

ResumeFiller

Team #8

Colin Homan, Joshua Williams, Hamzeh Sabatini, Jamie Antoun

Software Design Specification & Project Delivery Report

Version: (1)

Date: (11/11/2024)

Table of Contents

1 Introduction.....	5
1.1 Goals and objectives.....	5
1.2 Statement of system scope.....	5
2 Architectural design.....	6
2.1 System Architecture.....	6
2.2 Design Rational.....	7
3 Key Functionality design.....	8
3.1 Resume Autofill.....	8
3.1.1 Resume Autofill Use Cases.....	8
3.1.2 Processing sequence for Resume Autofill.....	9
3.1.3 Structural Design for Resume Autofill.....	9
3.1.4 Key Activities.....	10
3.1.5 Software Interface to other components.....	10
3.2 Settings Manager.....	11
3.2.1 Settings Manager Use Cases.....	11
3.2.2 Processing sequence for Settings Manager.....	11
3.2.3 Structural Design for Settings Manager.....	11
3.2.4 Key Activities.....	12
3.2.5 Software Interface to other components.....	12
4 User interface design.....	12
4.1 Interface design rules.....	12
4.2 Description of the user interface.....	12
4.2.1 Main Menu Page.....	12
4.2.2 Forms Page.....	13
4.2.3 Autofill Form Page.....	14
4.2.4 Settings Page.....	14
4.2.5 Create a Form Page.....	15
4.2.6 Edit a Form Page.....	16
4.2.7 Delete a Form Page.....	17
4.2.8 View a Form Page.....	18
5 Restrictions, limitations, and constraints.....	19
6 Testing Issues (SLO #2.v).....	19

6.1 Types of tests.....	19
6.2 List of Test Cases.....	19
6.3 Test Coverage.....	21
7 Appendices.....	22
7.1 Packaging and installation issues.....	22
7.2 User Manual.....	23
7.3 Open Issues.....	24
7.4 Lessons Learned.....	24
7.4.1 Project Management & Task Allocations (SLO #2.i).....	24
7.4.2 Implementation (SLO #2.iv).....	24
7.4.3 Design Patterns.....	27
7.4.4 Team Communications.....	27
7.4.4 Technologies Practiced (SLO #7).....	27
7.4.5 Desirable Changes.....	28
7.4.6 Challenges Faced.....	28

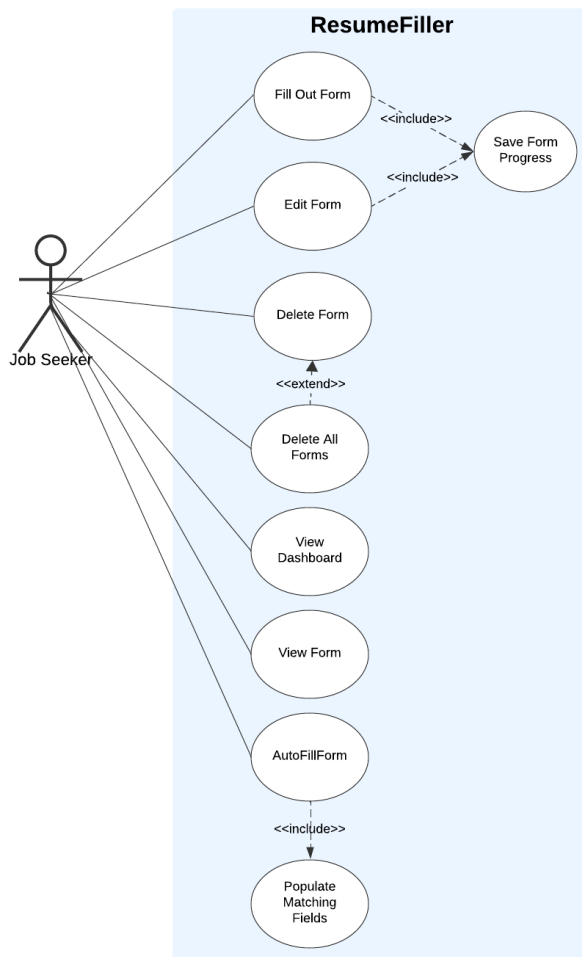
1 Introduction

1.1 Goals and objectives

Our goal is to allow a user to input resume information that will be stored in a database and recalled to fill out application forms on job application websites.

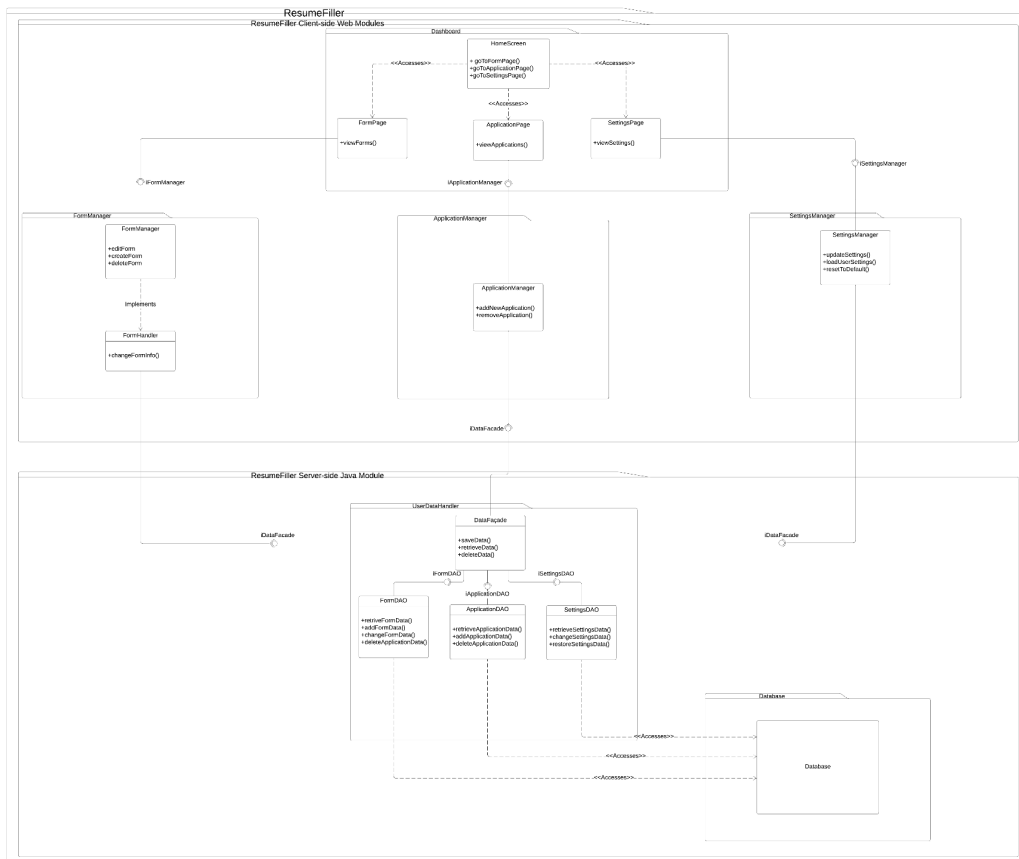
1.2 Statement of system scope

The software will allow users to create forms with resume information, edit the information within forms, and delete forms and have those additions and changes stored in the database. In the software, users can fill in information in forms on job application websites using pre entered information entered by the users in the forms saved on the database, as well as viewing and deleting applications from the application.



2 Architectural design

2.1 System Architecture



The system contains a main dashboard that allows the user to access the different functionalities of the system. We set aside the three main functions of the system into three different components, one for handling user form information, one for handling the autofill, and one for managing settings. This allows for one component to focus on each of the different services to provide to the user. The user opens the program and accesses the home screen, from where the user can access the form page, the autofill page, the settings page, or close the program. From the settings page, the settings page can access the interface for the settings manager, which connects to the database through a facade and data access object, allowing the user to make changes to the user's data. From the autofill page, the interface for the autofill manager can be accessed, which similarly through the data facade and a data access object accesses the database for the forms used in the autofill. The form page also uses the interface for the form manager, which implements the form handler, which allows users to create, modify, delete and view forms. The form handler through another interface accesses the data facade, who through another interface accesses the form data access object. Through this, changes, new additions, and deletion of form data is handled by the program.

2.2 Design Rational

We decided to use a client-server model for our application as the application is dealing with directly working with websites, as the ResumeFiller program will automatically fill in entries for job application websites.

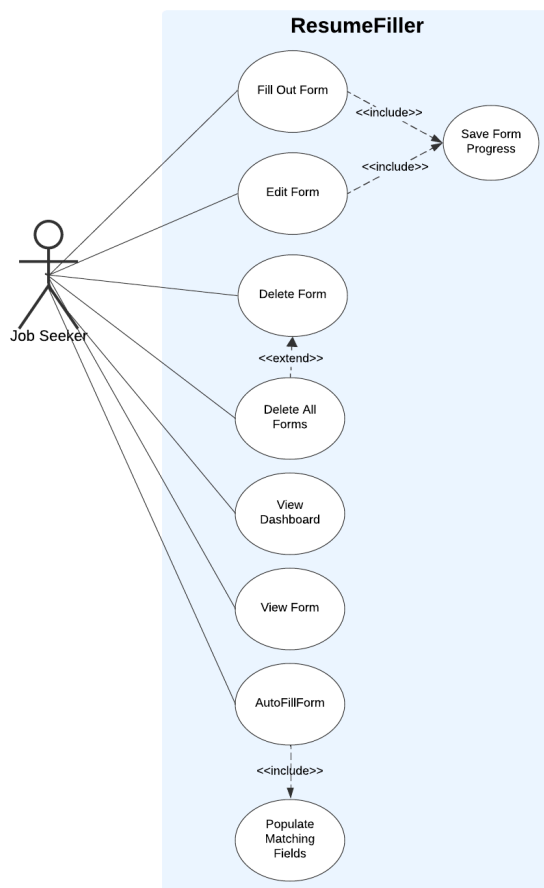
3 Key Functionality design

3.1 Resume Autofill

The ResumeFiller program focuses on two primary functionalities: resume autofill and user settings management. Each of these functionalities is designed to streamline the job application process, improve organization, and provide a customizable user experience.

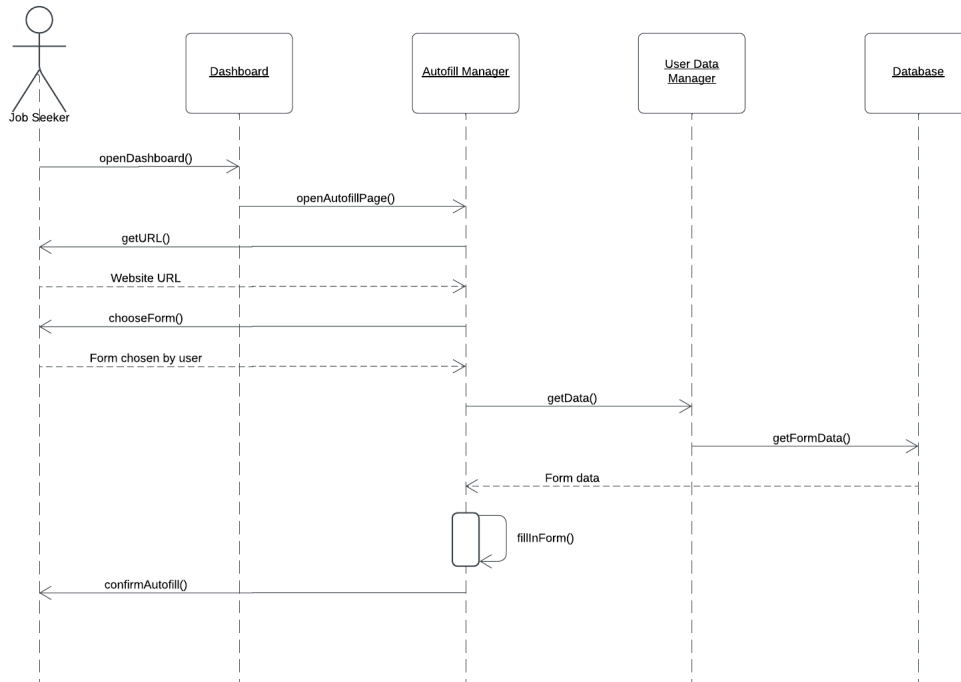
3.1.1 Resume Autofill Use Cases

The Resume Autofill feature allows users to automatically fill job application forms on various portals using data parsed from their uploaded resumes. Users upload their resumes, which are processed by the ResumeFiller extension to extract personal details, work history, and education. The extension uses this data to populate job application forms, streamlining the application process and reducing manual input. This feature is particularly helpful for job seekers who apply to multiple jobs, as it saves time and ensures consistency across applications.



3.1.2 Processing sequence for Resume Autofill

The Resume Autofill sequence begins when the user uploads their resume to the ResumeFiller program. Java-based parsing extracts key details from the resume, storing this information locally. When the user navigates to a supported job portal, the extension automatically fills in the form fields using the stored data. This data is organized into sections (e.g., personal information, work history) and mapped to corresponding fields on the job application. Users have the option to review and modify the filled information before submitting the application.

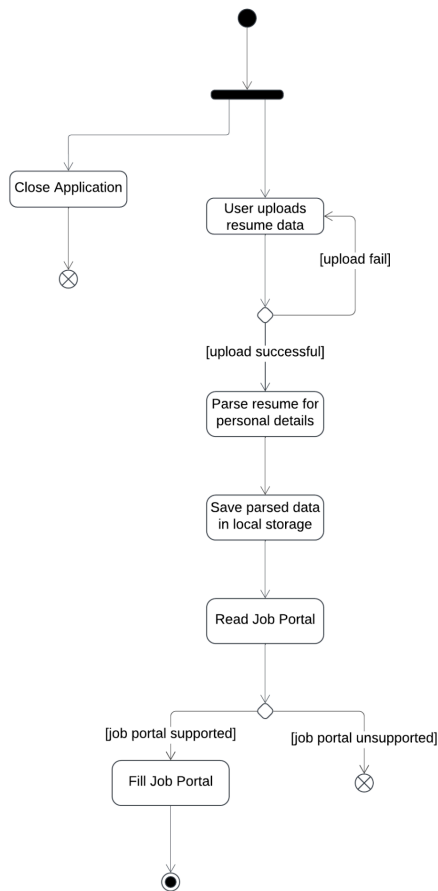


3.1.3 Structural Design for Resume Autofill

The structural design of Resume Autofill consists of several Java modules. The FormManager class handles the extraction and storage of resume data, while the ApplicationManager manages the autofill requests, mapping resume data to job portal form fields. The LocalStorageManager module manages the storage of user data locally within the browser, ensuring quick access to parsed resume data during autofill.

3.1.4 Key Activities

AutoFill form activity diagram



Key activities for the Resume Autofill functionality include:

1. Uploading and parsing the resume file.
2. Storing parsed data locally for use during form filling.
3. Mapping data fields to corresponding form fields on job portals.
4. Allowing users to review and adjust data before final submission.

3.1.5 Software Interface to other components

The Resume Autofill interface connects with local storage for retrieving and storing resume data, which is managed through Java-based modules. It integrates with the ResumeFiller program's Dashboard to display parsed data for user review and allows seamless transition to job application filling when users access supported job portals.

3.2 Settings Manager

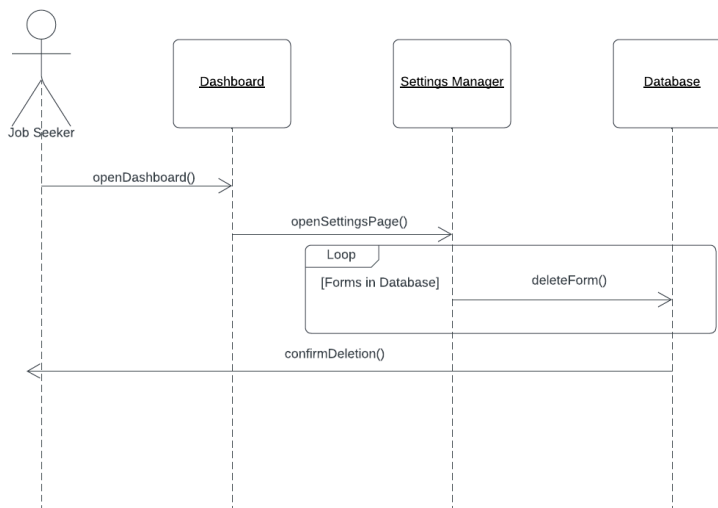
3.2.1 Settings Manager Use Cases

The User Settings Management feature enables users to manage the user data that they have already entered into the system. The settings manager allows the user to delete all information entered into the system.

3.2.2 Processing sequence for Settings Manager

When a user accesses the settings page, they can choose to clear all of the information they entered into the system. All of the information for every form will be deleted, leaving zero forms left in the system database.

Delete All Forms Sequence Diagram

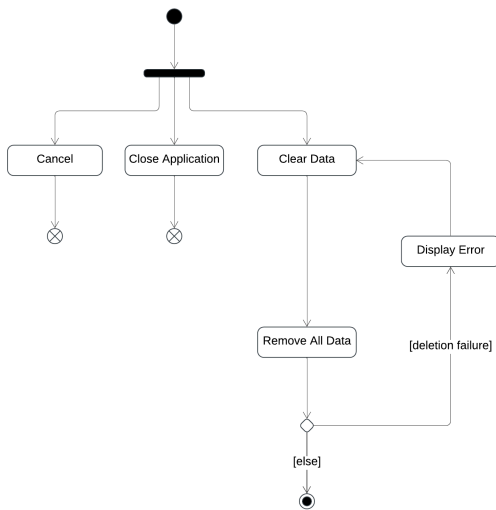


3.2.3 Structural Design for Settings Manager

The structural design for User Settings Management consists of Java classes like `SettingsPage` for managing the settings interface and `FormHandler` for handling form-related settings. The `LocalStorageManager` saves user preferences, ensuring they are applied consistently across all functionalities.

3.2.4 Key Activities

Clear All Data Activity Diagram



Key activities in the User Settings Management functionality include:

1. Clearing all user data.
2. Managing user data.

3.2.5 Software Interface to other components

User Settings Management interacts with LocalStorageManager to save and retrieve user preferences. These influence the database of the system as the user can manage the information that is stored within.

4 User interface design

4.1 Interface design rules

Our interface design only uses the colors blue, black, and white throughout the interface, with a uniform font and square shaped buttons throughout.

4.2 Description of the user interface

Our user interface is a software based application that provides a front-end for users to manage resume information and autofill the resume information of their choice into a job form.

4.2.1 Main Menu Page

The main menu page allows users to navigate to the forms page, the autofill page, the settings page, or to quit the program.

4.2.1.1 Screen Images



4.2.1.2 Objects and Actions

All pages contain a header that states the name of the program as well as the name of the page that they are currently on. All pages are navigated between using buttons titled after the pages they navigate to.

4.2.2 Forms Page

The main forms page allows the user to select the option they want to perform in relation to forms, either creating a new form, editing a pre-existing form, deleting a form, or viewing forms.

4.2.2.1 Screen Images



4.2.2.2 Objects and Actions

If the user clicks on the “Create a Resume” button, then the user will initiate the creation of a new form. If a user clicks on the “Edit a Resume” button, then the user will initiate the process of editing a pre-existing form. If a user clicks on the “Delete a Resume” button, then the user will initiate the process of deleting a form. If the user clicks on the “View Resumes” button, then the user will be navigated to the page where they can view resumes. If the user clicks on the “Return to Main Menu” button, then the user will return to the main menu page.

4.2.3 Autofill Form Page

The autofill form page allows the user to enter the url for the job website, choose the form they want to use in the autofill, then autofill the job website using the information from the form selected by the user.

4.2.3.1 Screen Images



The screenshot shows a window titled "ResumeFiller" with a blue header bar containing the text "ResumeFiller" and "Autofill Form". Below the header, there is a text input field labeled "Job Site URL:". Below the input field, there is a dropdown menu labeled "Choose the resume to autofill:" with "Jamies Resume" selected. At the bottom of the window, there are two buttons: "Cancel" and "Autofill".

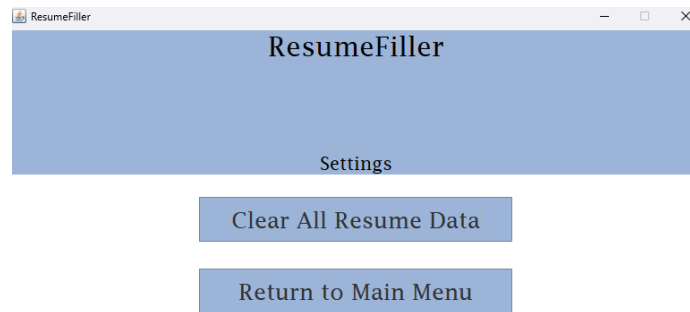
4.2.3.2 Objects and Actions

Here the user can enter a url into the input text field. From the dropdown menu, the user can select the form they want to use in the autofill of the application. By clicking the autofill button, the program will automatically fill in the appropriate information into the job website link input by the user. Clicking the cancel button will return the user to the main menu.

4.2.4 Settings Page

Allows the user to alter some settings with the software.

4.2.4.1 Screen Images



4.2.4.2 Objects and Actions

By clicking the “Clear All Data” button, the user can delete all of their form and application data from the database. By clicking the “Return to Main Menu” button, the user will return to the main menu.

4.2.5 Create a Form Page

This page allows users to enter information pertaining to a resume and allows them to save the inputted form data into the database.

4.2.5.1 Screen Images

The screenshot shows a window titled "ResumeFiller" with a blue header bar containing the text "ResumeFiller". Below the header, the text "Edit Resume" is centered. The form consists of several input fields with labels to their left: "Form to Edit:", "Full Name:", "Email:", "Phone Number:", "Address:", "Experience:", and "Qualifications:". At the bottom of the form, there are two buttons: "Cancel" and "Save".

4.2.5.2 Objects and Actions

Here, the user can input text into each of the input fields. Each input field requires that information be input into it before the user can save the information. Once the user populates the form with the appropriate information, then the user can click the save button to store the form

information in the database. The user can cancel the form creation at any time by clicking the cancel button, which will redirect them back to the forms page.

4.2.6 Edit a Form Page

This page allows the user to choose the pre-existing form of their choice, edit the information within, and save those changes to the database.

4.2.6.1 Screen Images

The screenshot shows the 'ResumeFiller' application window. The main title is 'ResumeFiller'. Below it is a section titled 'Your Resumes'. A 'Resume Selection' dialog box is open, prompting the user to 'Select a resume:' with a dropdown menu showing 'Jamies Resume'. Below the dialog box are four buttons: 'Delete a Resume', 'View Resumes', and 'Return to Main Menu'.

The screenshot shows the 'ResumeFiller' application window. The main title is 'ResumeFiller'. Below it is a section titled 'Edit Form'. The form contains the following fields: 'Form to Edit:' (a dropdown menu), 'Form Name:', 'Full Name:', 'Email:', 'Phone Number:', 'Address:', 'Experience:', and 'Qualifications:'. At the bottom of the form are two buttons: 'Cancel' and 'Save'.

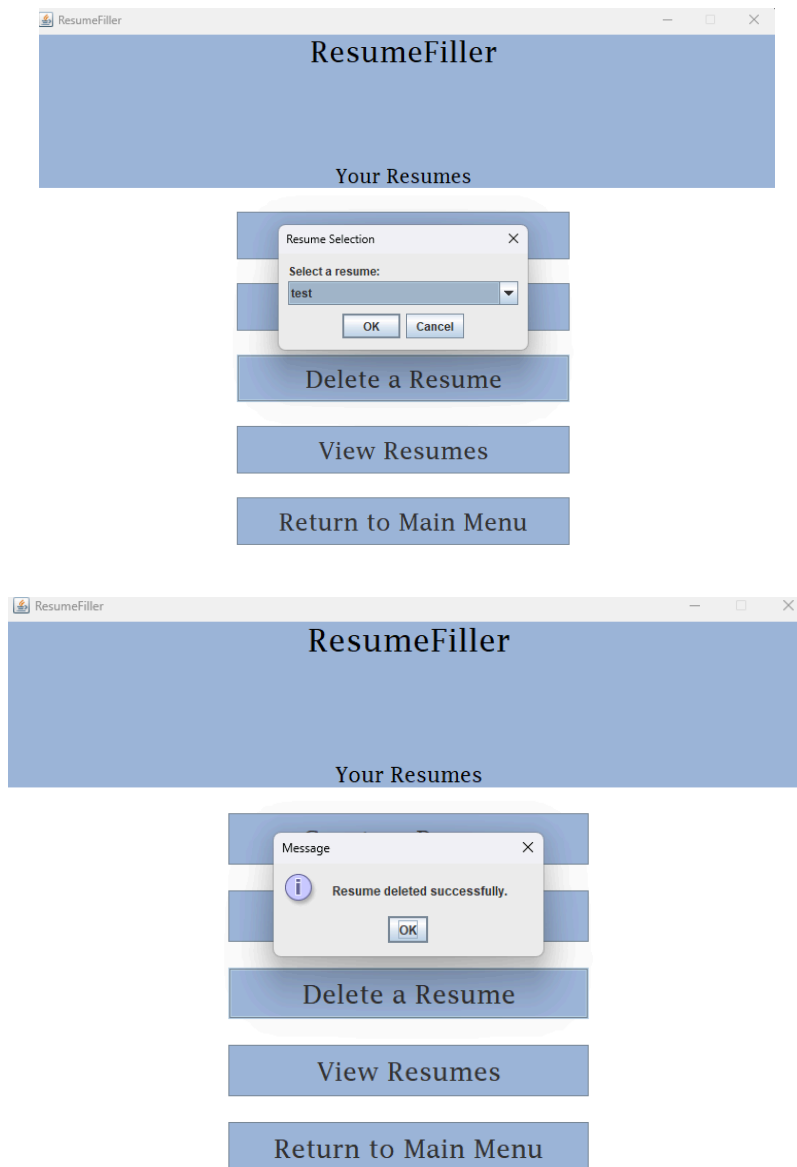
4.2.6.2 Objects and Actions

Here the user can select from a list of pre-existing forms from the drop down popup. Upon selecting a form, that form's information will then be populated into the other input fields. From there, the user can make adjustments to the data fields. The user can then save these changes by clicking the save button. The user can also cancel the edits by clicking the cancel button, which will redirect them to the forms page.

4.2.7 Delete a Form Page

This page allows the user to choose the pre-existing form of their choice and then delete it.

4.2.7.1 Screen Images



4.2.7.2 Objects and Actions

Here the user can select from a list of pre-existing forms from the drop down popup. By then clicking the ok button after making the selection, the user can delete the form they selected. The user can return to the forms page by clicking the cancel button.


4.2.8 View a Form Page

This page allows the user to choose the pre-existing form of their choice, and view the information contained within the chosen form.

4.2.8.1 Screen Images



The screenshot shows the 'ResumeFiller' application window. The main title is 'ResumeFiller' and the subtitle is 'Your Resumes'. A 'View Resumes' dialog box is open, prompting the user to 'Select a resume to view:'. The dropdown menu shows 'Jamies Resume' selected. There are 'OK' and 'Cancel' buttons in the dialog. Below the dialog, there are three buttons: 'Delete a Resume', 'View Resumes', and 'Return to Main Menu'.



The screenshot shows the 'ResumeFiller' application window. The main title is 'ResumeFiller' and the subtitle is 'View Resume'. The form contains the following fields:

- Full Name: test
- Email:
- Phone Number:
- Address:
- Experience:

At the bottom of the form, there is a button labeled 'Return to Form Page'.

4.2.8.2 Objects and Actions

Here the user can select from a list of pre-existing forms from the drop down popup. When the user selects a form, they can view the contents of the form they selected by clicking the ok button. The user can return to the form page by clicking on the “Return to Resume Page” button.

5 Restrictions, limitations, and constraints

- The application must work in the most recent version of the Google Chrome browser.
- Java language must be used for the application.

6 Testing Issues (SLO #2.v)

6.1 Types of tests

For the program we will conducted the following types of tests:

- (1). **Performance Test** – Ensuring that the autofill feature works quickly, as we stated it should fill in a form in 2 seconds or less.
- (2). **Accuracy Test** – Ensuring that when a user wants to view a form or edit a form, it always displays the correct information. Ensuring when a user deletes a form, it always deletes the correct form. When a user wants to autofill a form, it opens on the correct website and fills in the correct information.
- (3). **User Interface Test** – A user can easily navigate to the form pages to manage their forms. A user can easily navigate to the autofill page and autofill a job application website.
- (4). **Repeatability Test** - To ensure that a user can freely creates forms, edit them, delete them, and view them and that these functionalities will not stop working despite repeated use. Repeated actions will not break the program and will return the same information.

6.2 List of Test Cases

Test Type	Performance Test
Testing range	Autofill Feature
Testing Input	Valid URL and form
Testing procedure	Enter valid URL and select form Click “Autofill” button
Expected Test Result	Form autofills in less than 2 seconds
Testers	Jamie Antoun
Test result	Passed

Test Type	Accuracy Test
Testing range	View Form Feature

Testing Input	Valid Form
Testing procedure	User chooses a form from the dropdown menu
Expected Test Result	User interface displays the correct information
Testers	Joshua Williams
Test result	Passed

Test Type	Accuracy Test
Testing range	Edit Form Feature
Testing Input	Valid Form
Testing procedure	User chooses a form from the dropdown menu
Expected Test Result	User interface displays the correct information
Testers	Joshua Williams
Test result	Passed

Test Type	Accuracy Test
Testing range	Delete Form Feature
Testing Input	Valid Form
Testing procedure	User chooses a form from the dropdown menu
Expected Test Result	The correct form is deleted from the database
Testers	Joshua Williams
Test result	Passed

Test Type	Accuracy Test
Testing range	Autofill Feature
Testing Input	Valid URL and form
Testing procedure	Enter valid URL and select form Click "Autofill" button
Expected Test Result	Form autofills the website on the correct website with the correct information
Testers	Jamie Antoun
Test result	Passed

Test Type	User Interface Test
Testing range	Create Form Page
Testing Input	User Information
Testing procedure	Click on "Your Resumes" button Click on "Create A Resume Button" button Enter form information into text boxes Click "Save" button
Expected Test Result	User correctly navigates to the page to create a form, inputs the information, and saves it
Testers	Colin Homan
Test result	Passed

Test Type	User Interface Test
Testing range	Autofill Page
Testing Input	Valid URL and form
Testing procedure	Click "Autofill an Application" button Enter valid url and choose form Click "Autofill" button
Expected Test Result	User correctly navigates to the Autofill Form page and auto fills an application.
Testers	Colin Homan
Test result	Passed

Test Type	Repeatability Test
Testing range	Create Form, Delete Form, View Form
Testing Input	User Data
Testing procedure	User creates a new form, views it, creates another, views it again, deletes one of the forms, creates another, edits one, views again 5 times.
Expected Test Result	User interface displays the correct information, where there has been 2 net forms added (3 added, 1 deleted from the database).
Testers	Joshua Williams
Test result	Passed

6.3 Test Coverage

First test - Non-functional: The autofill was to fill out the data fields on the website within 2 seconds of the autofill button being clicked; this was a success.

Second test - Functional: The view form page was to show the correct information of the desired form when its selected; this was a success.

Third test - Functional: The edit form page was to allow the user to be able to choose a form of their choice and be shown the latest state of that form's data and then be able to edit it; this was a success.

Fourth test - Functional: The delete form page allows the user to select the form to be deleted and there should be a popup that informs the user that the resume had been deleted, also viewing the available forms, it should not be there; this was a success.

Fifth test - Functional: The autofill feature is to fill out the correct website with the correct information from the user's form; this was a success.

Sixth test - Functional: The create form page allows the user to create a new form to hold their data and save it, the User Interface should work and be easy to navigate; this was a success.

Seventh test - Functional: User navigates to the “Autofill An Application”, clicks button, and enters the URL of the desired website to autofill the information with, the User Interface should work and be easy to navigate; this was a success.

Eighth test - Functional: The correct data should be displayed regardless of the amount of times that a user clicks through different functionalities such as create form, edit form, view form, and delete form, repeated functionalities requests should not break the program and it should return correct data; this was a success.

7 Appendices

7.1 Packaging and installation issues

Form Filler Project - Project Installation Guide

1. Install Java

Install Java version 17 or higher.

Install Oracle OpenJDK 23.0.1 or newer.

2. Set Up ChromeDriver

Download ChromeDriver from <https://googlechromelabs.github.io/chrome-for-testing/>.

Update the file path for chromedriver.exe in FormFiller.java.

Example:

```
System.setProperty("webdriver.chrome.driver", "C:\\path\\to\\chromedriver.exe");
```

3. Configure Maven

Open the pom.xml file.

Update the Selenium version under <properties> to

```
<selenium.version>4.26.0</selenium.version>.
```

In your IDE, right-click the project, go to "Maven," and select "Reload Project."

4. Set Project Settings

Open the project settings in your IDE.

Set the language level to 17.

Set the Module SDK to Oracle OpenJDK 23.0.1 or newer.

5. Run the Code

Open FormFiller.java in your IDE.

Run the fillForm method to start the automation.

Troubleshooting

Incorrect ChromeDriver Path

If the program fails to launch Chrome, ensure the file path in System.setProperty matches the location of your chromedriver.exe.

Maven Issues

If dependencies are not resolved, make sure you have reloaded the Maven project after editing pom.xml.

Outdated JDK

If you encounter errors related to the Java version, ensure the JDK installed is at least Oracle OpenJDK 23.0.1 and matches the Module SDK in your IDE settings.

Selenium Version Mismatch

Verify that the Selenium version in pom.xml is compatible with your ChromeDriver version.

IDE Errors

Ensure the language level is set to 17 and that your IDE is configured correctly in the Project Structure settings.

7.2 User Manual

To begin using the system, launch the application by running the Main.java file. This will open the UI, which provides a main menu with various options. From here, users can manage their resumes or access the auto-fill feature.

To create a new resume, click on "Your Resumes" from the main menu and select "Create a Resume." This will open a form where you can enter details such as name, email, phone number, address, and work experience. Once all fields are completed, click "Save" to store the resume. A confirmation message will appear to notify you of the successful save.

If you need to edit an existing resume, navigate to "Your Resumes" and select "Edit a Resume." Choose the desired resume from the list, make the necessary updates to the fields, and click "Save" to overwrite the previous data. The system will confirm that the changes have been successfully applied.

For deleting a resume, select "Your Resumes" and then "Delete a Resume." A list of saved resumes will appear, allowing you to choose the one you wish to delete. Confirm the deletion, and the resume will be removed from the system. The application will notify you once the operation is complete.

To clear all saved resume data, navigate to "Your Resumes" in the main menu and select the "Clear All Data" option. This feature is designed to delete all resumes stored in the system, providing a fresh start. When selected, the system will prompt you to confirm the action to avoid accidental data loss. Upon confirmation, all resumes will be permanently removed from the local storage, and the system will display a success message to confirm that the data has been cleared. This feature is particularly useful for resetting the system when managing multiple users or starting a new batch of resumes.

Finally, to use the auto-fill feature, go back to the main menu and select "Auto-Fill Form." Enter the URL of the web form you want to populate and choose a resume from the dropdown menu. Click "Fill Form," and the system will automatically fill in the fields on the webpage using the selected resume's data. If any issues occur, such as an invalid URL or missing form fields, the system will provide error messages to guide you.

7.3 Open Issues

Although functional, the application was developed to work exclusively with a single website, which limits its adaptability and broader usability. It also lacks support for cloud-based resume storage, restricting its scalability. Error handling during the auto-fill process is not comprehensive; for example, missing or mismatched field names on the web form can cause the automation to fail. Furthermore, it does not have the capability to handle non-text input fields, such as dropdown menus or checkboxes, in the auto-fill feature.

7.4 Lessons Learned

7.4.1 Project Management & Task Allocations (SLO #2.i)

Our team allocated tasks based on individual skills and interests. One member focused on Selenium integration for browser automation, another handled the UI design using Java Swing, and a third worked on JSON data handling with Jackson. While this division of responsibilities was efficient, it led to delays when one member encountered challenges, particularly with learning Selenium. In the future, a more collaborative approach or shared knowledge of key components could improve efficiency.

Project planning started with a general timeline and a list of key features. However, the lack of detailed milestones made tracking progress difficult. While weekly meetings helped identify and address issues, a more structured approach with clear deliverables and deadlines would have been more effective.

We performed minimal risk analysis at the beginning of the project, which caused delays when we encountered unexpected technical issues, such as pivoting from a Chrome extension to a Java-based application. Incorporating risk assessment early in the planning phase would have helped mitigate these challenges.

Updates were shared during weekly meetings, where we discussed progress and adjusted plans as needed. While effective for general updates, more frequent check-ins or real-time communication tools could have improved coordination. Changes were tracked using Git and Discord, which allowed us to review and manage code modifications effectively. However, more frequent commits with detailed messages would have further streamlined the process.

7.4.2 Implementation (SLO #2.iv)

Our team conducted periodic code reviews to ensure consistency and quality. During these reviews, we focused on simplifying complex logic and improving readability. For example, we refactored redundant code in the ResumeManager class to streamline file handling operations. These efforts improved the maintainability of the codebase.

The implementation is closely aligned with the initial system design. The project was divided into three main components: the UI, database management, and Selenium automation. Each component functioned as intended, although some adjustments were made to accommodate technical limitations. For instance, we originally planned to use cloud storage for resumes but opted for local JSON files due to time constraints.

Overall, the implementation was consistent with the design, and code reviews helped ensure high quality. However, greater attention to risk management and iterative testing would have addressed issues earlier in the process.

```
public static ResumeData loadResumeData() throws IOException {
    File file = new File(FILE_PATH);
    if (!file.exists()) {
        return new ResumeData();
    }
    return objectMapper.readValue(file, ResumeData.class);
}

public static void saveResumeData(ResumeData data) throws IOException {
    objectMapper.writeValue(new File(FILE_PATH), data);
}
}
```

The refactored loadResumeData and saveResumeData methods in ResumeManager streamline JSON file handling by centralizing logic for loading and saving resume data. This improves maintainability, reduces redundancy, and ensures consistent error handling throughout the application.

```
public class FormFiller {  
    1 usage  
    public void fillForm(String url, ResumeData data) {  
        System.setProperty("webdriver.chrome.driver", "C:\\Users\\Jamie\\Downloads\\chromedriver-win64\\chromedriver-win64.exe");  
        WebDriver driver = new ChromeDriver();  
        driver.get(url);  
  
        try {  
            WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));  
            System.out.println("Attempting to fill Full Name...");  
            wait.until(ExpectedConditions.visibilityOfElementLocated(By.name("name")));  
            driver.findElement(By.name("name")).sendKeys(data.getName());  
            System.out.println("Full Name filled!");  
  
            System.out.println("Attempting to fill Email...");  
            wait.until(ExpectedConditions.visibilityOfElementLocated(By.name("email")));  
            driver.findElement(By.name("email")).sendKeys(data.getEmail());  
            System.out.println("Email filled!");  
  
            System.out.println("Attempting to fill Phone Number...");  
            wait.until(ExpectedConditions.visibilityOfElementLocated(By.name("phone")));  
            driver.findElement(By.name("phone")).sendKeys(data.getPhone());  
            System.out.println("Phone Number filled!");  
  
            System.out.println("Attempting to fill Address...");  
            wait.until(ExpectedConditions.visibilityOfElementLocated(By.name("address")));  
            driver.findElement(By.name("address")).sendKeys(data.getAddress());  
            System.out.println("Address filled!");  
        }  
    }  
}
```

```
        System.out.println("Attempting to fill Experience...");  
        wait.until(ExpectedConditions.visibilityOfElementLocated(By.name("experience")));  
        driver.findElement(By.name("experience")).sendKeys(data.getExperience());  
        System.out.println("Experience filled!");  
  
        System.out.println("Attempting to fill Qualifications...");  
        wait.until(ExpectedConditions.visibilityOfElementLocated(By.name("qualifications")));  
        driver.findElement(By.name("qualifications")).sendKeys(data.getQualifications());  
        System.out.println("Qualifications filled!");  
    } catch (Exception e) {  
        e.printStackTrace();  
    } finally {  
        driver.quit(); // close the browser after completing  
    }  
}
```

The fillForm method demonstrates Selenium's ability to interact with live web pages by dynamically locating form fields and populating them with user data. This implementation highlights the integration of automation into the project's core functionality, addressing real-world challenges in browser-based form filling.

```
private void switchCard(String cardName) {  
    System.out.println("Switching to card: " + cardName);  
    CardLayout cl = (CardLayout) cardPanel.getLayout();  
    cl.show(cardPanel, cardName);  
}
```

The switchCard method utilizes the CardLayout to dynamically switch between different pages in the graphical user interface. This approach ensures a seamless user experience by organizing the GUI into modular components, simplifying navigation and future expansions.

7.4.3 Design Patterns

The Singleton pattern was applied in ResumeManager to centralize database interactions. This ensured that all read and write operations were consistent across the application. Additionally, the Observer pattern was loosely implemented in the UI to reflect changes dynamically, though this could be improved for better responsiveness.

7.4.4 Team Communications

Team communication relied heavily on weekly meetings and online messaging through discord. While this sufficed for routine updates, urgent issues were hard to solve with everyone's schedule being different. In future projects, tools like shared project boards or daily stand-ups could mitigate these delays.

7.4.4 Technologies Practiced (SLO #7)

This project introduced Selenium, a completely new technology for browser automation. Learning to locate web elements using selectors and handle dynamic content was a steep learning curve. Additionally, the project honed skills in UI development with Java Swing, JSON manipulation using Jackson, and dependency management with Maven.

7.4.5 Desirable Changes

Assume that you have another month to work on the project, what aspects of the system you would like to improve? What are the additional features you want to add to the system? [Each student should use a separate paragraph to respond to the questions]

If given more time, the program would benefit from cloud-based storage, enabling users to access resumes across devices. Enhancing the auto-fill feature to handle complex form elements and improving error handling would also significantly improve usability. Another potential improvement is introducing users / accounts for people to sign in and have all of their information saved. Finally, I would like to make the application work for more websites.

If we had another month to work on the project, I would like to implement the application tracker feature that we considered early in development. We could also improve the program by including some accessibility features that we were originally planning on implementing but decided not to due to time constraints, like adding a high contrast mode.

If we had another month, I would like to implement a fully embedded sqlite3 database. This would allow for faster data access and saving. I would also like to add a more robust data field reading autofill feature that could properly understand even differently worded fields and fill them out correctly. A new form format would complement this new autofill and its capabilities better.

7.4.6 Challenges Faced

Among requirements specification, system design, and system implementation, which one you think is the hardest task? Why? [Each student should use a separate paragraph to respond to the questions]

The hardest aspect of the project was adapting the requirements after discovering that Chrome extensions required JavaScript instead of Java. Transitioning to a standalone program was not only a technical challenge but also required redefining the scope of the project. Learning Selenium was particularly difficult, as it required understanding web technologies that were unfamiliar to the team. Additionally, balancing simplicity with functionality in the UI posed design challenges.

For the project, the most challenging part for me was struggling with the layout of certain pages unintentionally messing up the layout of other pages. I found the Java Swing system to be much more complicated to work with compared to other built in GUI systems for other programming languages like Tkinter in Python. I had to learn a lot about the different parts of the Java Swing system in order to get the UI to not only be able to display multiple different pages, but to maintain consistency across all of the UI pages.

The hardest task for me was the system implementation as I have never worked in a group before on a project and not with so many different files accomplishing different tasks and using different technologies. Also, in the system implementation phase is when you find out through trial and error what you may have overlooked or oversimplified some aspect of the original specification or design. It also comes with the challenges of understanding these new

technologies and how to properly set them up with each other in an environment, getting all the correct drivers, updates, and file paths. I am very appreciate of this class and project though because it was the first time that I have been asked to create something of this size, and I know it's nothing compared to the "real world" but it's the closest I've ever been and that is of such great value to me.