Authors: Samira Khan, Michele Pham, Ammar Elzeftawy, Jana Afifi, Carter Drewes, Amna Hassan and Muhammad Nasir

Contributors: Ali Dehaghi

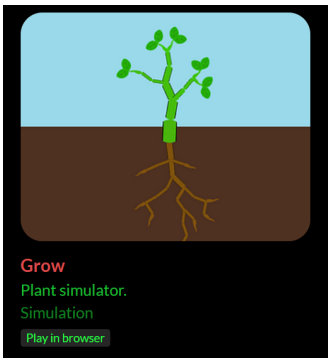
Last updated: 02/07/2023

Status: Complete

Purpose

Over the years, there has been an increase in awareness of mental health issues within our community. Studies conducted by Mayo Clinic Health System have stated that “1 in 3 college students experience significant depression and anxiety.”[1] A popular way to improve many aspects of mental health is gardening. It can significantly reduce stress, anxiety, sadness and more. However, many students do not have access to gardens. Digital Flora aims to imitate the steps of growing a real plant in a way that is accessible to everyone. It is a calming and simple web based game that individuals can play to take time off from their stressful days. Furthermore, it provides users a space to share their thoughts and express themselves with the option to interact with individuals who share common interests including gardening.

Background Reading



Grow by Japes

Pros:

- Easy to play, No complex instructions: Straightforward to grow one plant.
- Sunlight and water are used as currency.

Cons:

- Limited options for users eg : Add ons such as flowers.
- Made with Unity : Performance issues on lower-end hardware.
- Made with Unity : Limited flexibility or control developers may have on their game as Unity has its own scripting language.
- Does not reflect real -life plant growth due to the complexity of the plant growth process.
- According to users : Certain fields have errors when refreshed.
- No real purpose of the game.

Plant Daddy

Pros:

- Realistic game : Many features depict real life plant growth process.
- Users can customise pot colour, pot rotation and size.
- Plant keeps growing : Users can change/close tabs while the plant grows in the background.
- The plant produces leaves that are collected as currency to buy new plants.
- Variety of plants.
- Users can take a picture of their plant : Gets saved on device.

Cons:

- As a browser game this runs on a unity window and is not optimal - performance wise.
- Lags as soon as the apartment is filled with plants.
- Can get boring after a certain amount of time.

Context

Digital Flora is a web based game where individuals who are passionate about gardening can come together and grow plants together. It is also a safe space for users to speak out about their mental health and interact with similar individuals. This virtual garden incorporates elements of mindfulness and self-care to help users improve their mental well-being. By taking care of virtual plants, users can be distracted from stress and negative thoughts. Also, the ability to water, take care and track the growth of their plants can provide a sense of accomplishment and satisfaction for the users, helping to boost their self-esteem. This feature is useful for users who are interested in learning about the needs of different plants and how to take care of them in a hands-on way. The virtual garden can also motivate users, encouraging them to return to the game regularly to check on their plants and provide the necessary care. Additionally, there is a trading feature between the users to promote a sense of engagement within the community. Users can trade their plants after harvesting them and trading it with interested users for coins. There will be a discussion board in the application where users posts their thoughts, reply to users, propose trades, offer trades accept reject trades.

The application is similar to games that users may have played before. However, it is an improved version. The improved version includes more realistic plant care and growth

simulation, a database to keep track of plants and user progress, and a system of credits to reward users for their plant care efforts. The database will help maintain the state of the virtual garden. This ensures the long-term viability of the game. Also, this allows users to keep track of their plants over time and see the progress they have made in caring for their virtual garden. Users will be using coins which they will be using to purchase plants. This can add an element of gamification, making the experience more enjoyable and engaging for users.

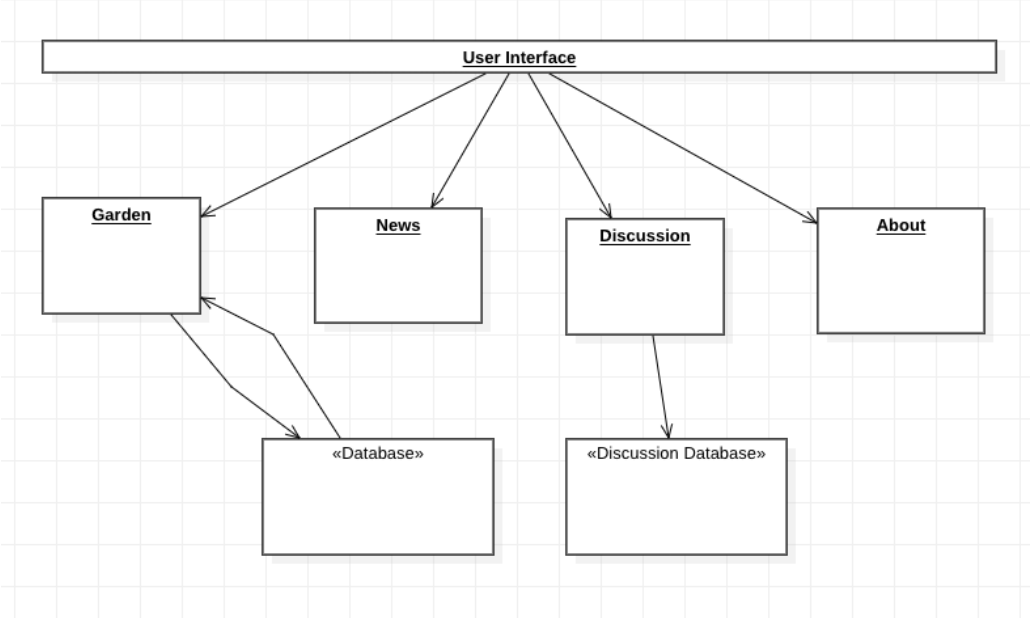
Upon logging in, users can access the news section to stay informed about newly available plants. Each user has a personal garden where they can purchase plants using earned coins. Each plant purchase is randomly assigned. Users can care for up to three plants by watering them every 30 minutes. Neglecting a plant for two days will result in its death, allowing for planting a new one. Users can earn coins not only by selling their plants, but also by watering them.

We chose to employ the Model-View-Controller (MVC) pattern for our application due to its superiority over event sourcing. MVC offers a clear separation between the model, responsible for database operations, and the view and controller, which handle presentation and user interaction, making it ideal for database integration. Furthermore, MVC is a straightforward pattern that is easier for developers to grasp and implement, leading to faster development and improved user experience. MVC's component-based structure also facilitates debugging by simplifying issue isolation. The separation of components in MVC leads to improved performance by reducing complexity.

Detailed Design

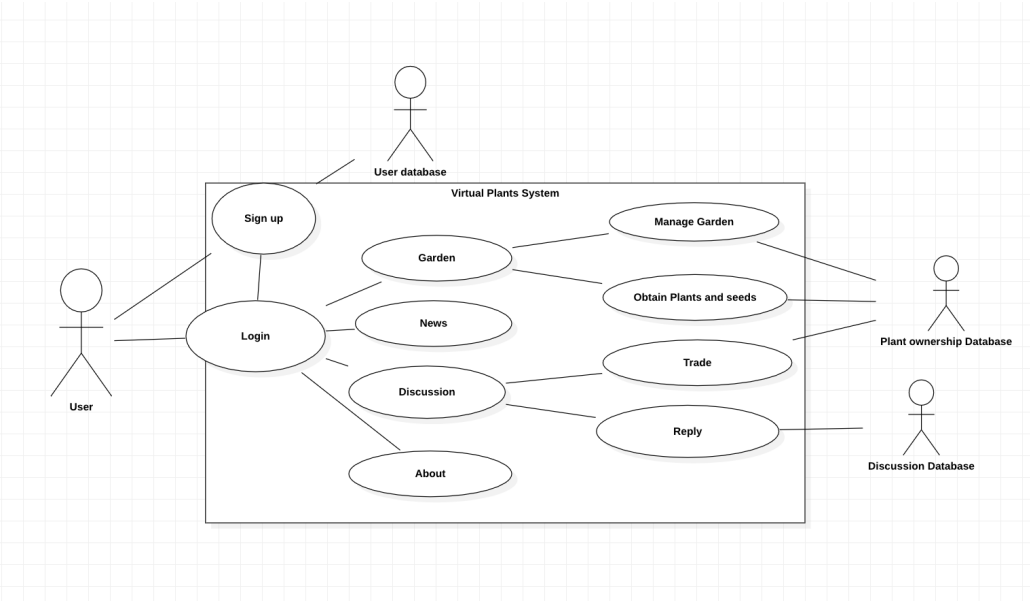
The system will be structured with MVC and service-based architecture. We will have a model that keeps track of users, plants (which are assigned to users), credits, plant growth, etc., as well as functionality to trade plants among users for credits. The user interface will allow users to log in to your account, and from there users will be able to interact with their plants, water them, trade them, trade for other plants with users, purchase new plants, and discuss with other users. The data will be structured with Firebase, and users will have a unique ID, identifying them. They will own plants, which also have their own unique ID’s and data, such as growth level and soil moisture. It will be possible to move plants amongst users, and this will be done by appending the unique ID of the new plant to their list of owned plants. Javascript and ReactJS will control the front end functionality of the website. Menus and buttons will be handled mostly from within the user interface, but more complicated interactions such as trading with other users, or manipulating data in the model, such as watering the plant, will be handled within an abstracted controller.

Overall system architecture (MVC and service-based architecture)

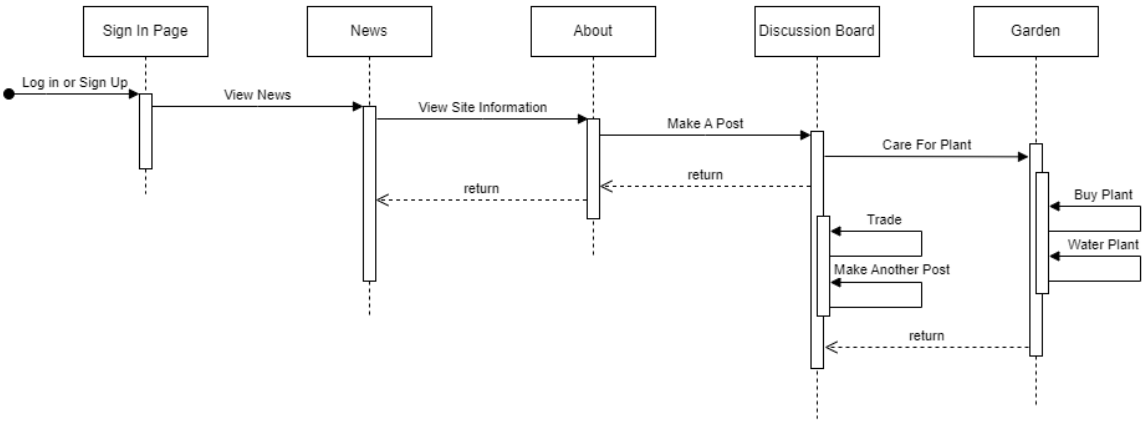


The “User Interface” component is equivalent to the View component of the MVC architecture. The middle row in the diagram consists of “Garden”, “News/Notifications”, “Discussion”, and “About”. These would act like the Controller but split into four different services to handle different logic. The two databases on the diagram will represent the Model component in MVC. However, both architectures suggest having one database. The group thinks that two databases would be easier to manage different information and may not need to be synced.

How users interact with the system

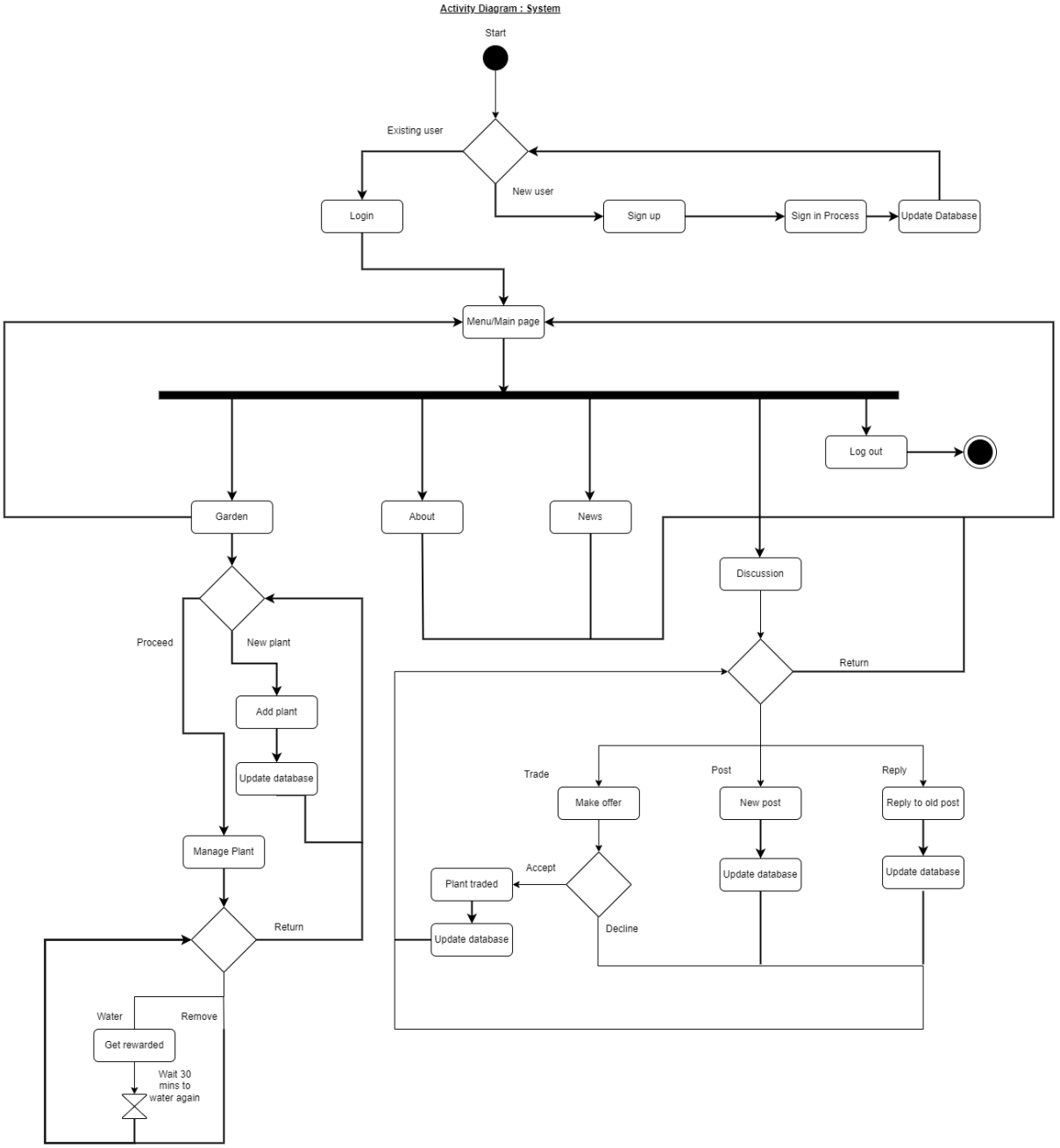


How software components interact



- Component for User Management: This component is in charge of overseeing user accounts, authentication, and authorization. Information like user profiles, login credentials, and user roles would be stored there.
- Component for Discussion Management: This part of the application is in charge of managing posts and comments. Information including post names, content, and comments would be stored. Additionally, it would manage tasks like adding new posts, locating posts, and changing post data.
- View/User Interface component: Displays information to the user. The View component in Digital Flora would show the list of posts, each post's content, and any other data.
- Controller/Services component: Between the Model and the View, there is a component called the Controller. When a user provides input, such as starting a new post or leaving a comment, the Model is updated appropriately. Additionally, the Controller would get information from the Model and give it to the View so it could be shown.

How data flows through the system



- If the user wants to view the contents of a discussion post posted on Digital Flora, the following steps depict how the data flows through the system:
    1. To get the data for the post, the View component makes a request to the Controller component.
    2. The Model is accessed by the Controller to obtain the data for the post. This data is given to the view.
    3. The View shows the user of Digital Flora the content of the post after receiving the data from the Controller.

4. The View delivers the user's input to the Controller whenever the user engages with the post such as liking it or adding a comment.
5. After adding the new data to the Model, the Controller obtains the modified data from the Model.
6. The View updates the user with the new information after receiving it from the Controller.

Implementation Plan

- Technologies you intend to use:

  The front end of the system will be developed in ReactJS since it is commonly used in industry, and is an efficient framework for developing user interfaces. The functionality of the system will be handled by JavaScript, and the database will be built in firebase. When developing and deploying the system NodeJS will be used to run the application. For bug tracking, Azure DevOps or Jira will be used.

- How you will manage source code:

  To manage source code between team members, a GitHub repository will be shared with team members. Members are expected to fork main and make their changes in the fork. When updates to main occur, the fork should be rebased to the newest commit in main to avoid merge conflicts when issuing a pull request. When the feature being worked on in the fork is completed, a pull request will be issued, and the changes will be added to main.

- How you will integrate components:

  ReactJS will work natively with NodeJS and Firebase. Firebase will be accessed via API requests.

- How you intend to test the system:

  A variety of testing methods will be used such as unit testing and integration testing. As previously mentioned, any bugs will be reported using Jira or Azure DevOps (will be discussed later on with the team).

- How will you deploy the system:

  The system will be deployed using firebase hosting.

Project Planning

- Team organisation

Team Member	Responsibilities
Ammar	User interface
Samira	User interface
Carter	Managing the project, integrating front and backend systems
Michele	User interface
Amna	Backend
Muhammad	Backend
Jana	Integrating front and backend systems, bug fixing

1-2 meetings per week, one in the lab and depending on the workload if we need another one we'll meet on Zoom. Tasks will be split equally between team members, and some tasks can be done individually while others should be done in pairs.

- Schedule

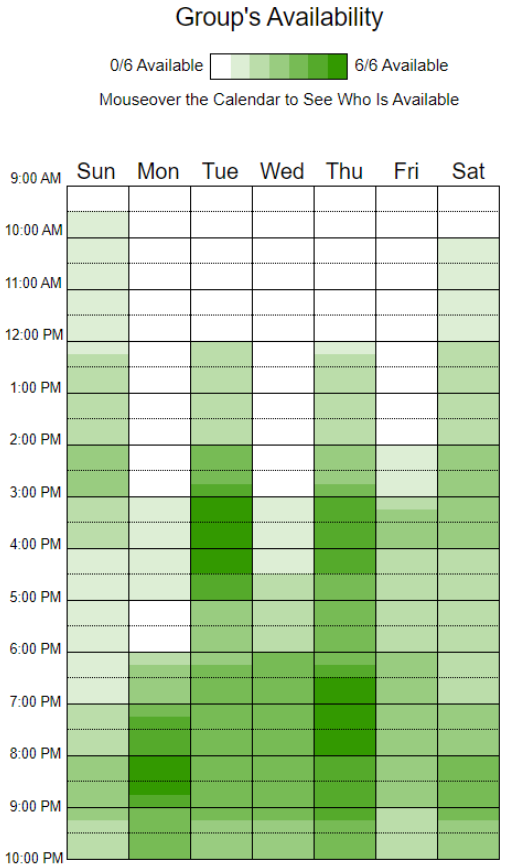
Since the project final report and deliverables are due on March 29, 2023 at 11:59 pm, the entire project must be completed within eight weeks. The first two weeks were used for designing the project, figuring out logistics, and establishing the outline and timeline for the project.

The next four weeks will be for implementation, where the coding will begin. During the first two weeks of this period, the large group will separate into two teams where one team will work on the database and the other team will work on the user interface. Throughout these two weeks, both teams will be communicating with each other to notify everyone about their progress. The final two weeks during the implementation period will be for integrating the two components created in the last two weeks and developing backend logic. Again, for efficiency, the group will also split into two teams.

Two weeks before the deadline will be reserved for finalizing the project. Tasks include ensuring that all components fit together, debugging, testing, completing the final report and updating any documents to preserve project accuracy.

Development milestones heavily rely on TA check-ins. As of now, there are no set deliverables that are expected to be presented during these check-ins. Therefore, the team has decided to have development milestones at the end of the scheduled times set above. For example, by the end of Week 2 of the project, all designs, software architecture, and diagrams should be agreed upon in preparation for the implementation period. The next two weeks will produce a complete database and a somewhat functional frontend. By the end of the following two weeks, all components of the project should be completed and be prepared for integration. The final two weeks are for integration and testing. By the end of the first week during this finalising period, the complete software should be ready to be used. The very last week of the entire project will result in a complete final report, updated documents, and all complete deliverables worked on throughout the project. As with every project, this is only a tentative plan as scheduling conflicts may happen which could result in delay or early progression of the project, but the group plans on closely following the schedule.

Appendix



References

[1] L. I. C. S. W. Reese Druckenmiller, “College students and Depression,” *Mayo Clinic Health System*, 07-Dec-2022. [Online]. Available: https://www.mayoclinichealthsystem.org/hometown-health/speaking-of-health/college-students-and-depression. [Accessed: 05-Feb-2023].

[2] Plant Daddy by Soglin - [Online]. Available: https://overfull.itch.io/plantdaddy. [Accessed: 04-Feb-2023]

[3] Grow by Japes - [Online]. Available: https://japes.itch.io/grow. [Accessed: 04-Feb-2023]