OCDD

Dakoda Meade

CST-452 Capstone Project Development Document

Grand Canyon University

Instructor: Professor Mohammad Shahadat Ullah

Revision: 4

Date: 8/24/2024

**ABSTRACT**

The mobile automotive detailing company web application project aims to create a comprehensive, user-friendly platform that streamlines various aspects of the business. The application will host an online appointment booking, service reviews(hardcoded for now), and administrative tasks. The primary objective for this project is to enhance the user experience by providing a convenient way to schedule appointments and access services while enabling the company to manage operations more efficiently. This project is being undertaken as a favor for a friend, adding a personal commitment to deliver a high-quality application in a short amount of time. However, there will be more features added after the deadline is met for this class to ensure all the features promised for this class are done on time.

The web application will be developed using ASP.NET for the backend and MySQL for the database, ensuring a robust and scalable system. Following the Model-View-Controller (MVC) architecture, separating application business logic from the front-end logic. The User entity, with attributes such as role to differentiate between admins and customers, ensures proper access control and functionality tailored to different user types. The database schema includes tables for Users, Roles, Services, Appointments, and Appointment Statuses, essential for capturing all necessary data. The project's success will be measured by its functionality, user-friendly UI, and meeting the deadline. By integrating a Christian worldview, the project emphasizes ethical practices and community service, aligning goals with a commitment to moral values.

|  |
| --- |
| History and Signoff Sheet |

**Change Record**

|  |  |  |
| --- | --- | --- |
| **Date** | **Author** | **Revision Notes** |
| 8/8/2024 | Dakoda Meade | Initial draft for review/discussion |
| 8/14/2024 | Dakoda Meade | Revision update |
| 8/23/2024 | Dakoda Meade | Revision update |

|  |
| --- |
| **Overall Instructor Feedback/Comments** |

|  |
| --- |
| **Overall Instructor Feedback/Comments** |

**Integrated Instructor Feedback into Project Documentation**

Yes  No

**TABLE OF CONTENTS**

Implementation Plan 4

Mapping of Functional Requirements 5

Source Code Listing 6

Test Plan and Test Cases 7

User Interface Design 8

**Implementation Plan**

The implementation plan provides a detailed outline of activities necessary to ensure that delivery of the planned user stories for the milestone assignment. This course does not specify which planning delivery methodology or tools you are required to use in your implementation plan. It is recommended you seek advice from your mentor or the instructor. The following are some suggestions you can follow for your implementation plan.

1. Use a formal methodology, such Agile Scrum or Kanban.
2. Planning tools could include but not limited to Microsoft Excel or Word template, boards on monday.com, boards and reports on notion.so, scrum burn down templates, and more.
3. At a minimum your implementation plan should include the follow elements:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Use Case or User Story | List of detailed Development Tasks | Estimate (hrs.) | Actual (hrs.) | Percent Complete |
| Registration | 1. Design the registration form 2. Implement form validation 3. Integrate registration with database 4. Display registration success and failure pages 5. Encrypt password for secure storage 6. Test the registration | 10 | 11 | 100% |
| Login | 1. Design the login form 2. Implement authentication logic 3. Integrate with database 4. Display login success and failure pages 5. Test the login flow | 7 | 7 | 100% |
| Profile Management | 1. Design the profile view page 2. Implement data retrieval and display for the user data 3. Implement client logic to edit the form for both the personal information and the password 4. Integrate each form to update the database 5. Test the profile updating for both forms | 12 | 14 | 100% |
| Appointment Booking | 1. Design the appointment booking page 2. Design to work for non-logged in users, with added functionality to register after booking(sends personal info from booking to the register page) 3. Implement service, date and time selection logic 4. Implement logic to update available time slots when a service and date is selected 5. Store booking information in the database 6. Test the scheduling workflow | 20 | 30 | 100% |
| My Appointments | 1. Design the my appointments page 2. Implement cards for the scheduled appointments 3. Implement a table for completed and cancelled appointments 4. Implement logic to view appointment details 5. Implement logic to cancel a scheduled appointment 6. Integrate the appointment data retrieval and cancelation with the database 7. Implement appointment filtering 8. Test appointments workflow | 15 | 18 | 100% |
| Admin Dashboard | 1. Design the UI for the Admin Dashboard 2. Implement CRUD operations for services with cards 3. Implement appointments table with filtering 4. Implement logic to view appointment details 5. Implement logic to complete appointments 6. Integrate logic to update the database 7. Test admin dashboard page | 18 | 22 | 100% |
| Reviews | 1. Design the reviews page 2. Hard code the review data 3. Test review page | 3 | 2 | 100% |
| Home | 1. Design the home page 2. Implement a scheduling button on the page 3. Test the home page | 3 | 3 | 100% |

Percent of User Stories complete for this iteration: 100%

Percent of User Stories complete for entire project: 100%

***NOTE: If necessary, you may add subsections to those listed in order to match the requirements in the assignment description. Do not remove any top-level sections (those that are listed in the Table of Contents). As required by your project, you may add additional top-level sections (please update the Table of Contents).***

**Mapping of Functional Requirements**

Ensure that all functional requirements have been satisfied by creating a mapping for each requirement to the technical Architecture Plan, and code module(s) or function(s) that satisfies the requirement, and the test cases used to test the requirement. This is referred to the Traceability and a template can be found in the Class Resources.

Example

|  |  |  |  |
| --- | --- | --- | --- |
| (Use Case/User Story) Monthly bill sent by email | (Architecture Plan)  Page or section where designed | (Code Module) Class and Method where implemented | (Test Case)  Test Case number from the Test Plan |

This will be attached as a separate file. REQUIREMENTS TRACEABILITY MATRIX.xlsx

**Source Code Listing**

Provide a listing of the source code for the project. Include a brief description of all classes and files. Code should follow industry best practices, formatting, and comment style appropriate for the programming language.

Controllers:

* AdminDashboardController.cs: Handles the functionality related to the admin dashboard, this includes displaying services, managing appointments, and applying filters to view specific appointment data.
  + Methods:
    - Index(): Displays the main page of the admin dashboard.
    - FilterAppointments(String status, DateTime startTime, DateTime endTime, string searchTerm): Filter appointments like based on the inputted data.
* AppointmentController.cs: Handles the viewing of appointment details and the functionality of completing and cancelling appointments.
  + Methods:
    - Index(int appointmentID, string source): Displays the appointment details and displays buttons depending on the source.
    - CompleteAppointment(int appointmentID): Changes status of appointment to complete and redirects to the Admin Dashboard page.
    - CancelAppointment(int appointmentID): Changes status of appointment to cancelled and redirects to the source page.
* AppoitnemtnsFilterController.cs: Controller for handing the filtering of appointments.
  + Methods:
    - Filter(string status, int appointmentId, DateTime? startDate, DateTime? endDate, string source): Filters appointments based on the inputs of the user.
* HomeController.cs: Manages the home page of the application. This is the entry point of the application.
  + Methods:
    - Index(): Displays the home page of the application.
* LoginController.cs: Handles user authentication through a login page that verifies user credentials allowing access to other parts of the application.
  + Methods:
    - Index(): Displays the login page of the application.
    - ProcessLogin(UserModel user): Processes the login by verifying the credentials and displaying a login success and failure page based on the results.
* MyAppointmentsController.cs: Displays the appointment data for the logged-in user, allowing them to view upcoming and past appointments, as well as apply filters to view specific appointments.
  + Methods:
    - Index(): Displays a list of the user's scheduled and past appointments.
    - FilterAppointments(String status, DateTime startTime, DateTime endTime, string searchTerm): Filter appointments based on the inputted data.
* ProfileController.cs: Manages user profile operations, allowing users to view and update their personal information, as well as change their password. As well as allowing the user to logout.
  + Methods:
    - Index(): Displays the user’s profile page.
    - UpdateProfile(UserModel user): Allows the user to update their personal information.
    - UpdatePassword(UserModel user): Allows the user to update their password.
    - Logout(): Logs the user out.
* RegisterController.cs: Handles user registration, allowing new users to create an account for the application.
  + Methods:
    - Index(): Displays the registration page.
    - RegisterFromAppointment(string name, string email, string phoneNumber, string address, int zipCode, string city, string state): Displays the register page when coming from after scheduling an appointment when not registered. It takes the personal information from the appointment and prefills the register page.
    - ProcessRegister(UserModel user): Handles the registration process if the email exists in the system, it displays a failure page. If the email does not exist, it adds the user to the database.
* ScheduleAppointmentController.cs: Manages the appointment scheduling process, allowing users to schedule appointments for the services offered by the company.
  + Methods:
    - Index(): Displays the appointment scheduling page.
    - Schedule(AppointmentModel appointment): Handles the scheduling of the appointment based on the users input. Saves to the database and displays a confirmation screen.
    - GetAvailableTimeSlots(DateTime date): Gets the available time slots based on the input date from the user.
* ServiceController.cs: Manages the CRUD operations related to the services offered by the company, allowing admins to create, edit, and delete services(deleted services are not removed from the database).
  + Methods:
    - Index(): Displays the service view.
    - EditService(int serviceID): Allows the admin to edit the service information, displaying the service’s current data.
    - SaveService(ServiceModel service): Handles the saving of the service to the database.
    - DeleteService(int serviceID): Handle the deletion of a service but changing of the IsDeleted status to 1.
* AdminAuthorizationAttribute.cs: Authorization filter for blocking non admin users from access parts of the application.
  + Methods:
    - OnAuthorization(AuthorizationFilterContext context): Checks if the user is logged in and an administrator, redirecting to the home page if not.
* CustomAuthorizationAttribute.cs: Authorization filter for blocking nonusers from access parts of the application.
  + Methods:
    - OnAuthorization(AuthorizationFilterContext context): Checks if the user is logged in, redirecting to the login page if not.
* ReviewsController.cs: This controller manages the reviews; in this case it is just displaying one view of hardcoded reviews and images.
  + Methods:
    - Index(): Displays the main review page.

Models:

* AdminDashboardModel.cs: Represents the data for the admin dashboard including the list of services and all appointments.
* AppointmentModel.cs: Represents an appointment, including details such as date, time, service type, and user information.
* ServiceModel.cs: Represents a service offered by the company, including service name, description, and price.
* UserModel.cs: Represents a user of the system, including personal information, login credentials, and role.
* TimeSlotModel: Represents a slot of time for an appointment.

Services:

* AppointmentDAO.cs/AppointmentService.cs: Services for managing logic related to appointments, including scheduling, filtering, and retrieving appointments.
  + Methods:
    - SaveAppointmentUser(AppointmentModel appointment): Handles saving an appointment to the database and returning the appointment id for users.
    - SaveAppointmentNonUser(AppointmentModel appointment): Handles saving an appointment to the database and returning the appointmentID for non-users.
    - GetAppointment(int appointmentID): Returns the data of an appointment based on the ID.
    - GetAllAppointments(): Returns the top 2000 appointments in order from newest to oldest.
    - GetAppointmentsByUserID(int userID): Returns the appointments based on the userID.
    - UpdateAppointmentsWithUserID(string email, int userID): Updates the database to match any appointments without a userid and the same email as the user to the user.
    - CancelAppointment(int appointmentID): Changes the status of the appointment to cancelled.
    - CompleteAppointment(int appointmentID): Changes the status of the appointment to completed.
    - GetAvailableTimeSlots(DateTime dateTime, TimeSpan serviceDuration): Returns a list of the available timeslots.
* SecurityDAO.cs /SecurityService.cs : Services for managing logic related to user authentication authorization.
  + Methods:
    - IsValid/FindByEmailAndPassword (UserModel user): Returns of the email and password match what is in the database.
    - UserExists/FindUserByEmail(UserModel user): Returns if the email already exists in the database.
    - AddUser(UserModel user): Adds the user to the database.
    - GetUserId/FindUserIDByEmail(UserModel user): Returns the based userID of the user.
    - GetUserById(int userId): Returns the user based on the userID.
    - UpdateProfile(UserModel user): Updates the user’s personal information in the database.
    - UpdatePassword(UserModel user): Updates the user’s password in the database.
    - HashPassword(string password): Hashes the user’s password for secure storage in the database.
* ServiceDAO.cs /ServiceService.cs: Services for managing CRUD operations related to services.
  + Methods:
    - GetServiceByID(int serviceID): Returns the service based on the ID.
    - GetServices(): Returns a list of the services that are no deleted.
    - SaveService(ServiceModel service): Saves/Updates a service in the database.
    - DeleteService(int serviceID): “Deletes” a service in the database.

**Test Plan and Test Cases**

Develop test cases for all modules. Test modules as part of development process and continually revise code as necessary. A Test Plan and Test Case template can be found in the Class Resources.

Example of Test Case Listing

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test Case Name:** | | | | | | |
| **Priority:** | | | | | | |
| **Module:** | | | | | | |
| **Test Objective:** | | | | | | |
|  |  |  |  |  |  |  |
| **Step** |  | **Test Name** | **Test Steps** | **Test Data** | **Expected Results** | **Test Pass/Fail** |
| **1** |  |  |  |  |  |  |
| **2** |  |  |  |  |  |  |
| **3** |  |  |  |  |  |  |

This will be a separate file. Test Case.xlsx

**Application Demonstration**

Provide a short screencast (5-10 minutes), per GCU policies used in previous classes, that demonstrates functionality that is working up to this point in the project.

# Screencast Links:

[**https://www.loom.com/share/ddac09e618c6492b8b92354c798abec6**](https://www.loom.com/share/ddac09e618c6492b8b92354c798abec6)

[**https://www.loom.com/share/cb86a4667836497589f9c939dda118b3**](https://www.loom.com/share/cb86a4667836497589f9c939dda118b3)