**Milestone 4: Development Phase (Coding and Testing)**

Noah Funderburgh

Grand Canyon University

Course Number: CST-452-O500

Michael Landreth

6/30/2024

**Christian Crush**

**Implementation Plan**

I chose the Waterfall methodology for my project mostly due to the short timeline of the class, which spans only four weeks. The Waterfall approach is sequential and has structured phases. These phases allow for clear planning and execution within this short timeframe. Each phase must be completed before moving on to the next, reducing the complexity and potential for scope creep that can arise in iterative methodologies like Agile. This structured approach ensures that all project requirements are fully understood and documented upfront, enabling a focused and streamlined development process. Given the tight deadline, the predictability and clarity provided by the Waterfall model are crucial for meeting project goals efficiently and effectively.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Use Case or User Story** | **List of detailed development task** | **Estimate (hrs)** | **Actual (hrs)** | **Percent Complete** |
| User Database Table | Create backend so users can be inserted and read from | 1 | 1 | 100% |
| User Registration | Design Registration form, validate user information, and create unit tests for registration functionality | 2 | 2 | 100% |
| Password Requirements | Implement password strength requirements, min characters, uppercase, lowercase, special characters and unit tests for password validation logic | 2 | 2 | 66% |
| Hash Passwords | Implement ability to hash passwords and verify that the password is equal to the hash. | 2 | 2 | 100% |
| User Login | Design login form, validate user login information and create unit tests for login functionality. | 2 | 2 | 100% |
| Login Authorization | Verify user has logged in before accessing certain pages | 0.5 | 1 | 100% |
| Profile Database Table | Create backend logic so profiles can be inserted and read from | 2 | 2 | 100% |
| Profile Creation | Design profile creation form, validate user inputs and create unit tests for profile creation functionality. | 4 | 3 | 66% |
| Messages Database Table | Create backend logic so messages can be inserted and read from | 2 |  | 100% |
| Sending Messages | Design message form, validate user inputs and create unit tests for sending messages | 4 | 2 | 33% |
| Match Database Table | Create backend logic so matches can be inserted and read from | 1 | N/A | 0% |
| Swipe to find matches | Design explore page so users can match with others, and create unit tests for matching functionality | 4 | N/A | 0% |
| List of matches | Design matches page so users can look through there matches, looks at their conversations. | 4 | N/A | 0% |

**Percent of User Stories complete for this iteration: 20%**

**Percent of User Stories complete for entire project: 50%**

**Mapping of Functional Requirements**

|  |  |  |  |
| --- | --- | --- | --- |
| **Use Case or User Story** | **Architecture Plan** | **Code Module** | **Test Case** |
| User Database Table | Design schema for users table | UserDAO.cs | Verify table creation and field constraints |
| User Registration | Implement user registration process | UserDAO.cs RegisterController.cs | Test user registration with valid and invalid data |
| Password Requirements | Define password polices (length & complexity) | PasswordValidation.cs | Validate password against defined requirements |
| Hash Passwords | Use hashing function for password storage | PasswordHasher.cs | Ensure passwords are hashed and verified correctly |
| User Login | Implement login functionality with authentication | UserDAO.cs  LoginController.cs | Test login with correct and incorrect credentials |
| Login Authorization | Restrict access to authenticated users only | CustomAuthorization.cs | Test access to restricted pages for logged in users and non-login users |
| Profile Database Table | Design schema for profile table | ProfileDAO.cs | Verify table creation and field constraints |
| Profile Creation | Implement profile creation functionality | ProfileDAO.cs  ProfileController.cs | Test profile creation with valid and invalid data |
| Messages Database Table | Design schema for messages table | MessageDAO.cs | Verify table creation and field constraints |
| Sending Messages | Implement functionality to send messages | MessageDAO.cs  MessageController.cs | Test sending messages between users |
| Match Database Table | Design schema for matches table | MatchDAO.cs | Test swipe functionality and match creation |
| Swipe to find matches | Implement swipe functionality for matching users | MatchDAO.cs  MatchController.cs | Test swipe functionality and match creation |
| List of matches | Display a list of user matches | MatchDAO.cs  MatchController.cs | Verify display of matches and user information. |

**Source Code Listing**

**Controllers –**

**HomeController –**

The HomeController class in C# contains methods for returning views such as Index, Login, Privacy, and Error, with logging functionality provided by a logger instance.

**LoginController –**

The LoginController class in C# handles user login, session management, and logout functionality.

**MatchController –**

The MatchController class in C# handles when users match with each other.

**MessageController –**

The MessageController class in C# contains methods for displaying messages and sending messages with custom authorization.

**ProfileController –**

The ProfileController class in C# contains methods for handling profile creation and rendering views.

**RegisterController –**

The RegisterController class in C# contains action methods for handling user registration and returning corresponding views based on the registration process.

**DataServices –**

**MatchDAO –**

The MatchDAO class contains methods for inserting, deleting and retrieving matches from a database using MySQL in C#.

**MessageDAO –**

The MessageDAO class contains methods for inserting, deleting, and retrieving messages from a database using MySQL in C#.

**ProfileDAO –**

The ProfileDAO class contains a method to insert a profile into a database using parameterized SQL statements.

**UserDAO –**

The UserDAO class in C# contains methods for user authentication, registration, retrieval, and deletion in a database.

**Models –**

**MatchModel –**

The MatchModel class represents two users who matched. The properties of the class contain both user ids.

**MessageModel –**

The MessageModel class represents a message entity with properties for message details such as sender, receiver, content, and timestamp.

**ProfileModel –**

The ProfileModel class represents a user profile with properties such as ProfileId, UserId,

FullName, Bio, Images, Occupation, and Hobbies.

**UserModel –**

The UserModel class in C# represents a user entity with properties for personal information and methods for age validation.

**Utility –**

**CustomAuthorization –**

The CustomAuthorizationAttribute class in C# checks if the "email" session variable is set and redirects the user to the login page if it is not.

**PasswordHasher –**

The PasswordHasher class provides methods to hash and verify passwords using SHA256 with salt for added security.

**PasswordValidation –**

The PasswordValidation class in C# validates password strength based on length, uppercase letters, digits, and special characters.

**Test Plan and Test Cases**

|  |  |
| --- | --- |
| **Test Case Name:** | PasswordHasherTest |
| **Priority:** | High |
| **Module:** | PasswordHasher.cs |
| **Test Objective:** | To verify that the PasswordHasher correctly hashes password and can verify them against their hashed values. |

|  |  |
| --- | --- |
| **Test Case Name:** | CustomAuthorization |
| **Priority:** | Medium |
| **Module:** | CustomAuthorization.cs |
| **Test Objective:** | To verify that CustomAuthorization restricts users from accessing pages unless logged in. |

|  |  |
| --- | --- |
| **Test Case Name:** | RegistrationTest |
| **Priority:** | High |
| **Module:** | UserDAO.cs, RegisterController.cs, UserModel.cs |
| **Test Objective:** | To verify that the user registration functionality correctly registers new users with valid information and handles cases were registration fails due to invalid or duplicate data. |

|  |  |
| --- | --- |
| **Test Case Name:** | UserLoginTest |
| **Priority:** | High |
| **Module:** | UserDAO.cs, LoginController.cs, UserModel.cs |
| **Test Objective:** | To verify that the user login functionality correctly authenticates users with valid credentials and rejects invalid credentials. |

|  |  |
| --- | --- |
| **Test Case Name:** | MessageTest |
| **Priority:** | Low |
| **Module:** | MessageDAO.cs, MessageController, MessageModel.cs |
| **Test Objective:** | To verify that the message sending functionality correctly stores messages in the database and retrieves them as expected. |

|  |  |
| --- | --- |
| **Test Case Name:** | ProfileTest |
| **Priority:** | High |
| **Module:** | ProfileDAO.cs, ProfileController, ProfileModel.cs |
| **Test Objective:** | To verify that the profile creation and retrieval functionalities work correctly and that the profile data is correctly stored and retrieved from the database. |

|  |  |
| --- | --- |
| **Test Case Name:** | MatchTest |
| **Priority:** | High |
| **Module:** | MatchDAO.cs, MatchController, MatchModel.cs |
| **Test Objective:** | To Verify that the match functionality works correctly including the ability o swipe and find matches, and that match data is correctly stored and retrieved from the database. |

**Application Demonstration**

<https://www.loom.com/share/3ffb64aff4214fb6b16e616b38f31db1>