SEENG 401 - Final Project Report

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

Contributors: Ali Dehaghi

Last updated: 12/04/2023

Status: Completed

Project Link: https://github.com/MrClean-1/seng401-finalproject

Functional Requirements:
To elcit functional requirements, the group used strategies and methods used in requirements engineering. This included doing market research, brainstorming, interface analysis and other techniques. Then, the group decided on the priority of each functional requirement and listed them in the list below:

- *User registration and login:* Users can register for an account and log in to access the game.

- *Database integration:* The game employs the Model-View-Controller (MVC) pattern for database integration and maintenance.

- *Personal garden:* Each user has a personal garden where they can purchase plants using earned coins.

- *Plant care:* Users can water their plants every 30 minutes to keep them healthy. Neglecting a plant for two days will result in its death, allowing for planting a new one.

- *Plant trading:* Users can trade their plants with interested users for coins.

- *News section:* Users can access a news section to stay informed about newly available plants.

- *Discussion board:* Users can post their thoughts, reply to users, propose trades, offer trades, accept or reject trades on a discussion board.

- *Progress tracking:* The game maintains a database to track plants and user progress, allowing users to keep track of their plants over time and see the progress they have made in caring for their virtual garden.

- *Coin system:* Users will use coins to purchase plants and earn coins by selling and watering their plants.

- *Realistic plant care and growth simulation:* The game incorporates realistic plant care and growth simulation.

- *User rewards:* The game includes a system of credits to reward users for their plant care efforts, adding an element of gamification and making the experience more enjoyable and engaging.

Non-Functional Requirements:
The group took a similar approach for non-functional requirements elicitation. For these, we viewed similar projects and other web applications and brainstormed a few non-functinal requirements that we observed. As a group, we have created a list of these requirements that we think should be deployed in our project:

- *Performance:* The application should respond quickly and smoothly to user interactions, and it should be able to handle a large number of users simultaneously without any noticeable decrease in performance.

- *Reliability:* The application should be highly reliable, meaning that it should be available and functioning correctly for a high percentage of time. It should also be able to recover quickly from errors or system failures.

- *Security:* The application should be designed with security in mind, and it should protect user data from unauthorized access or malicious attacks.

- *Usability:* The application should be easy to use and navigate, with clear instructions and visual cues for users to follow. It should also be accessible to users with disabilities.

- *Compatibility:* The application should be compatible with a wide range of devices and browsers, and it should be able to adapt to different screen sizes and resolutions.

- *Scalability:* The application should be able to handle increasing levels of user traffic without a significant increase in the amount of resources needed to operate it.

- *Maintainability:* The application should be designed with maintainability in mind, meaning that it should be easy to update and modify as needed without introducing new bugs or issues.

- *Testability:* The application should be easy to test, with clear test cases and debugging tools available to developers.

Requirements Traceability Matrix:

The following table organizes the requirements mentioned in the previous sections and includes its priority, if the requirement has been met, and any additional comments. Since this was done early, it helped the group keep track of what needed to be done and the progress made. Some functionalities were not met due to the short time frame.

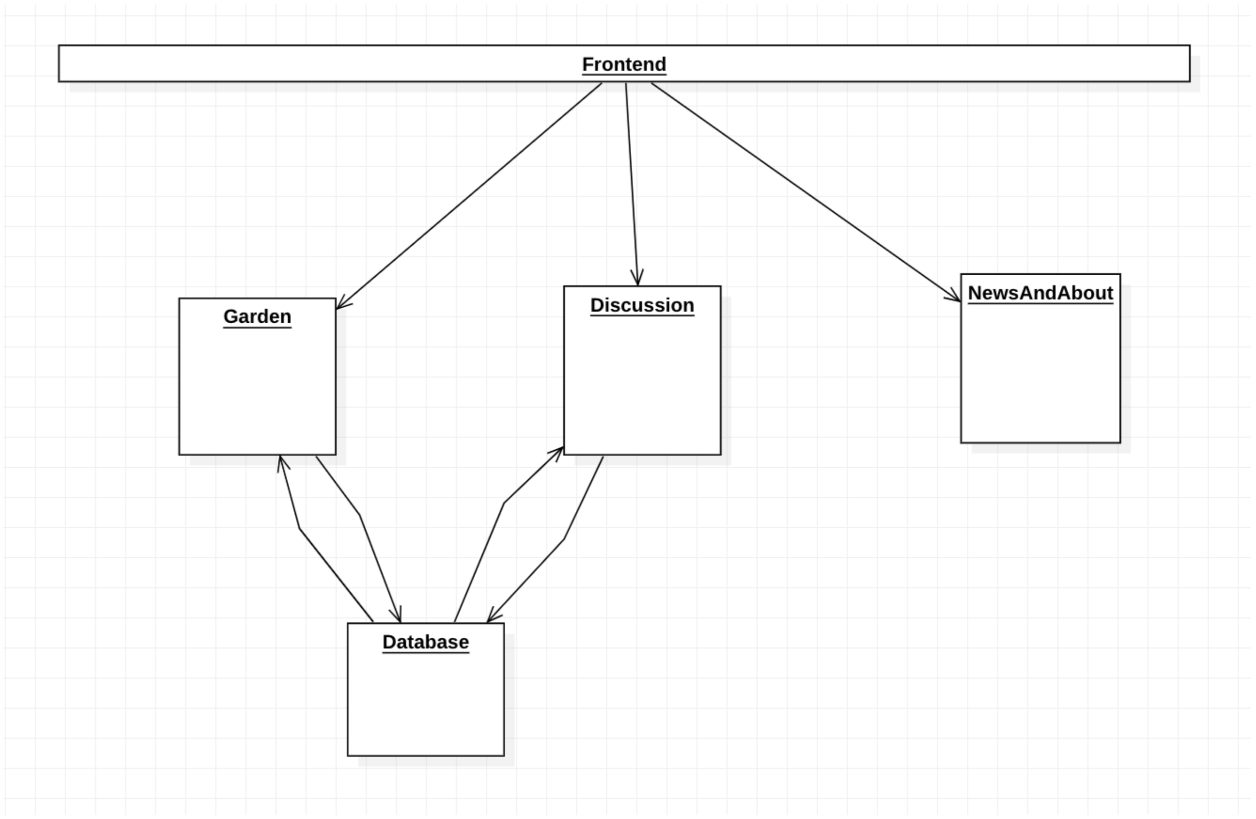
Req-No	**Description**	**Rational**	**Functional Specification**	**Priority**	**Req-Met**	**Comment**
FR1	User registration and login	Users need to be able to create an account and log in to access the game	Users can register for an account and log in with their credentials	High	Yes	
FR2	Personal garden	Each user needs a personal garden where they can purchase and care for plants	Each user has a personal garden where they can purchase plants using earned coins	High	Yes	
FR3	Plant care	Users need to be able to care for their plants to keep them healthy and alive	Users can water their plants every 30 minutes to keep them healthy. Neglecting a plant for two days will result in its death, allowing for planting a new one	High	Yes	
FR4	Plant trading	Users need to be able to trade their plants with others to earn coins	Users can trade their plants with interested users for coins	Medium	No	Was not able to implement within time period.
FR5	News section	Users need to be informed about newly available plants	Users can access the news section to stay informed about newly available plants	Low	Yes	
FR6	Discussion board	Users need to be able to communicate with other users about their plants and mental health	Users can post their thoughts, reply to users, propose trades, offer trades, accept or reject trades on a discussion board	Low	Yes	Posting a new post and replying to a post works, but

						any trading functionality does not work.
FR7	Coin system	Users need a way to earn and spend coins in the game	Users will use coins to purchase plants and earn coins by selling and watering their plants	High	Yes	Coins can be earned if plant(s) are watered, but plants cannot be sold for more gold.
FR8	Realistic plant care and growth simulation	Users need a realistic and accurate representation of plant care and growth	The game incorporates realistic plant care and growth simulation	High	Yes	
FR9	Database integration	The game needs to be able to maintain the state of the virtual garden and user progress	The game employs the Model-View-Controller (MVC) pattern for database integration and maintenance	High	Yes	
FR10	Progress tracking	Users need to be able to track their progress in the game	The game maintains a database to track plants and user progress, allowing users to keep track of their plants over time and see the progress they have made in caring for their virtual garden	High	Yes	
FR11	User rewards	Users need to be rewarded for their efforts in caring for their plants	The game includes a system of credits to reward users for their plant care efforts	Medium	Yes	

NFR1	Performance	The application should respond quickly and smoothly to user interactions	The application should be optimized for performance and able to handle a large number of users simultaneously	High	Yes	
NFR2	Reliability	The application should be highly reliable and available for a high percentage of time	The application should be designed with reliability in mind and able to recover quickly from errors or system failures	High	Yes	
NFR3	Security	The application should be secure and protect user data from unauthorized access or malicious attacks	The application should be designed with security in mind and employ best practices for data security	High	Yes	
NFR4	Usability	The application should be easy to use and navigate	The application should have an intuitive user interface, clear navigation, and easy-to-understand instructions and feedback messages	Medium	Yes	
NFR5	Compatibility	The application should be compatible with a wide range of web browsers and devices	The application should be tested on different browsers and devices to ensure compatibility	Low	Yes	
NFR6	Scalability	The application should be able to handle an increasing number of users and data	The application should be designed with scalability in mind and able to scale up or down as needed	High	Yes	

NFR7	Accessibility	The application should be accessible to users with disabilities	The application should comply with accessibility standards and provide alternative text for images, keyboard navigation, and other accessibility features	Medium	Yes	
NFR8	Maintainability	The application should be easy to maintain and update	The application should be designed with maintainability in mind, with clear code structure, comments, and documentation	Low	Yes	

Architecture Diagram:



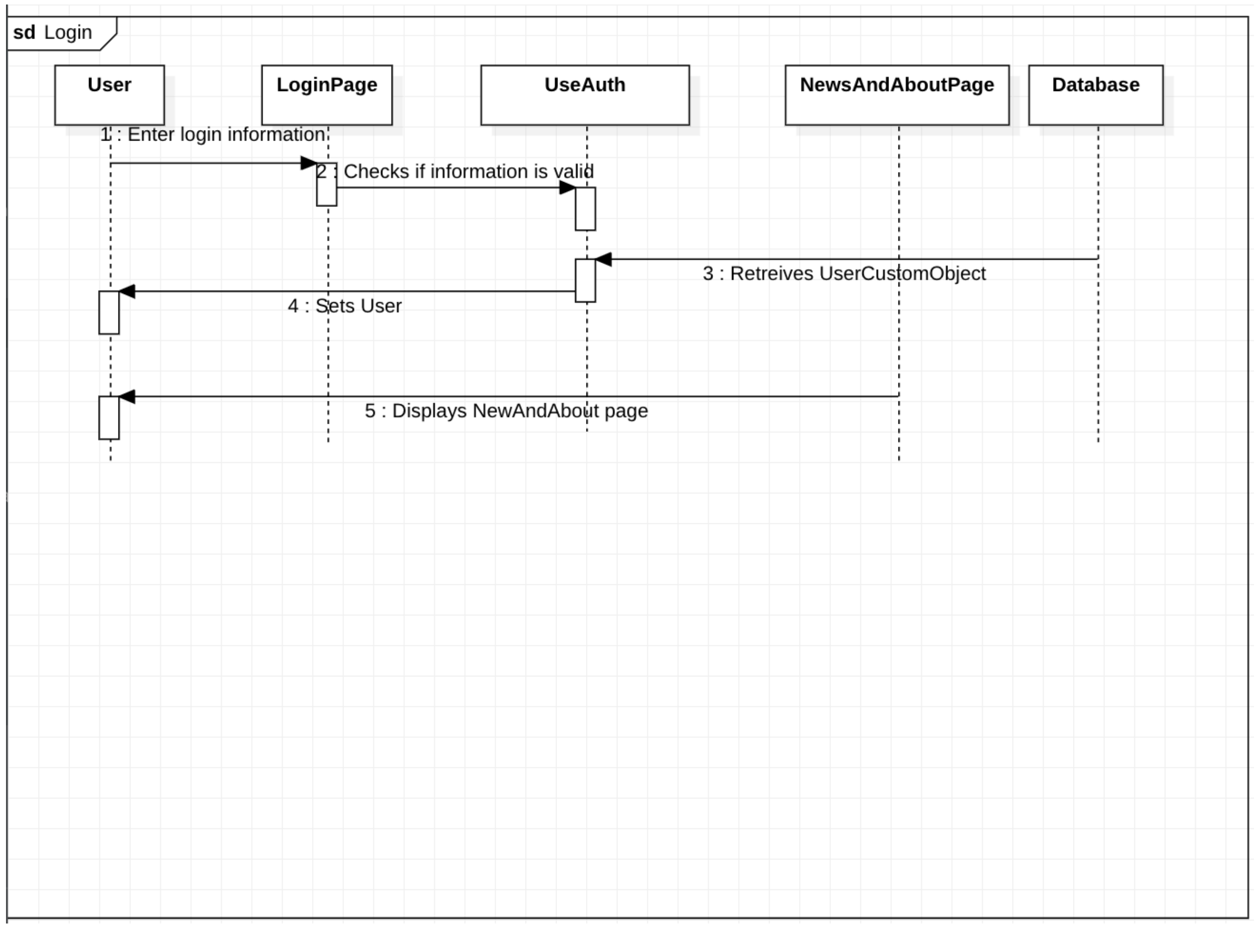
The architecture diagram presents a clear picture of the various components and their relationships within the Digital Flora application. At the top of the diagram, the frontend layer is responsible for the user interface and presentation aspects of the application.

The frontend layer connects to three primary components: Garden, Discussion, and News/About. These components function as controllers, taking care of the application's logic and user interactions in their specific areas.

The Garden component deals with all user interactions related to the virtual garden, such as planting, watering, and taking care of plants.
The Discussion component oversees the discussion board, where users can share their thoughts, reply to others, and engage in conversations.
The News/About component is in charge of updating users on new plant availability and other relevant information, as well as providing background information about the application and its purpose.
The Garden and Discussion components interact with the database, which stores user and plant data. This database acts as the model in the MVC architecture, handling data storage and retrieval. The Garden component communicates with the database to update plant information and user actions, and the Discussion component interacts with the database to store and retrieve user discussions.

In turn, the database provides the essential data to the Garden and Discussion components, allowing them to provide users with up-to-date information. This two-way communication between the components and the database keeps the programme up to date and responsive to user activities.

Sequence Diagram(s):



sd Login
User
LoginPage
UseAuth
NewsAndAboutPage
Database
1 : Enter login information
2 : Checks if information is valid
3 : Retreives UserCustomObject
4 : Sets User
5 : Displays NewAndAbout page

The sequence diagram for the login process in the Digital Flora application provides a detailed illustration of the interactions between the five participating components, which include User, Login Page, User Authentication (User Auth), News and About Page, and Database. The diagram demonstrates the step-by-step process of user login, from entering login information to accessing the News and About Page.

1. Enter login information: The login procedure is initiated by the user entering their login credentials (username and password) on the Login Page.

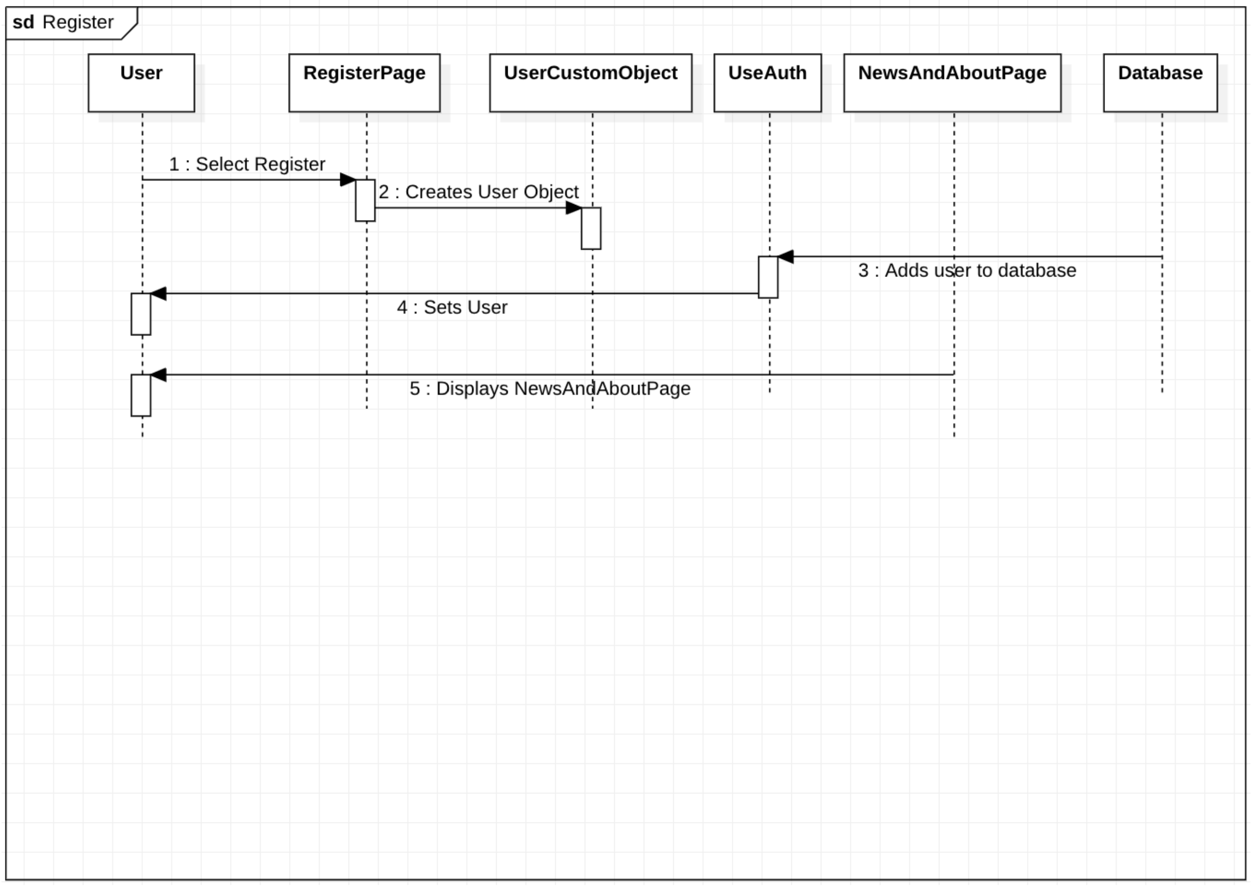
2. Checks if information is valid: The Login Page delivers the entered credentials to the User Auth component, which is in charge of validating the provided information.

3. Retrieve UserCustom object: If the entered credentials are valid, the User Auth component communicates with the Database to retrieve the corresponding UserCustom object, which contains user-specific information and preferences.

4. Set user: After retrieving the UserCustom object, the User Auth component sets the User object with the retrieved data, effectively logging the user into the system.

5. Display News and About Page: Finally, upon successful login, the News and About Page is displayed to the user, presenting them with the latest plant-related news and background information about the application.

The sequence diagram clearly demonstrates the flow of interactions and the roles of each component in the login process. This representation helps developers understand the system's behavior and ensures proper implementation of the login functionality.



The sequence diagram for the register process in the Digital Flora application provides a detailed illustration of the interactions between the six participating components, which include User, Register Page, User Custom Object, User Authentication (User Auth), News and About Page, and Database. The diagram demonstrates the step-by-step process of user registration, from selecting the register option to accessing the News and About Page.

1. Select register: The registration process is initiated by the user selecting the register option, which leads them to the Register Page.

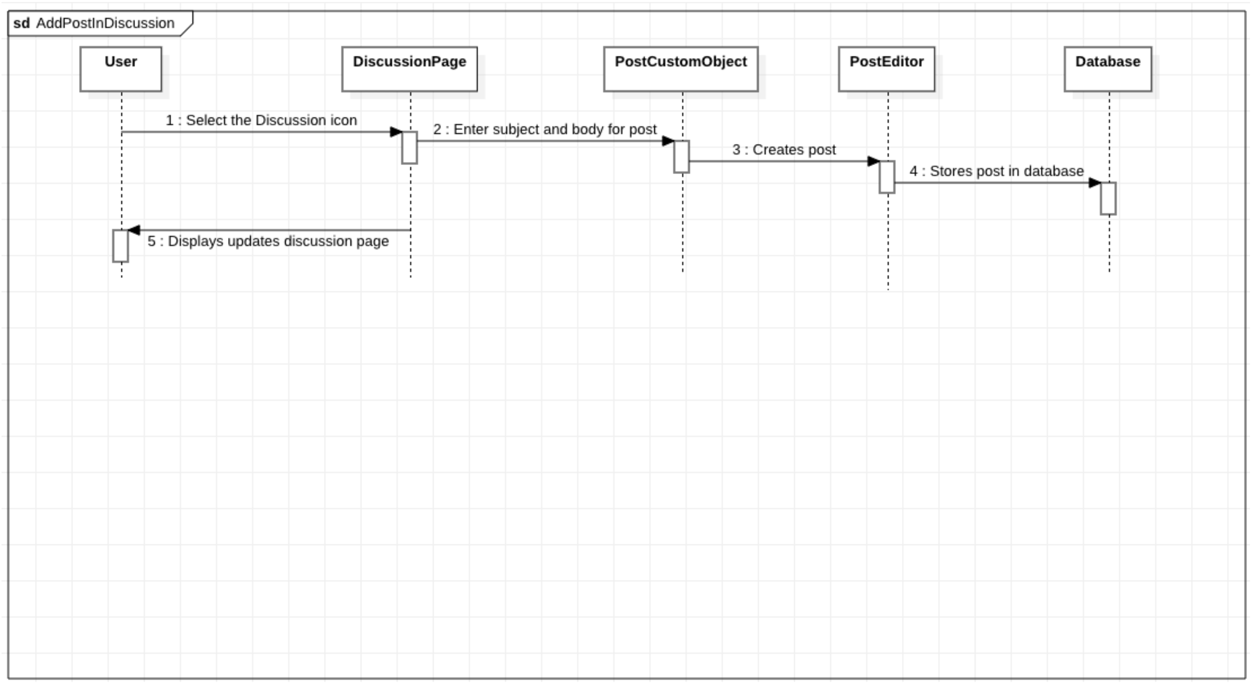
2. Create User object: The Register Page collects the necessary information from the user and creates a User Custom Object containing the user's data, such as their username, password, and email address.

3. Add user to Database: The User Custom Object is sent to the User Auth component, which is in charge of adding the new user to the Database.

4. Set user: After the new user has been added to the Database, the User Auth component sets the User object with the corresponding data, effectively creating a new user account in the system.

5. Display News and About Page: Finally, upon successful registration, the News and About Page is displayed to the user, presenting them with the latest plant-related news and background information about the application.

The sequence diagram clearly demonstrates the flow of interactions and the roles of each component in the registration process. This representation helps developers understand the system's behavior and ensures proper implementation of the registration functionality.


sd AddPostInDiscussion
User
DiscussionPage
PostCustomObject
PostEditor
Database
1 : Select the Discussion icon
2 : Enter subject and body for post
3 : Creates post
4 : Stores post in database
5 : Displays updates discussion page

The sequence diagram for the add post in discussion process in the Digital Flora application provides a detailed illustration of the interactions between the five participating components, which include User, Discussion Page, Post Custom Object, Post Editor, and Database. The diagram demonstrates the step-by-step process of adding a post to the discussion page, from selecting the discussion icon to displaying the updated discussion page.

1. Select the discussion icon: The add post process is initiated by the user selecting the discussion icon, which leads them to the Discussion Page.

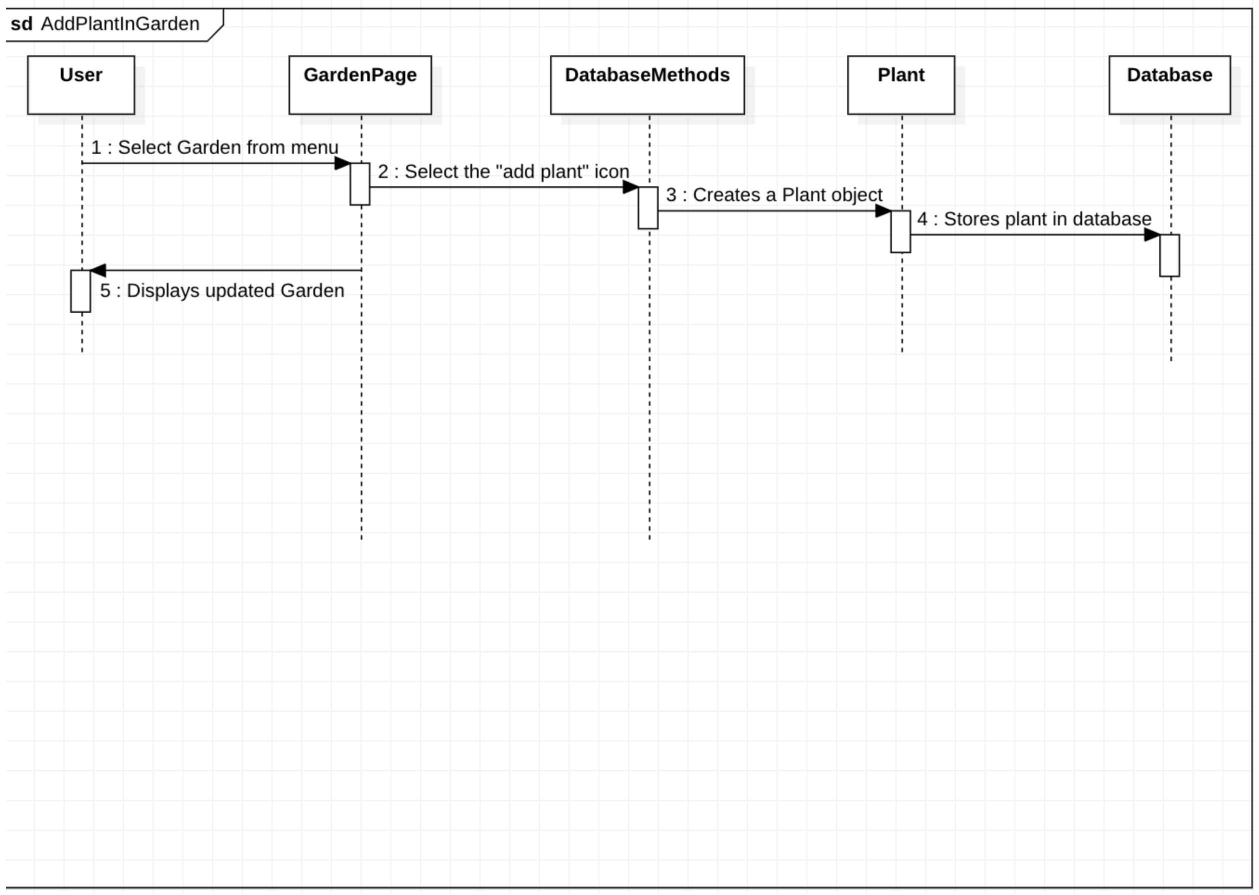
2. Enter subject and body for post: On the Discussion Page, the user enters the subject and body of their post, which are then sent to the Post Custom Object.

3. Create post: The Post Custom Object receives the entered data and creates a post by sending the data to the Post Editor component.

4. Store post in Database: The Post Editor component, after creating the post, communicates with the Database to store the new post, ensuring it becomes a part of the discussion.

5. Display updated discussion page: Finally, upon successful storage of the post in the Database, the Discussion Page displays the updated discussion page, now featuring the user's new post, to the user and other participants.

The sequence diagram clearly demonstrates the flow of interactions and the roles of each component in the add post process. This representation helps developers understand the system's behavior and ensures proper implementation of the add post functionality.


sd AddPlantInGarden
User
GardenPage
DatabaseMethods
Plant
Database
1 : Select Garden from menu
2 : Select the "add plant" icon
3 : Creates a Plant object
4 : Stores plant in database
5 : Displays updated Garden

The sequence diagram for the add plant in garden process in the Digital Flora application provides a detailed illustration of the interactions between the five participating components, which include User, Garden Page, Database Methods, Plant, and Database. The diagram demonstrates the step-by-step process of adding a plant to the user's garden, from selecting the garden option to displaying the updated garden.

1. Select garden from menu: The add plant process is initiated by the user selecting the garden option from the menu, which leads them to the Garden Page.

2. Select the "add plant" icon: On the Garden Page, the user selects the "add plant" icon, triggering the Database Methods component to create a new plant.

3. Create a Plant object: The Database Methods component, after receiving the request to create a new plant, generates a Plant object containing the necessary information about the plant, such as its type, growth stage, and unique ID.

4. Store plant in Database: The Plant object, once created, communicates with the Database to store the new plant, ensuring it becomes a part of the user's garden.

5. Display updated garden: Finally, upon successful storage of the plant in the Database, the Garden Page displays the updated garden view to the user, now featuring the newly added plant.

The sequence diagram clearly demonstrates the flow of interactions and the roles of each component in the add plant process. This representation helps developers understand the system's behavior and ensures proper implementation of the add plant functionality.

Explanation - Why OOP would not work:

OOP would not be a good decision for a project such as this. Web applications typically involve lots of network communications and are very complex. OOP would be very exhausting for something like this. It would clutter the code and make it very hard to read and also implement since one would have to make a significant amount of classes and definitions. Implementing this also raises another problem. Unlike any other software system, a web application has multiple inferences and components. If we implemented an OOP system, despite the exhaustive nature, it would ruin the performance of our web application from very slow loading times to having multiple system crashes.

Testing:

User registration and login:

1. Test that users can successfully register for an account and log in with their credentials.
2. Test that users cannot register with invalid information (e.g., invalid email address, password length).

Personal garden:

1. Test that each user has a personal garden where they can purchase plants using earned coins.
2. Test that users cannot purchase plants if they do not have enough coins.
3. Test that users can only have a maximum of three plants in their garden.

Plant care:

1. Test that users can water their plants every 30 minutes to keep them healthy.
2. Test that neglecting a plant for two days will result in its death, allowing for planting a new one.
3. Test that users can only water their own plants and not other users' plants.

Plant trading:

1. Test that users can trade their plants with interested users for coins.
2. Test that users can only trade plants that they own and not plants that belong to other users.
3. Test that users cannot trade a plant that is already dead.

News section:

1. Test that users can access the news section to stay informed about newly available plants.
2. Test that new plants are added to the news section as they become available.

Discussion board:

1. Test that users can post their thoughts.
2. Test that users can write multiple posts.

Coin system:

1. Test that users can use coins to purchase plants.
2. Test that users can earn coins by selling and watering their plants.
3. Test that users cannot have a negative coin balance.

4. Test that the plant care and growth simulation is realistic and accurately reflects the needs of each plant type.

Database integration:

1. Test that the game employs the Model-View-Controller (MVC) pattern for database integration and maintenance.
2. Test that the database can maintain the state of the virtual garden and keep track of plants and user progress.

Progress tracking:

1. Test that users can keep track of their plants over time and see the progress they have made in caring for their virtual garden.
2. Test that the database accurately reflects the state of each user's garden.

User rewards:

1. Test that the game includes a system of credits to reward users for their plant care efforts.
2. Test that users are appropriately rewarded for their plant care efforts, and that the reward system is fair and consistent.

By testing the Digital Flora web-based game using these test cases, developers can ensure that the game functions as intended and provides a rewarding and engaging user experience for its intended audience.

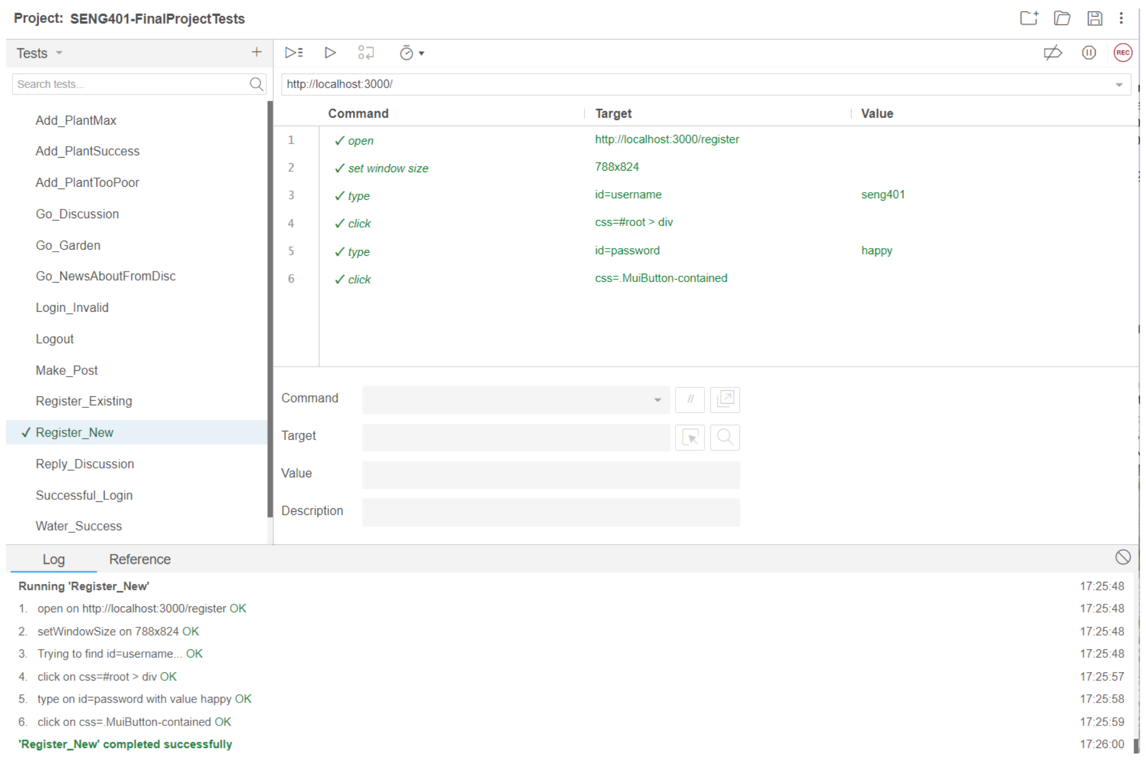
Test case #	**Use Case**	**Function being tested**	**Initial System state**	**Input**	**Expected Output**
1	Successful User Registration	Test that users can successfully register for an account	User is asked to enter their new username and password	Username: seng401 Password: happy	Takes you to the home page of your account
2	Unsuccessful User Registration	Test that users that are already registered cannot register again with the same credentials	User is asked to enter their new username and password	Username: michie Password: 1234	Popup notification that user already exists

3	Successful login	Test that users can log in with their credentials	User is asked to enter their username and password	Username: Password:	News and about page is displayed
4	Unsuccessful login	Test that users can not log in with incorrect credentials	User is asked to enter their username and password	Username: Password:	Popup notification that username or password is incorrect
5	Logout	Test that users can logout	User is on any of the given pages	Click log out button	News and About page is displayed
6	Go to Discussion page from News and About	Test that users can access discussion page from news and about page	User is on the news and about page	Click discussion page button on dashboard	News and About page is displayed
7	Go to Garden page from News and About	Test that users can access garden page from news and about page	User is on the news and about page	Click garden page button on dashboard	Garden page is displayed
8	Go to News and About from Discussion page	Test that users can access news and about page from discussion page	User is on the discussion page	Click news and about page button on dashboard	News and About page is displayed
9	Add plant successful	Test that users can add plants if they are sufficient gold and capacity	User is on the garden page	Click add plant button	A new plant appears
10	Add plant unsuccessful (max reached)	Test that users can not add plant if there is not enough capacity	User is on the garden page	Click add plant button	Popup notification appears that says “User has too many plants already or is too poor (hehehe poor moment)”
11	Add plant unsuccessful (not enough money)	Test that users can not add plant if there is not enough gold	User is on the garden page	Click add plant button	Popup notification appears that says “User has too

					many plants already or is too poor (hehehe poor moment)”
12	Remove plant	Test that users can delete their plants	User is on the garden page	Click garbage button	Plant is removed from display
13	Water successful	Test that users can water plants	User is on the garden page	Click water button	Number of gold increases
14	Water unsuccessful (too soon)	Test that users can not water plants if they have recently watered plant	User is on the garden page	Click water button	Popup notification saying “Please wait 2 hours before re-watering your plants (it has been __ min)
15	Make post	Test that users can make a post	User is on the discussion page	User enters a Subject: and Body: then Click post	The post should be displayed as a chain in the discussion posts with new posts at the top
16	Reply to post	Test that users can reply to posts	User is on the discussion page	User clicks on a post they want to reply to, Enter a Reply, then click post,	The reply should show up under the post, and the number of replies beside the post should increment for every new reply

Automated Integration Testing Using Selenium:

TC 1 - Successful Register



Project: SEng401-FinalProjectTests

Tests

Search tests...

- Add_PlantMax
- Add_PlantSuccess
- Add_PlantTooPoor
- Go_Discussion
- Go_Garden
- Go_NewsAboutFromDisc
- Login_Invalid
- Logout
- Make_Post
- Register_Existing
- ✓ Register_New
- Reply_Discussion
- Successful_Login
- Water_Success

http://localhost:3000/

	Command	Target	Value
1	✓ open	http://localhost:3000/register	
2	✓ set window size	788x824	
3	✓ type	id=username	seng401
4	✓ click	css=#root > div	
5	✓ type	id=password	happy
6	✓ click	css=.MuiButton-contained	

Command

Target

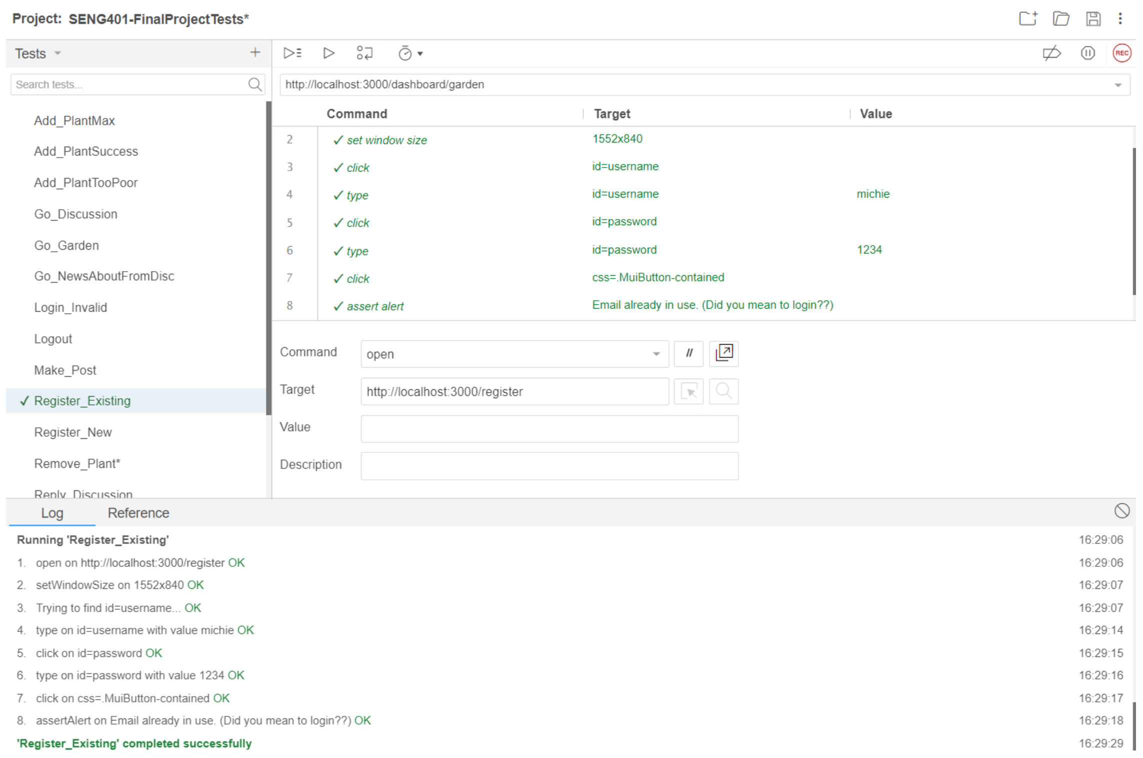
Value

Description

Log Reference

```
Running 'Register_New'                                         17:25:48
1. open on http://localhost:3000/register OK                   17:25:48
2. setWindowSize on 788x824 OK                                 17:25:48
3. Trying to find id=username... OK                            17:25:48
4. click on css=#root > div OK                                 17:25:57
5. type on id=password with value happy OK                     17:25:58
6. click on css=.MuiButton-contained OK                        17:25:59
'Register_New' completed successfully                          17:26:00
```

TC 2 - Register with an existing account



Project: SEng401-FinalProjectTests*

Tests

Search tests...

- Add_PlantMax
- Add_PlantSuccess
- Add_PlantTooPoor
- Go_Discussion
- Go_Garden
- Go_NewsAboutFromDisc
- Login_Invalid
- Logout
- Make_Post
- ✓ Register_Existing
- Register_New
- Remove_Plant*
- Reply_Discussion

http://localhost:3000/dashboard/garden

	Command	Target	Value
2	✓ set window size	1552x840	
3	✓ click	id=username	
4	✓ type	id=username	michie
5	✓ click	id=password	
6	✓ type	id=password	1234
7	✓ click	css=.MuiButton-contained	
8	✓ assert alert	Email already in use. (Did you mean to login??)	

Command: open

Target: http://localhost:3000/register

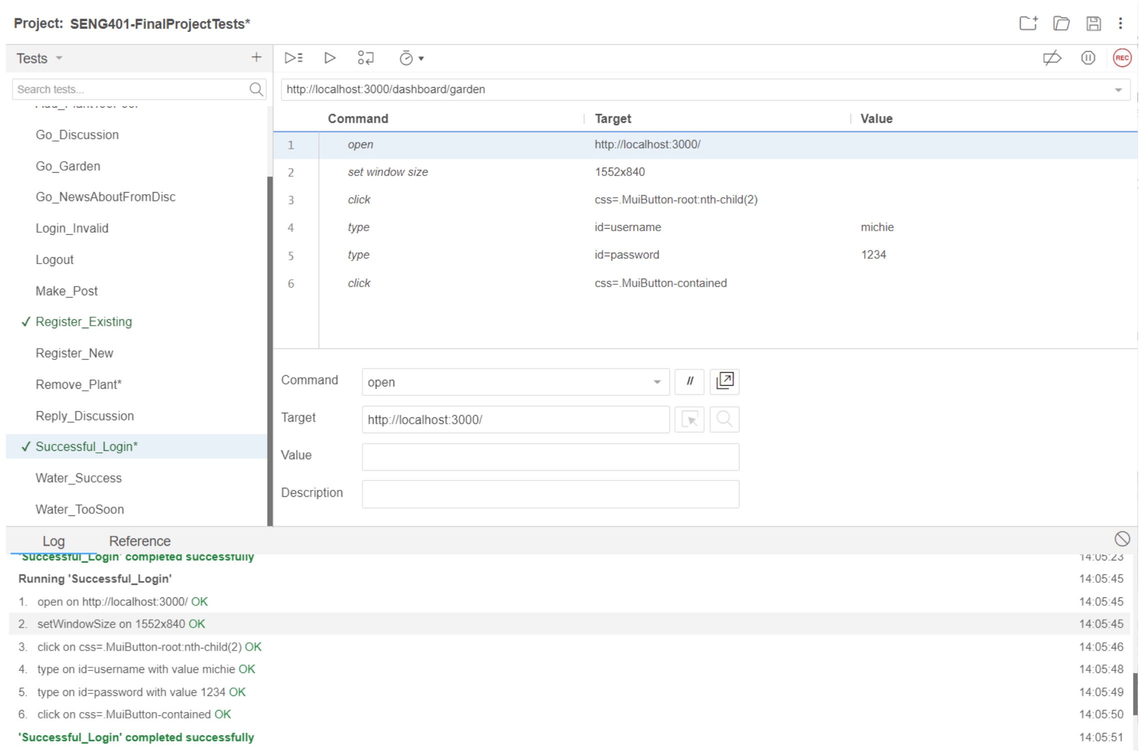
Value

Description

Log Reference

```
Running 'Register_Existing'                                                  16:29:06
1. open on http://localhost:3000/register OK                                 16:29:06
2. setWindowSize on 1552x840 OK                                              16:29:07
3. Trying to find id=username... OK                                          16:29:07
4. type on id=username with value michie OK                                  16:29:14
5. click on id=password OK                                                   16:29:15
6. type on id=password with value 1234 OK                                    16:29:16
7. click on css=.MuiButton-contained OK                                      16:29:17
8. assertAlert on Email already in use. (Did you mean to login??) OK         16:29:18
'Register_Existing' completed successfully                                   16:29:29
```

TC 3 - Successful login



Project: SENG401-FinalProjectTests*

Tests

Search tests...

Go_Discussion
Go_Garden
Go_NewsAboutFromDisc
Login_Invalid
Logout
Make_Post
✓ Register_Existing
Register_New
Remove_Plant*
Reply_Discussion
✓ Successful_Login*
Water_Success
Water_TooSoon

http://localhost:3000/dashboard/garden

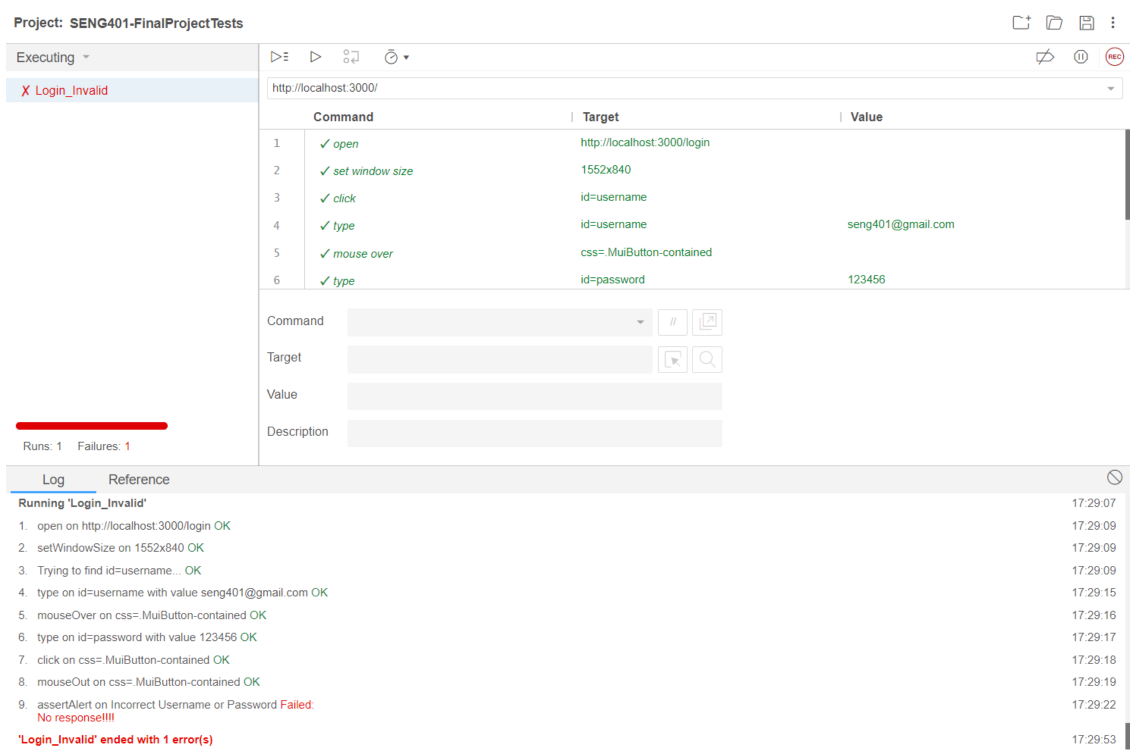
	Command	Target	Value
1	open	http://localhost:3000/	
2	set window size	1552x840	
3	click	css=.MuiButton-root:nth-child(2)	
4	type	id=username	michie
5	type	id=password	1234
6	click	css=.MuiButton-contained	

Command: open
Target: http://localhost:3000/
Value:
Description:

Log Reference

```
Running 'Successful_Login'                                         14:05:45
1. open on http://localhost:3000/ OK                               14:05:45
2. setWindowSize on 1552x840 OK                                    14:05:45
3. click on css=.MuiButton-root:nth-child(2) OK                    14:05:46
4. type on id=username with value michie OK                        14:05:48
5. type on id=password with value 1234 OK                          14:05:49
6. click on css=.MuiButton-contained OK                            14:05:50
'Successful_Login' completed successfully                          14:05:51
```

TC 4 - Unsuccessful login



Project: SENG401-FinalProjectTests

Executing

✗ Login_Invalid

http://localhost:3000/

	Command	Target	Value
1	✓ open	http://localhost:3000/login	
2	✓ set window size	1552x840	
3	✓ click	id=username	
4	✓ type	id=username	seng401@gmail.com
5	✓ mouse over	css=.MuiButton-contained	
6	✓ type	id=password	123456

Command
Target
Value
Description

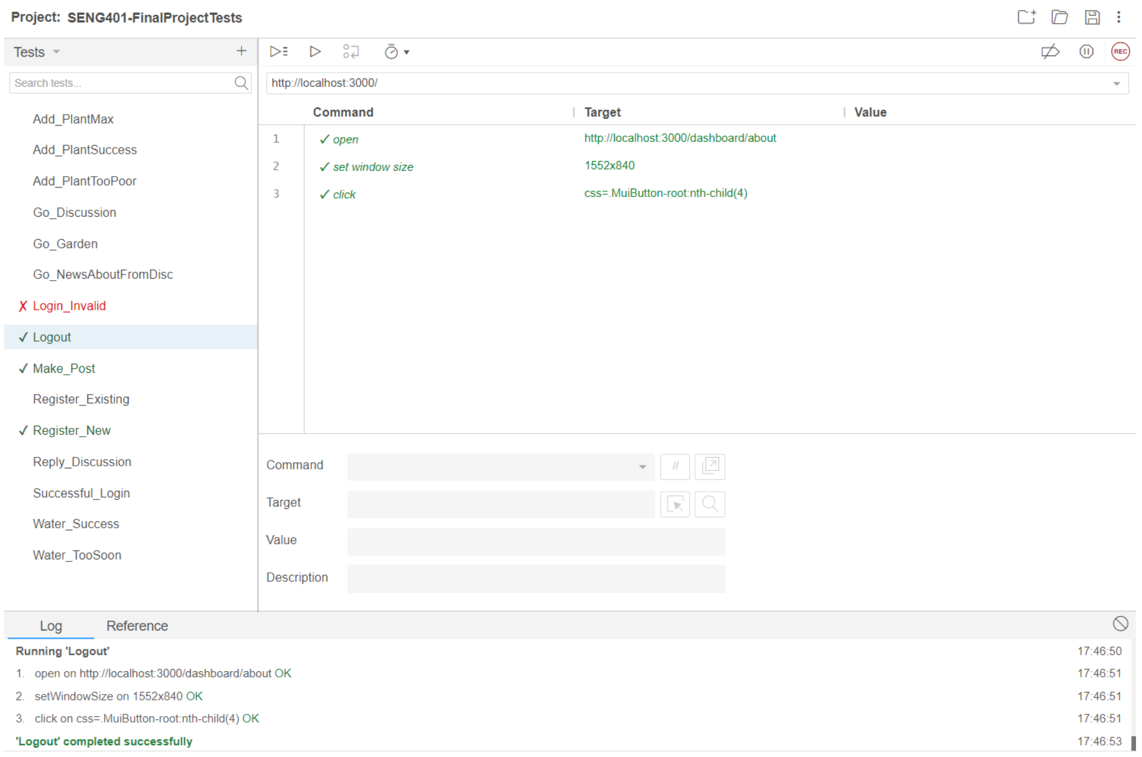
Runs: 1 Failures: 1

Log Reference

```
Running 'Login_Invalid'                                            17:29:07
1. open on http://localhost:3000/login OK                          17:29:09
2. setWindowSize on 1552x840 OK                                    17:29:09
3. Trying to find id=username... OK                                17:29:09
4. type on id=username with value seng401@gmail.com OK             17:29:15
5. mouseOver on css=.MuiButton-contained OK                        17:29:16
6. type on id=password with value 123456 OK                        17:29:17
7. click on css=.MuiButton-contained OK                            17:29:18
8. mouseOut on css=.MuiButton-contained OK                         17:29:19
9. assertAlert on Incorrect Username or Password Failed:           17:29:22
   No response!!!!
'Login_Invalid' ended with 1 error(s)                              17:29:53
```

The only error resulted from not responding to the pop-up alert within a timely manner so the test case timed out.

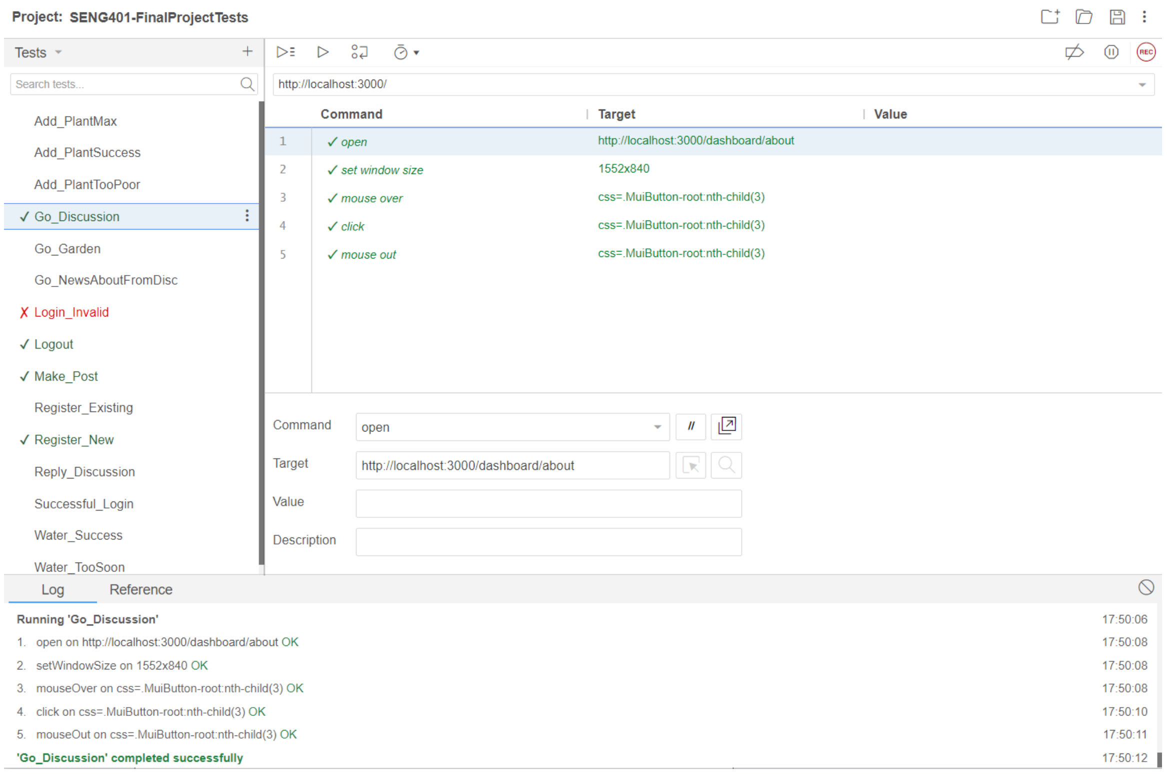
TC 5 - Logout



	Command	Target	Value
1	✓ open	http://localhost:3000/dashboard/about	
2	✓ set window size	1552x840	
3	✓ click	css=.MuiButton-root:nth-child(4)	

```
Running 'Logout'                                                    17:46:50
1. open on http://localhost:3000/dashboard/about OK                 17:46:51
2. setWindowSize on 1552x840 OK                                     17:46:51
3. click on css=.MuiButton-root:nth-child(4) OK                     17:46:51
'Logout' completed successfully                                     17:46:53
```

TC 6 - Go to Discussion page from News and About page

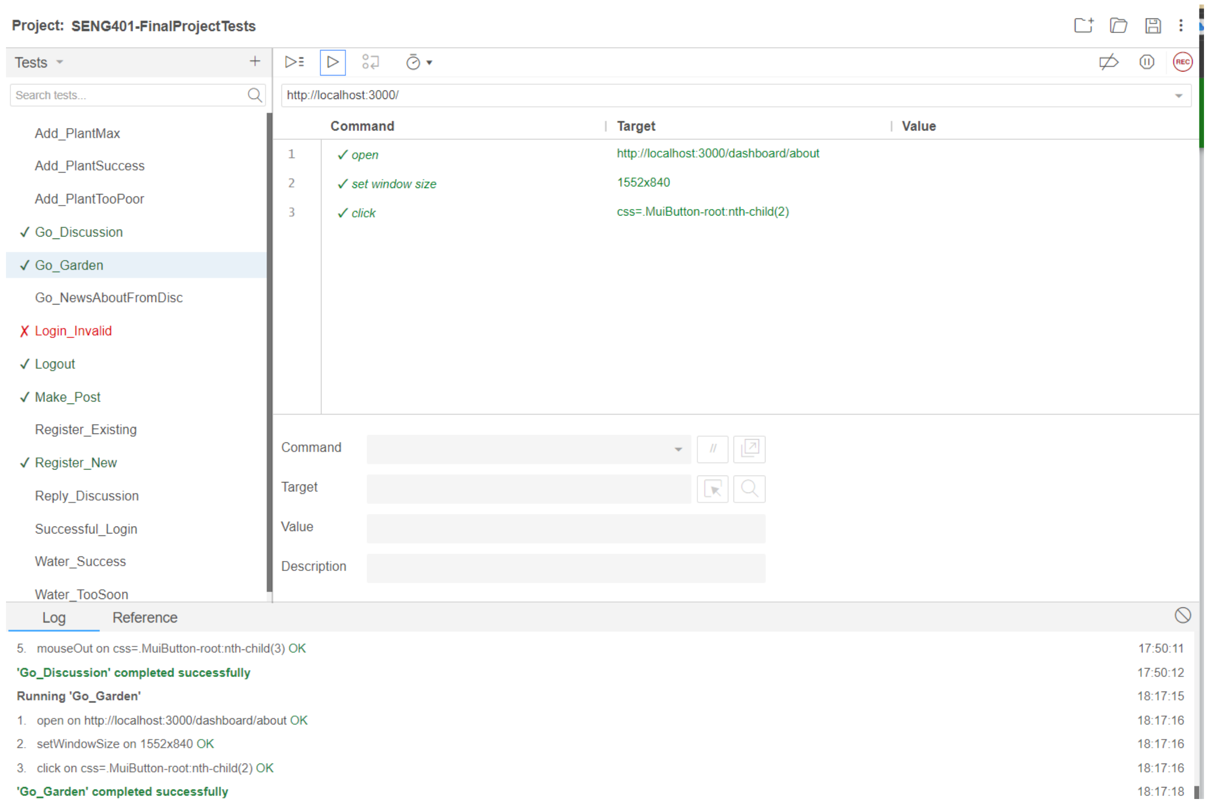


	Command	Target	Value
1	✓ open	http://localhost:3000/dashboard/about	
2	✓ set window size	1552x840	
3	✓ mouse over	css=.MuiButton-root:nth-child(3)	
4	✓ click	css=.MuiButton-root:nth-child(3)	
5	✓ mouse out	css=.MuiButton-root:nth-child(3)	

Command: open
Target: http://localhost:3000/dashboard/about
Value:
Description:

```
Running 'Go_Discussion'                                             17:50:06
1. open on http://localhost:3000/dashboard/about OK                 17:50:08
2. setWindowSize on 1552x840 OK                                     17:50:08
3. mouseOver on css=.MuiButton-root:nth-child(3) OK                 17:50:08
4. click on css=.MuiButton-root:nth-child(3) OK                     17:50:10
5. mouseOut on css=.MuiButton-root:nth-child(3) OK                  17:50:11
'Go_Discussion' completed successfully                              17:50:12
```

TC 7 - Go to Garden page from Discussion page


Project: SENG401-FinalProjectTests

Tests

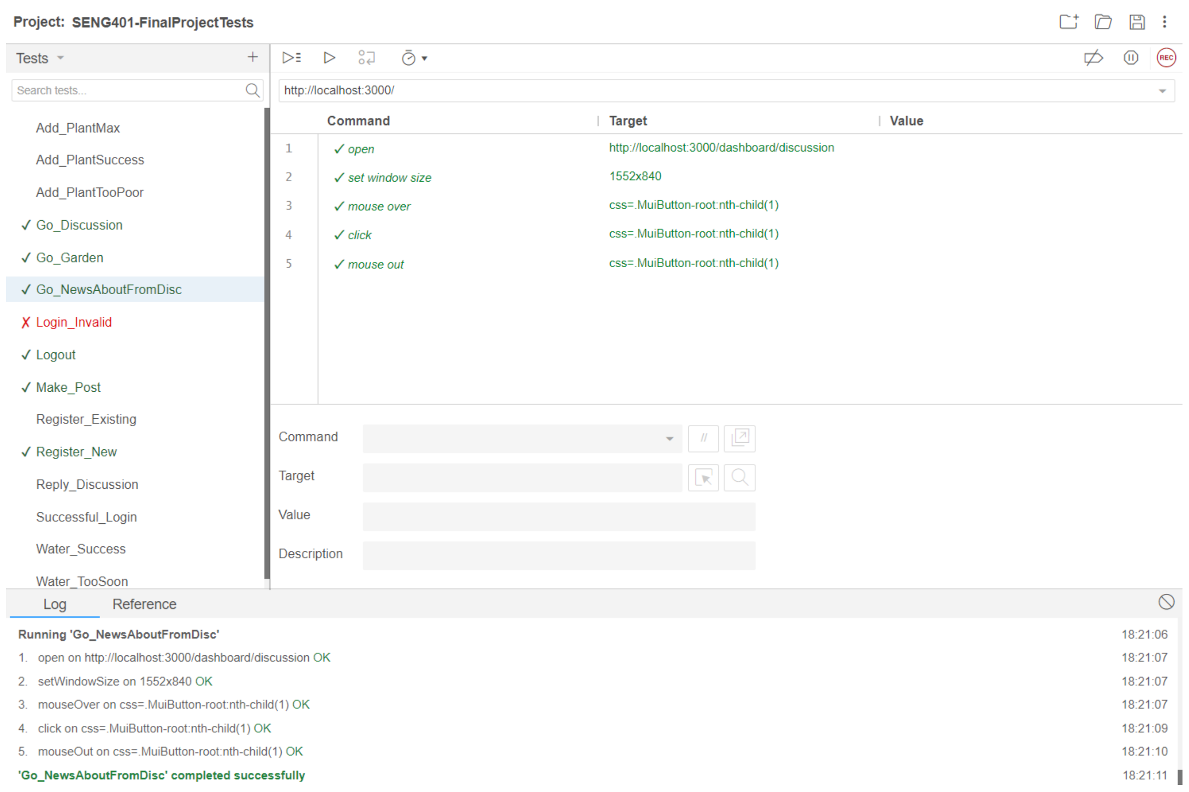
- Add_PlantMax
- Add_PlantSuccess
- Add_PlantTooPoor
- ✓ Go_Discussion
- ✓ Go_Garden
- Go_NewsAboutFromDisc
- ✗ Login_Invalid
- ✓ Logout
- ✓ Make_Post
- Register_Existing
- ✓ Register_New
- Reply_Discussion
- Successful_Login
- Water_Success
- Water_TooSoon

http://localhost:3000/

	Command	Target	Value
1	✓ open	http://localhost:3000/dashboard/about	
2	✓ set window size	1552x840	
3	✓ click	css=.MuiButton-root:nth-child(2)	

```
5.  mouseOut on css=.MuiButton-root:nth-child(3) OK                    17:50:11
'Go_Discussion' completed successfully                                 17:50:12
Running 'Go_Garden'                                                    18:17:15
1.  open on http://localhost:3000/dashboard/about OK                   18:17:16
2.  setWindowSize on 1552x840 OK                                       18:17:16
3.  click on css=.MuiButton-root:nth-child(2) OK                       18:17:16
'Go_Garden' completed successfully                                     18:17:18
```

TC 8 - Go to News and About page from Discussion page


Project: SENG401-FinalProjectTests

Tests

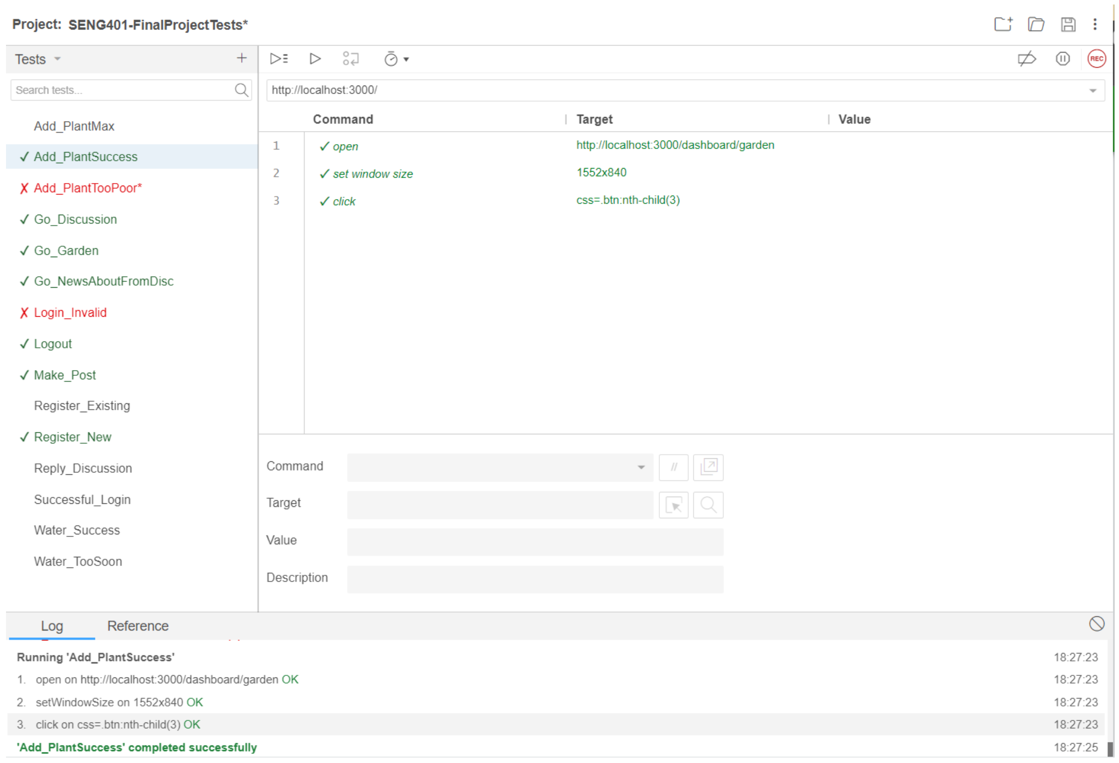
- Add_PlantMax
- Add_PlantSuccess
- Add_PlantTooPoor
- ✓ Go_Discussion
- ✓ Go_Garden
- ✓ Go_NewsAboutFromDisc
- ✗ Login_Invalid
- ✓ Logout
- ✓ Make_Post
- Register_Existing
- ✓ Register_New
- Reply_Discussion
- Successful_Login
- Water_Success
- Water_TooSoon

http://localhost:3000/

	Command	Target	Value
1	✓ open	http://localhost:3000/dashboard/discussion	
2	✓ set window size	1552x840	
3	✓ mouse over	css=.MuiButton-root:nth-child(1)	
4	✓ click	css=.MuiButton-root:nth-child(1)	
5	✓ mouse out	css=.MuiButton-root:nth-child(1)	

```
Running 'Go_NewsAboutFromDisc'                                         18:21:06
1.  open on http://localhost:3000/dashboard/discussion OK              18:21:07
2.  setWindowSize on 1552x840 OK                                       18:21:07
3.  mouseOver on css=.MuiButton-root:nth-child(1) OK                   18:21:07
4.  click on css=.MuiButton-root:nth-child(1) OK                       18:21:09
5.  mouseOut on css=.MuiButton-root:nth-child(1) OK                    18:21:10
'Go_NewsAboutFromDisc' completed successfully                          18:21:11
```

TC 9 - Add plant successful



Project: SENG401-FinalProjectTests*

Tests

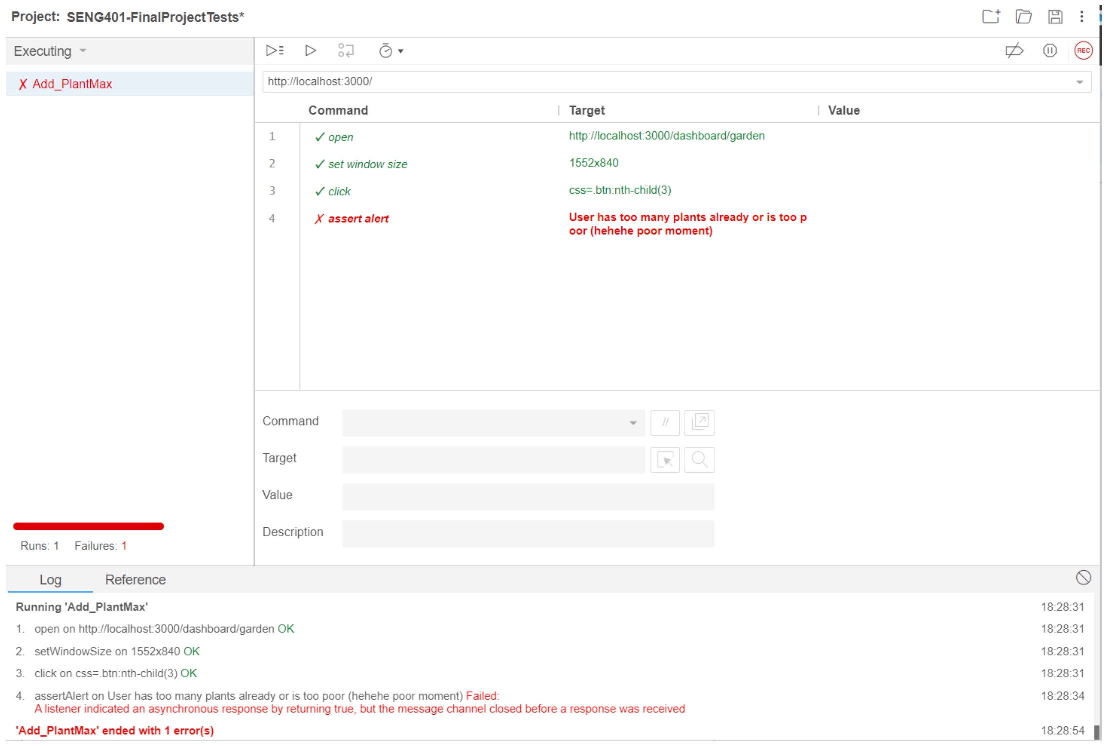
- Add_PlantMax
- ✓ Add_PlantSuccess
- ✗ Add_PlantTooPoor*
- ✓ Go_Discussion
- ✓ Go_Garden
- ✓ Go_NewsAboutFromDisc
- ✗ Login_Invalid
- ✓ Logout
- ✓ Make_Post
- Register_Existing
- ✓ Register_New
- Reply_Discussion
- Successful_Login
- Water_Success
- Water_TooSoon

http://localhost:3000/

	Command	Target	Value
1	✓ open	http://localhost:3000/dashboard/garden	
2	✓ set window size	1552x840	
3	✓ click	css=.btn:nth-child(3)	

```
Running 'Add_PlantSuccess'                                              18:27:23
1. open on http://localhost:3000/dashboard/garden OK                    18:27:23
2. setWindowSize on 1552x840 OK                                         18:27:23
3. click on css=.btn:nth-child(3) OK                                    18:27:23
'Add_PlantSuccess' completed successfully                               18:27:25
```

TC 10 - Add plant max reached



Project: SENG401-FinalProjectTests*

Executing

- ✗ Add_PlantMax

Runs: 1 Failures: 1

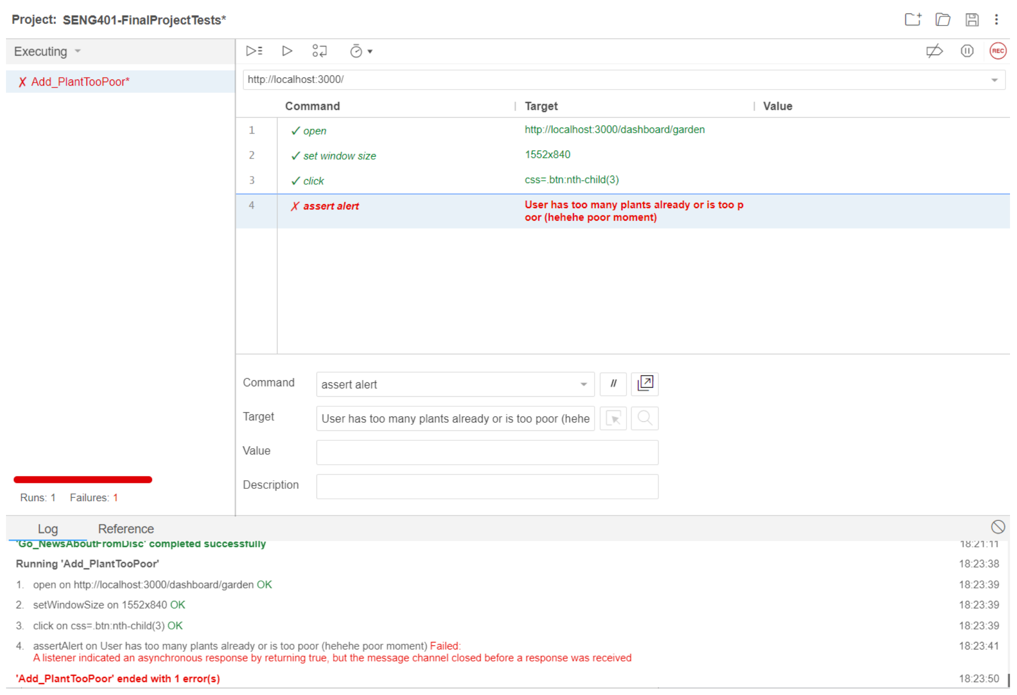
http://localhost:3000/

	Command	Target	Value
1	✓ open	http://localhost:3000/dashboard/garden	
2	✓ set window size	1552x840	
3	✓ click	css=.btn:nth-child(3)	
4	✗ assert alert	User has too many plants already or is too poor (hehehe poor moment)	

```
Running 'Add_PlantMax'                                                  18:28:31
1. open on http://localhost:3000/dashboard/garden OK                    18:28:31
2. setWindowSize on 1552x840 OK                                         18:28:31
3. click on css=.btn:nth-child(3) OK                                    18:28:31
4. assertAlert on User has too many plants already or is too poor (hehehe poor moment) Failed:   18:28:34
   A listener indicated an asynchronous response by returning true, but the message channel closed before a response was received
'Add_PlantMax' ended with 1 error(s)                                    18:28:54
```

The only error resulted from not responding to the pop-up alert within a timely manner so the test case timed out.

TC 11 - Add plant but not enough money


Project: SEN G401-FinalProjectTests*

Executing

✗ Add_PlantTooPoor*

http://localhost:3000/

	Command	Target	Value
1	✓ open	http://localhost:3000/dashboard/garden	
2	✓ set window size	1552x840	
3	✓ click	css=.btn:nth-child(3)	
4	✗ assert alert	User has too many plants already or is too poor (hehehe poor moment)	

Command: assert alert

Target: User has too many plants already or is too poor (hehe

Value:

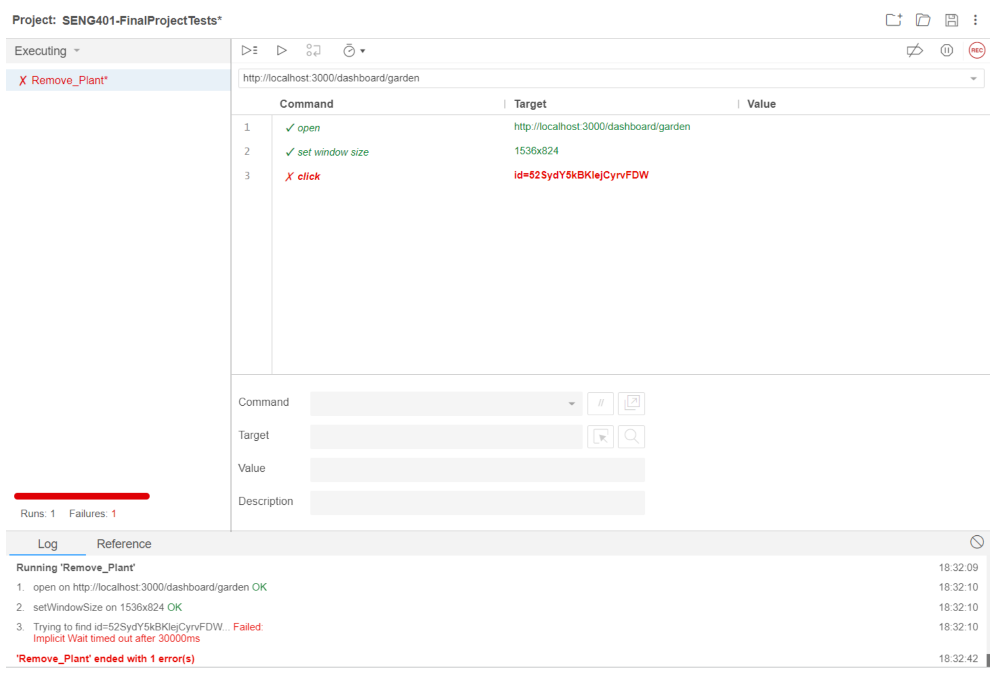
Description:

Runs: 1 Failures: 1

```
'Go_NewsAboutFromDisc' completed successfully                                              18:21:11
Running 'Add_PlantTooPoor'                                                                 18:23:38
1. open on http://localhost:3000/dashboard/garden OK                                       18:23:39
2. setWindowSize on 1552x840 OK                                                            18:23:39
3. click on css=.btn:nth-child(3) OK                                                       18:23:39
4. assertAlert on User has too many plants already or is too poor (hehehe poor moment) Failed:   18:23:41
   A listener indicated an asynchronous response by returning true, but the message channel closed before a response was received
'Add_PlantTooPoor' ended with 1 error(s)                                                   18:23:50
```

The only error resulted from not responding to the pop-up alert within a timely manner so the test case timed out.

TC 12 - Remove plant


Project: SENG401-FinalProjectTests*

Executing

✗ Remove_Plant*

http://localhost:3000/dashboard/garden

	Command	Target	Value
1	✓ open	http://localhost:3000/dashboard/garden	
2	✓ set window size	1536x824	
3	✗ click	id=52SydY5kBKlejCyrvFDW	

Command:

Target:

Value:

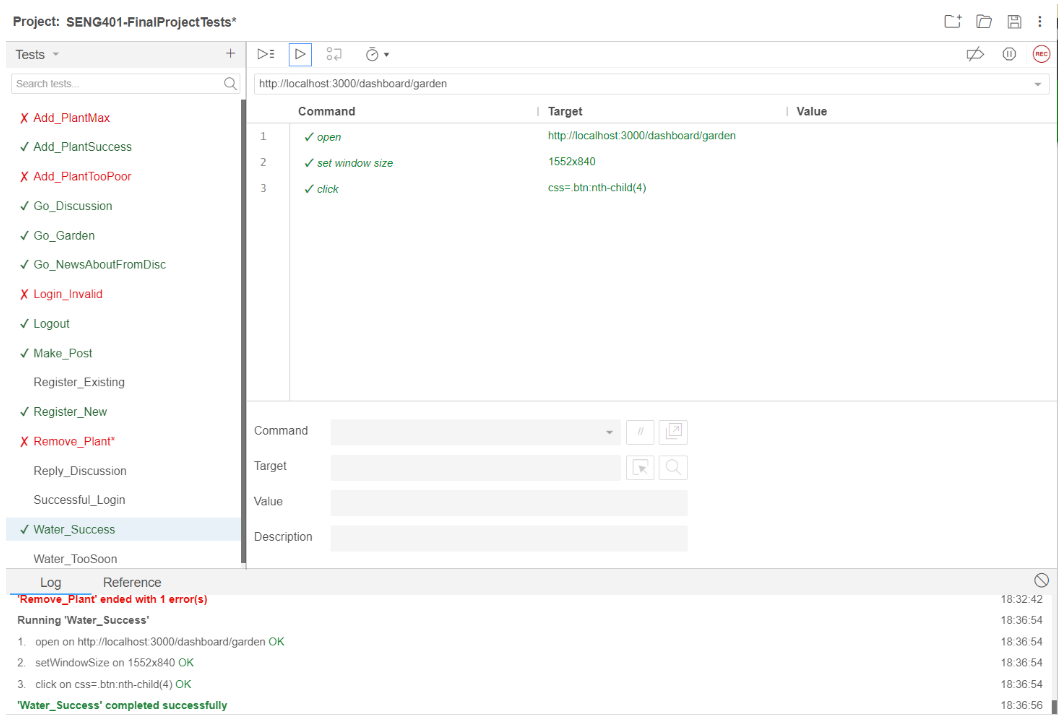
Description:

Runs: 1 Failures: 1

```
Running 'Remove_Plant'                                                  18:32:09
1. open on http://localhost:3000/dashboard/garden OK                    18:32:10
2. setWindowSize on 1536x824 OK                                         18:32:10
3. Trying to find id=52SydY5kBKlejCyrvFDW... Failed:                    18:32:10
   Implicit Wait timed out after 30000ms
'Remove_Plant' ended with 1 error(s)                                    18:32:42
```

The only error resulted when the remove button for the plant with that unique ID has been clicked. However, when writing the test, that specific ID has been removed already, so when the test case is played, it does not know what to remove since it is no longer in the database and causes an error.

TC 13 - Water plants successfully



Project: SENG401-FinalProjectTests*

Tests

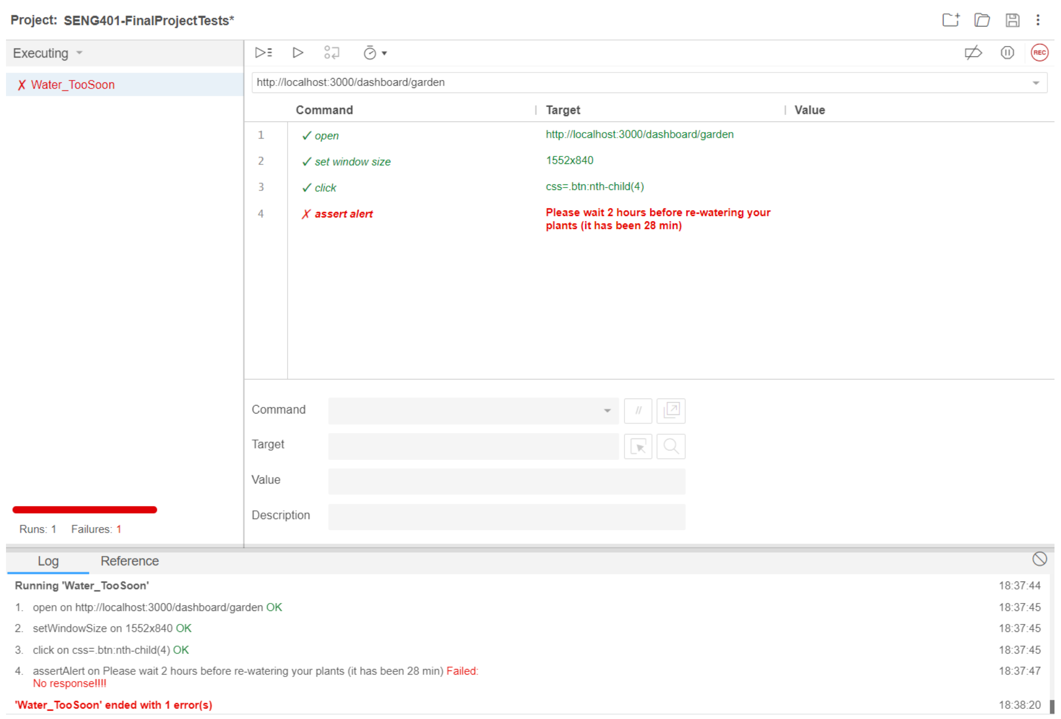
✗ Add_PlantMax
✓ Add_PlantSuccess
✗ Add_PlantTooPoor
✓ Go_Discussion
✓ Go_Garden
✓ Go_NewsAboutFromDisc
✗ Login_Invalid
✓ Logout
✓ Make_Post
Register_Existing
✓ Register_New
✗ Remove_Plant*
Reply_Discussion
Successful_Login
✓ Water_Success
Water_TooSoon

http://localhost:3000/dashboard/garden

	Command	Target	Value
1	✓ open	http://localhost:3000/dashboard/garden	
2	✓ set window size	1552x840	
3	✓ click	css=.btn:nth-child(4)	

```
'Remove_Plant' ended with 1 error(s)                          18:32:42
Running 'Water_Success'                                       18:36:54
1.  open on http://localhost:3000/dashboard/garden OK         18:36:54
2.  setWindowSize on 1552x840 OK                              18:36:54
3.  click on css=.btn:nth-child(4) OK                         18:36:54
'Water_Success' completed successfully                        18:36:56
```

TC 14 - Water plants unsuccessfully



Project: SENG401-FinalProjectTests*

Executing

✗ Water_TooSoon

http://localhost:3000/dashboard/garden

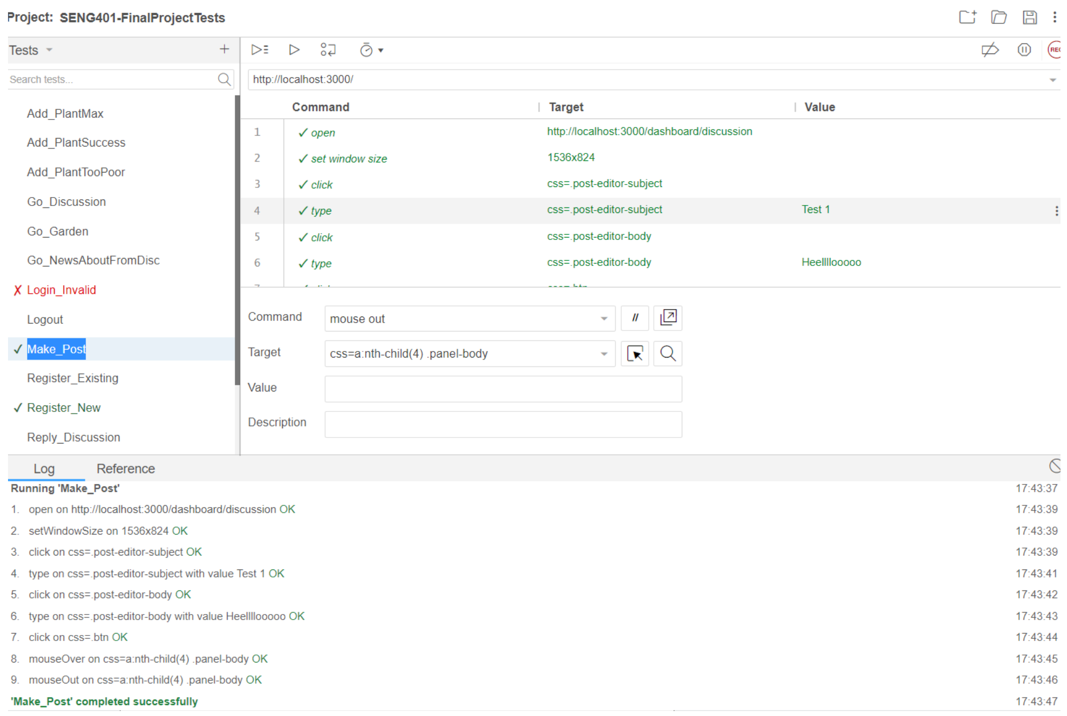
	Command	Target	Value
1	✓ open	http://localhost:3000/dashboard/garden	
2	✓ set window size	1552x840	
3	✓ click	css=.btn:nth-child(4)	
4	✗ assert alert	Please wait 2 hours before re-watering your plants (it has been 28 min)	

Runs: 1 Failures: 1

```
Running 'Water_TooSoon'                                                                      18:37:44
1.  open on http://localhost:3000/dashboard/garden OK                                        18:37:45
2.  setWindowSize on 1552x840 OK                                                             18:37:45
3.  click on css=.btn:nth-child(4) OK                                                        18:37:45
4.  assertAlert on Please wait 2 hours before re-watering your plants (it has been 28 min) Failed:  18:37:47
    No response!!!!
'Water_TooSoon' ended with 1 error(s)                                                        18:38:20
```

This error resulted because the alert message does not match the message that is currently displayed when the test case is ran again. The difference is the number of minutes. Since the down time is set at 2 hours or 120 minutes, the test case would pass exactly when the user has watered their plants exactly 28 minutes ago.

TC 15 - Make a post on discussion page


Project: SENG401-FinalProjectTests

Tests

- Add_PlantMax
- Add_PlantSuccess
- Add_PlantTooPoor
- Go_Discussion
- Go_Garden
- Go_NewsAboutFromDisc
- ✗ Login_Invalid
- Logout
- ✓ Make_Post
- Register_Existing
- ✓ Register_New
- Reply_Discussion

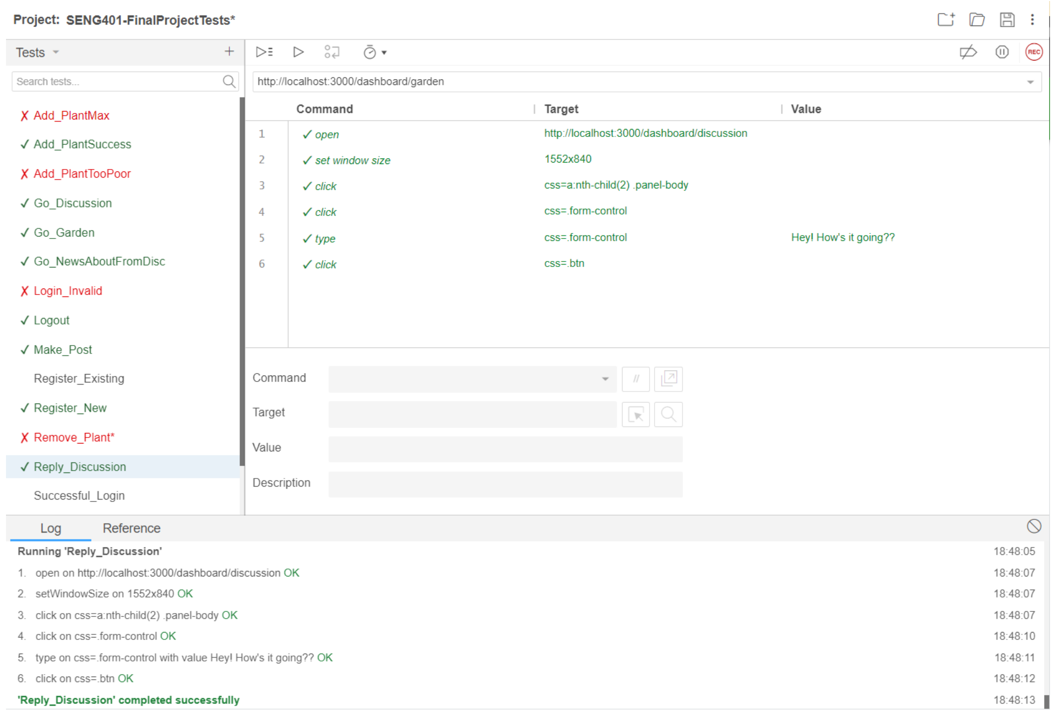
http://localhost:3000/

	Command	Target	Value
1	✓ open	http://localhost:3000/dashboard/discussion	
2	✓ set window size	1536x824	
3	✓ click	css=.post-editor-subject	
4	✓ type	css=.post-editor-subject	Test 1
5	✓ click	css=.post-editor-body	
6	✓ type	css=.post-editor-body	Heelllloooo

Command: mouse out
Target: css=a:nth-child(4) .panel-body
Value:
Description:

```
Running 'Make_Post'                                                      17:43:37
1. open on http://localhost:3000/dashboard/discussion OK                 17:43:39
2. setWindowSize on 1536x824 OK                                          17:43:39
3. click on css=.post-editor-subject OK                                  17:43:39
4. type on css=.post-editor-subject with value Test 1 OK                 17:43:41
5. click on css=.post-editor-body OK                                     17:43:42
6. type on css=.post-editor-body with value Heelllloooo OK               17:43:43
7. click on css=.btn OK                                                  17:43:44
8. mouseOver on css=a:nth-child(4) .panel-body OK                        17:43:45
9. mouseOut on css=a:nth-child(4) .panel-body OK                         17:43:46
'Make_Post' completed successfully                                       17:43:47
```

TC 16 - Reply to a post


Project: SENG401-FinalProjectTests*

Tests

- ✗ Add_PlantMax
- ✓ Add_PlantSuccess
- ✗ Add_PlantTooPoor
- ✓ Go_Discussion
- ✓ Go_Garden
- ✓ Go_NewsAboutFromDisc
- ✗ Login_Invalid
- ✓ Logout
- ✓ Make_Post
- Register_Existing
- ✓ Register_New
- ✗ Remove_Plant*
- ✓ Reply_Discussion
- Successful_Login

http://localhost:3000/dashboard/garden

	Command	Target	Value
1	✓ open	http://localhost:3000/dashboard/discussion	
2	✓ set window size	1552x840	
3	✓ click	css=a:nth-child(2) .panel-body	
4	✓ click	css=.form-control	
5	✓ type	css=.form-control	Hey! How's it going??
6	✓ click	css=.btn	

```
Running 'Reply_Discussion'                                                        18:48:05
1. open on http://localhost:3000/dashboard/discussion OK                          18:48:07
2. setWindowSize on 1552x840 OK                                                   18:48:07
3. click on css=a:nth-child(2) .panel-body OK                                     18:48:07
4. click on css=.form-control OK                                                  18:48:10
5. type on css=.form-control with value Hey! How's it going?? OK                  18:48:11
6. click on css=.btn OK                                                           18:48:12
'Reply_Discussion' completed successfully                                         18:48:13
```

Inputs and outputs of the test cases are summarized in the table above. For further inspection of each test case, please view the test files, written in JavaScript Mocha, in the “tests” folder in the GitHub repository.