todos use case:
Guest user clicks on the "check mark" next to a todo entry to cross it out on the list.

Guest: Logout use case:
Guest user clicks on the "left arrow" to return to the login page.

<-----User----->
User: Login use case:
User enters login credentials on the login page,
then hits "enter" or clicks on "Sign in" to access the main app.

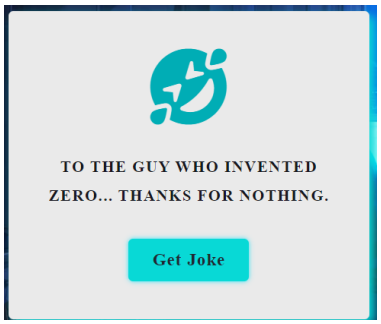


User: Register use case:
User clicks on “Sign Up Now” on the Sign In page to access the sign up page,
from there the user enters new username and new password to create a new account.



User: Display Weather Info use case:
User views the local weather info(Current date, weather icon, and temperature).

User: Get a Dad Joke use case:
User clicks on "Get a Joke" to display a dad joke.

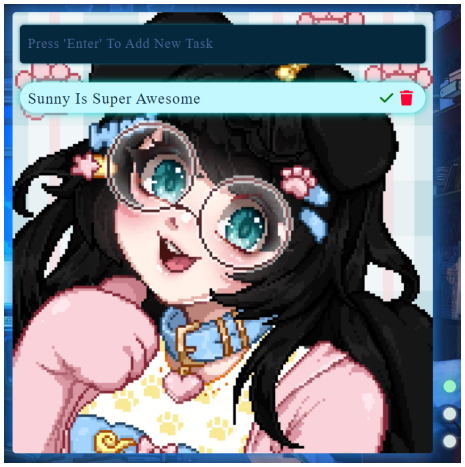


user: Play Background Music use case:
User clicks on a play button to play background music.
If the music is playing, clicking on the button will pause the music.

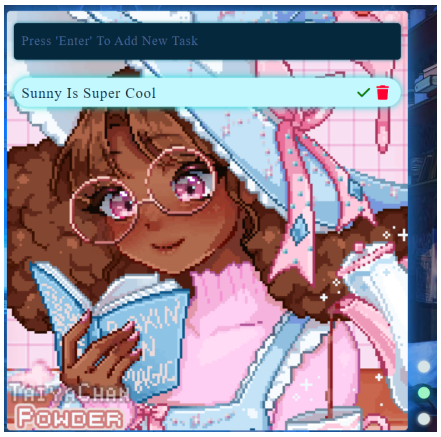
User: Input Todos use case:
User types a todo in the input box and hits "enter" to add it to the list.

User: Delete Todos use case:
User clicks on the "trashcan" next to a todo entry to delete it from the list.

User: Check Todos use case:
User clicks on the "check mark" next to a todo entry to cross it out on the list.



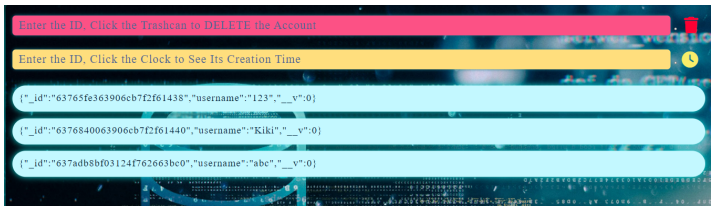
User: Switch List use case:
User clicks on the “circles” to switch between lists.



User: Logout use case:
User clicks on the "left arrow" to return to the login page.

<-----Admin----->
Admin: Login use case:
Admin clicks on the "lock" icon on the login page to enter the admin page,
then inputs the admin code to access the user account info database.

Admin: View User Account Info use case:
Admin views user account infomation.



Admin: Check Creation Time use case:
Admin types a specific user's ID in the yellow input box,
and click on the "clock" button to display the account's creation time in a popup window.

Admin: Delete a User Account use case:
Admin types a specific user's ID in the red input box,
and click on the "trashcan" button to delete the user account from the database.

Admin: Logout use case:
Admin clicks on the "left arrow" to return to the login page.