Madeline Febinger
CSC 415
10/26/18
Assignment 4 - OSS Analysis and Design
Path to project on VM (csc415-server09): /home/student1/seniorsafeguard
Github repo: https://github.com/madelinefebinger/senior-safeguard

Use Case Descriptions

Use Case: Log-in
Primary Actor: User
Goal in context: To log in to the website
Preconditions: The user has a Senior Safeguard account
Trigger: The user is on the login page
Scenario:

1. The system displays a form to enter an email and password.
2. The user enters their email and password.
3. The user clicks on “Log in”
4. The system validates the email and password.
5. The system displays the home page and the user is logged in.

Exceptions:

1. If the user enters an email that does not have an account, the system will display a link to create an account, as well as display the login page.
2. If the user enters a valid email but an incorrect password, the system will display an “Incorrect password” message and show the login page again.

Priority: High
Open Issues: How should account recovery be implemented? (Ex. If a user loses their password)

Use Case: Report a scam
Primary Actor: User
Goal in context: To report a scam for other users to review
Preconditions: The user is logged in, The user is suspicious of a scam
Trigger: The user clicks on “Report a scam”
Scenario:

1. The system displays a form to enter information about the scam (Ex. type of scam, a description, an optional photo upload)
2. The user fills out the form.
3. The user clicks on “Submit”
4. The system adds the report to the database of open requests for other users to view.
5. The system sends an email notification to everyone on the user’s friend list.
6. The system shows a message that the report has been successfully saved.
7. The system returns to the home page.

Exceptions: If the user does not fill out the whole form, show an error message with what still needs to be filled out.
Priority: High

Use Case: Review a scam
Primary Actor: User
Goal in context: To submit feedback for a scam that another user has reported
Preconditions: There is an open request for the user to answer
Trigger: The user clicks on “Review” next to the request they want to answer.
Scenario:

1. The system displays a form to enter feedback about the scam.
2. The user types in their opinion of the scam.
3. The user clicks on “Submit”.
4. The system displays a message that the feedback has been successfully saved.
5. The system returns to the home page.

Exceptions: If the user does not fill out the whole form, show an error message with what still needs to be filled out.
Priority: High

Use Case: Add a friend
Primary Actor: User
Goal in context: To add a friend
Preconditions: The user is on the friend list page
Trigger: The user clicks on “Add friend”
Scenario:

1. The system displays a form for the user to enter the email of the friend they want to add.
2. The user types in the email of the friend they want to add.
3. The user clicks on “Submit”.
4. The system displays a message that the friend request went through.
5. The system returns to the friend list page.

Exceptions:

1. If the user enters an email that does not have an account, the system displays an error message.

Priority: Moderate

Use Case: View Educational Information
Primary Actor: User
Goal in context: To learn about scams/fraud involving technology
Preconditions: The user is logged in
Trigger: The user clicks on “Learn about online scams”
Scenario:

1. The system displays a page of educational information.

2. The system searches online resources for recent popular scams involving technology.
3. The system displays this information.
4. The user views the educational information.

Priority: Moderate

Open Issues: What websites should this information come from? Are there other similar resources to our system?

Design Class Diagram

Submitted as madelinefebinger_assign4_designclass.pdf

Statechart

Submitted as madelinefebinger_assign4_statechart.pdf

System Sequence Diagram(s)

Submitted as madelinefebinger_assign4_ssd.pdf

User Interface Mock-Ups

Submitted as madelinefebinger_assign4_mockups.pdf

Explain key principles of affordance and visibility

The user interface of Senior Safeguard will follow the principle of affordance, meaning that the appearance of any control should suggest its functionality. For the most part, I plan on putting text on buttons to describe what they do. To submit a “report a scam” form, the button will stand out from its surroundings and say “Submit” to clearly indicate its functionality.

Senior Safeguard will also follow the principle of visibility, meaning that all controls will be visible and provide immediate feedback to the user. The user interface will include a visible menu at the top of the screen with links to the pages for all of the functionality. Clicking on a link will provide the user with immediate feedback by bringing the user to a new page. After clicking a submit button, there will be a successful completion message shown.

Eight Golden Rules

1. Strive for consistency

To keep Senior Safeguard consistent, there will be a navigation menu at the top of each page. The “report a scam” and “review a scam” forms will look similar so that once users learn how to fill out one, they can easily learn how to fill out the other.

2. Enable Frequent Users to Use Shortcuts

I don’t have that many plans to use shortcuts, but one idea is for frequent users to use the enter key on the keyboard to submit forms instead of clicking on a submit button.

3. Offer Informative Feedback

One example of informative feedback would be showing a message when the user enters an incorrect password, prompting them to try again or click on “Forgot password”. When a user

tries to add a friend with an email that does not have an account, the user interface will show a message describing this problem and recommending that the friend makes their own account.

4. Design Dialogs to Yield Closure

After a scam report or scam review is submitted, there will be a message shown that it was completed successfully. After a friend is added, there will be a message shown that the friend was successfully added.

5. Offer Simple Error Handling

There will be error handling for the login page. If a user enters an incorrect password, they won't have to type in their email again to attempt another login. There will be a "Forgot password" button to handle the case when the user cannot log in to their account. The adding a friend functionality will also have error handling. If a user types in an email that does not have a Senior Safeguard account, the friend will not be added.

6. Permit Easy Reversal of Actions

There will be options to delete/edit a scam request or review. Users can add and delete friends as often as they want.

7. Support Internal Locus of Control

Since this application is intended to be used by the elderly, it is important that they always feel in control because technology can easily intimidate older people. The error messages will be non-threatening. There will be detailed instructions how to submit a scam report, and an FAQ page that the user can return to whenever they feel confused. The easy reversal of actions such as deleting requests and friends will also help the user feel in control.

8. Reduce Short-Term Memory Load

The user interface of Senior Safeguard will be fairly simple to reduce the memory load for senior citizens who are already hesitant to use technology. A navigation menu at the top of each page will make it easy for users to find the functionality they're looking for. The email notifications also remind users to check the application, so they don't need to remember to constantly check it.

Design Requirements

Modularity and encapsulation to facilitate information hiding and reuse

To ensure modularity for Senior Safeguard, I plan to break the functionality into modules. Some examples of these modules will be adding a friend, reporting a scam, and viewing educational information. These modules have high cohesion because they group related things together, specifically through object oriented classes. For example, I will have a Report class that groups together the information about the type of scam, description, and date submitted.

This Report class will encapsulate all of the information when a user reports a scam. This class will use information hiding because all of the class variables will be private. These private variables are important because it adds to the security of the application. The use of object oriented classes will also contribute to reuse of code through class methods. For example, the report class could have a method that sends an email notification whenever it's created, and this code would be ran often.

Elegance and efficiency of the algorithms

One of the algorithms implemented will be for creating and resolving reports. The form, completed by the user, will be encapsulated in a Report class. The Report class will send an email notification to all of the user's friends. The Report class will have a subclass called Review, which will encapsulate the feedback about the reported scam. By using these classes to encapsulate the information, the algorithm with be elegant and easy to understand for others who want to learn more about this open source project.

Appropriateness of the data structures used for the problem

Most of the data structures to be used will be object oriented classes such as the User, ScamReport, and ScamReview classes. These classes are appropriate because they abstract the information about the scams. Using object oriented classes, the code is also more readable, which is beneficial because this is an open source project. Also, the PostgreSQL database will contain all of the user account information such as emails and password. This database is appropriate because it can securely contain the information for a large number of user accounts.

Test Case Design

Specify the approaches you will use for each of unit, integration, and system testing.

Unit testing will involve testing the small components, known as modules, in the system. My approach for unit testing will be to test each module right after I finish implementing it. For example, one of the first modules I will implement will be the "report a scam" module. The unit test will test this module on its own. An example of a unit test would be to check if the form completed by the user is successfully created into a ScamReport class.

I will also utilize integration testing to make sure that modules successfully work with each other. My approach will be to do integration testing after I finish implementing a module. For example, two modules that interact will be the "report a scam" module and the "friend" module. After these modules are implemented, I will run an integration test for the functionality that once a new ScamReport class is created, the user's friends are sent an email notification.

I will complete system testing once I am done completing a majority of the functionality. For example, I will use the system to create a new account, add friends, report a scam, review a scam, and view feedback. If everything works as intended, it will pass the system test.

Specify which debugging and testing tools you will use

I usually debug code by adding print statements to output variables or by commenting out lines of code to isolate the problem. Since this is the first time I'm developing such a complex application, if I can't fix bugs on my own, I will use the gdb debugger. The gdb debugger can add breakpoints to the code and watch certain variables, so that could be helpful for finding bugs in my application. Another possible debugging tool I could use is pry, which is a gem for debugging Ruby code. Pry seems to be similar to gdb but specifically for Ruby code, so I might try out that debugging tool as well.

Test Case Design

Functionality Tested	Inputs	Expected Output	Actual Output
Log in	An email and password that has a Senior Safeguard account	The system logs in the user and shows the home page	
Log in error handling	An incorrect email and password combination	The system shows an error message and lets the user log in again	
Report a scam	A filled out form with scam information	The report is added to the list under “My Scam Reports” on the home page	
Report a scam email functionality	A filled out form with scam information	The user’s friends are sent an email notification	
Review a scam	A filled out scam review form	The user who reported the scam sees the review on their home page	
Review a scam email functionality	A filled out scam review form	The user who reported the scam receives an email notification that the scam has been reviewed	
Friend list	Clicking on “Friends” from the menu	The system displays all of the users friends and friend requests	
Add a friend	An email that has an account	The account associated with that email receives a friend request	
Add a friend error handling	An email that does not have an account	The system shows an error message that that email is not associated with an account	