A. Write a comprehensive summary that addresses the following requirements:

- **the business problem or opportunity you are solving for, including a description of the customers and why this application will fulfill their needs**

  The enhancements to the scheduling application aim to address inefficiencies and accessibility barriers faced by the consulting organization, which operates across multiple languages and time zones in Phoenix, White Plains, Montreal, and London. The existing system struggles with slow data retrieval and a lack of multilingual support, limiting its effectiveness for a global workforce. By introducing real-time search filtering and full localization for English, French, Spanish, and Russian, this project will create a solution that is fast, accessible, and tailored to the organization's diverse needs.

- **existing gaps in the software application you are replacing or modifying (if applicable)**

  The current application lacks the ability to dynamically filter large datasets in the TableView(s), which forces users to perform manual searches—an error-prone and time-consuming process. It also does not support localization other than the login form, making it difficult for non-English-speaking users to interact with the system. These gaps create operational inefficiencies and reduce overall productivity. The proposed enhancements solve these issues, ensuring quick access to customer records and a seamless multilingual experience.

- **the software development life cycle methodology you use to guide and support software development activities**

  Development will follow the Waterfall methodology, a structured approach that ensures each phase of the project is completed before moving to the next. This methodology aligns with the project's requirements for clear deliverables and predictable outcomes. Each phase—planning, design, implementation, testing, and deployment—will be executed sequentially to minimize risk and maintain focus on specific goals.

- **deliverables associated with the applied software development life cycle methodology**
  - A fully localized application with language settings that automatically adjust based on system locale.
  - Dynamic search functionality that filters TableView data in real time as the user types.
  - Documentation, including design diagrams, test reports, user guides, and developer notes.
  - An executable .jar file for easy deployment and distribution.

- **the plan for implementation of your software solution, including the anticipated outcomes from this development**

  The implementation will follow a structured plan, beginning with the design phase to outline how new features will integrate seamlessly with the existing JavaFX application and MySQL database. Development will proceed iteratively, starting with the dynamic search functionality, implemented using optimized algorithms to ensure minimal latency with large datasets. Localization will then be incorporated using resource bundles, enabling the application to dynamically adapt language settings. The testing phase will validate the functionality of these features under various conditions and platforms. The anticipated outcomes include a robust, user-friendly application that delivers efficient search capabilities and accurate localization, meeting both functional requirements and customer expectations.

- **the methods for validating and verifying that the developed software application meets the requirements and subsequently the needs of the customers**

The developed software application will be verified and validated through JUnit unit tests to ensure functionality like real-time filtering and localization accuracy. Functional testing will be conducted through manual step-through testing of the user interface, verifying scenarios such as accurate language adaptation based on system locale and correct filtering behavior in the TableView. Regression tests will confirm that new features integrate seamlessly with the existing system without introducing fresh issues. The process concludes with user acceptance testing to ensure the application meets professional standards and customer needs.

- **the programming environments and any related costs, as well as the human resources that are necessary to execute each task in the development of the software application**

Development will take place in a Windows 10 environment, using Java (OpenJDK 17), JavaFX, and MySQL. These tools are free and reliable, or already acquired (Windows 10), minimizing project costs. The project will be executed by a single individual, fulfilling both developer and tester roles, responsible for implementing features, writing documentation, validating functionality, and ensuring feedback is incorporated.

- **a projected timeline including milestones, start and end dates, duration for each milestone, dependencies, and resources assigned to each task**

The projected timeline is as follows:

Milestone	Start Date	End Date	Duration	Dependencies	Resources
Requirements Gathering	01/15/2025	01/16/2025	2 days	None	Developer
Design and Planning	01/17/2025	01/18/2025	2 days	Requirements	Developer
Feature Implementation	01/19/2025	01/25/2025	7 days	Design Completion	Developer
Testing and Validation	01/26/2025	01/28/2025	3 days	Implementation	Dev/Tester
Documentation and Review	01/29/2025	01/31/2025	3 days	Testing	Dev/Tester

By **January 31, 2025**, the project will deliver a robust, scalable scheduling application that improves data retrieval speed, enhances multilingual support, and expands accessibility to a wider audience. This solution will not only address current pain points like slow data retrieval and limited multilingual support but also set the foundation for future scalability and global collaboration.

The January 31 completion date *is well ahead* of the true cutoff point of March 31, 2025. This buffer period accounts for potential delays, ensuring that any unforeseen challenges during development or testing can be addressed without risking the project's overall deadline.

B. Design and develop a fully functional software application that addresses your identified business problem or organizational need. Include each of the following attributes as they are the minimum required elements for the application:

- **one of the following application types: mobile, web, or stand-alone application**

The application is a standalone Java-based system designed for desktop environments. It uses JavaFX for its graphical interface and runs locally without relying on web services.

- **code including inheritance, polymorphism, and encapsulation**

Inheritance:

The Main class extends JavaFX's Application, inheriting lifecycle methods such as start, which is used to set up the primary stage. Similarly, the FormLoader class demonstrates code reuse through method overloading, with one version of the openForm including a ResourceBundle parameter and the other lacking it.

Polymorphism:

Overridden methods like initialize in controllers provide specialized behavior for loading resources, setting up tables, and localizing user interfaces. Lambda expressions dynamically implement filtering logic at runtime, demonstrating polymorphism through functional interfaces like Predicate.

Encapsulation:

Key data members in classes like JDBC, Session, and various DAOs are private, with controlled access through public getters and setters. This design ensures data integrity and hides implementation details from external classes, maintaining a clean and secure architecture.

- **search functionality with multiple row results and displays**

  The application provides robust search functionality in ViewAppointments and ViewCustomers. Users can dynamically filter rows in a TableView by entering text into a search field. The search compares the input against multiple columns, such as appointment ID, title, description, location, type, start and end times, or customer details. Matching rows are displayed instantly, offering seamless refinement of results.

  To enhance filtering performance, data is initially fetched from the database and stored locally in an ObservableList. This minimizes database calls during filtering--all searches operate on the local dataset, ensuring efficient and responsive user interaction.

- **a database component with the functionality to securely add, modify, and delete the data**

  The application implements a secure database component through DAOs like AppointmentDAO and CustomerDAO. These DAOs handle adding, updating, and deleting records using prepared statements with parameterized queries, preventing SQL injection. This approach ensures secure and reliable modification of appointment and customer data while maintaining transactional integrity.

- **ability to generate reports with multiple columns, multiple rows, date-time stamp, and title**

  The application includes a report titled Contact Schedules, which fulfills the requirement for generating reports with multiple columns, multiple rows, a title, and date-time values. This report displays a table of appointments filtered by contact, including columns for appointment ID, title, description, type, start and end date-time, and customer ID. Users can dynamically select a contact to generate the relevant schedule.

- **exception controls**

  The application employs robust exception handling in DAOs and controllers. Try-catch blocks and throws clauses ensure that runtime errors, such as database connectivity issues or invalid user input, are gracefully managed. Alerts are used to communicate errors to the user without causing application crashes.

- **validation functionality**

  Input validation is implemented across forms, such as in AddCustomerForm and AddAppointmentForm. Fields are validated for completeness and logical consistency before processing. For example, appointments are checked for scheduling conflicts, and missing fields trigger user-facing alerts.

- **industry appropriate security features**

  The application secures user authentication with a LoginForm that verifies credentials against the database. Login attempts, both successful and unsuccessful, are logged to an external file for auditing purposes. SQL injection is mitigated through parameterized queries, and sensitive user data is managed securely in session storage.

- **design elements that make the application scalable**

  The DAO pattern and resource bundles support scalability by decoupling business logic from the UI and enabling seamless addition of new languages or features.. Localization is supported through resource bundles, allowing efficient expansion to additional languages. The use of JavaFX ensures compatibility across platforms without significant changes to the codebase.

- **a user-friendly, functional GUI**

  The application features an intuitive GUI built with JavaFX. It includes navigational buttons, organized forms, and clear labels. Dynamic alerts provide user feedback for actions like invalid input or operation success. Consistency in design across forms, especially button placement, ensures ease of use for end-users.

C. Create each of the following forms of documentation for the application you have developed:

- **a design document including a class diagram and design diagram**

  Please refer to the design_document.pdf file, located in the root of the project directory.

- **a test plan for a unit test, including screenshots**

  Test Case Name:

  Report Generation Across Locales

  Test Objective:

  To verify the locale-independent functionality of report generation by grouping appointments by month and type across multiple locales.

  Test Environment:

  Java Version: OpenJDK 17.0.2
  Framework: JUnit 5
  Database: MySQL (via JDBC connection)
  Application Modules Involved:
  ViewReports (Controller)
  AppointmentDAO (Database Access Object)
  Session (Locale Management)

  Test Setup:

  Database Connection: Established using JDBC.openConnection() in @BeforeAll setup.
  Locale Configuration: Tests are executed for the following locales:
  English (en)
  Spanish (es)
  French (fr)

Russian (ru)

Month Mapping Approach (Current Implementation):

report1MonthComboBox is manually populated with localized month names via loadedBundle.getString("january"), loadedBundle.getString("february"), ...

Database Query Execution:

AppointmentDAO.getMonthTypeCount(selectedMonth, selectedType) retrieves appointment counts.

Test Execution Steps:

Iterate through each locale (en, es, fr, ru) using Session.setCurrentLocale(new Locale(locale)).
Ensure report1MonthComboBox displays the correct localized month names.
Select a month and an appointment type, then execute onReport1GenerateButtonClick().
Verify that the displayed count matches expected values in the database.

Expected Results:

The correct localized month names should be displayed in report1MonthComboBox.
Query execution should return valid appointment counts.
No errors should occur due to locale switching.

Potential Failure Points:

Query Not Returning Results:

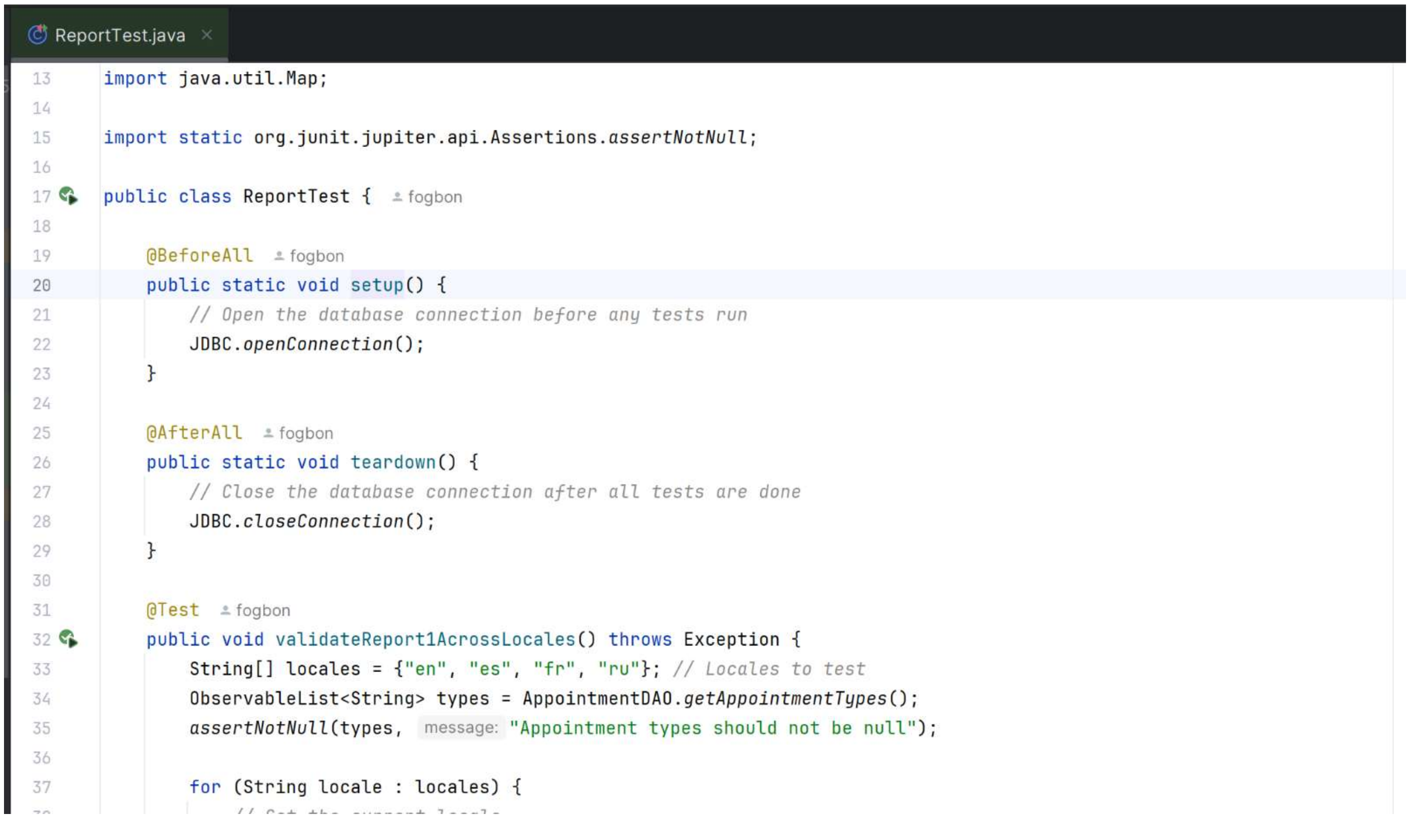
If the database expects month names in English only, but the UI provides localized names (e.g., "ENERO" instead of "JANUARY"), the query might return zero results or fail entirely.

Inconsistent Month Mapping Across Locales:

If the localization system does not guarantee that "January", "Enero", "Janvier", and "Январь" all map to the same month in SQL, counts might be incorrect or missing for certain locales.

Conclusion:

This test will determine whether the current localization method correctly maps month names to their numeric values in the database for each locale. If issues arise, we may need to refactor to ensure locale independence.



```java
import java.util.Map;

import static org.junit.jupiter.api.Assertions.assertNotNull;

public class ReportTest {  fogbon

    @BeforeAll  fogbon
    public static void setup() {
        // Open the database connection before any tests run
        JDBC.openConnection();
    }

    @AfterAll  fogbon
    public static void teardown() {
        // Close the database connection after all tests are done
        JDBC.closeConnection();
    }

    @Test  fogbon
    public void validateReport1AcrossLocales() throws Exception {
        String[] locales = {"en", "es", "fr", "ru"}; // Locales to test
        ObservableList<String> types = AppointmentDAO.getAppointmentTypes();
        assertNotNull(types,  message: "Appointment types should not be null");

        for (String locale : locales) {
```

- **the results of the unit test based on the provided test plan, including screenshots:**

  Test Purpose:

  To verify the locale-independent functionality of report generation by grouping appointments by month and type across multiple locales.

  Implementation Notes:

  To avoid rewriting a localized HashMap in the test class, the MonthMap helper class was introduced. This class dynamically loads month mappings based on the application's locale.
  The ViewReports controller was revised to leverage MonthMap for month localization, ensuring consistency between the application's UI and logic.
  The AppointmentDAO was also updated to use numeric month values (MONTH(start) in SQL queries), ensuring locale independence and accuracy.

  Test Execution:

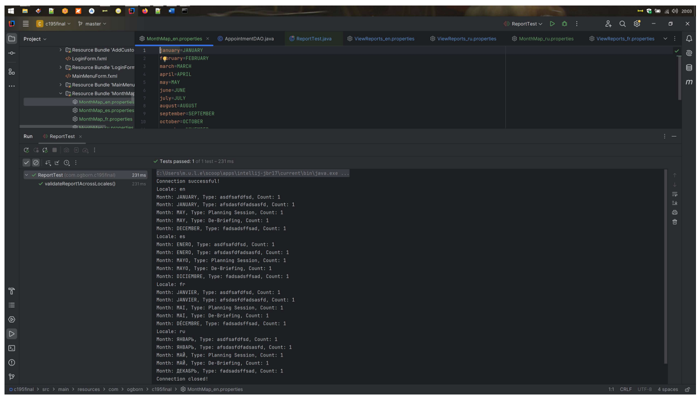
  The test executed successfully using four locales: English (en), Spanish (es), French (fr), and Russian (ru).
  The MonthMap dynamically loaded the localized month names.
  Appointment counts were retrieved and displayed only for months and types with non-zero values.

  Console Output:

```
Connection successful!
Locale: en
Month: JANUARY, Type: asdfsafdfsd, Count: 1
Month: JANUARY, Type: afsdasfdfadsasfd, Count: 1
Month: MAY, Type: Planning Session, Count: 1
Month: MAY, Type: De-Briefing, Count: 1
```

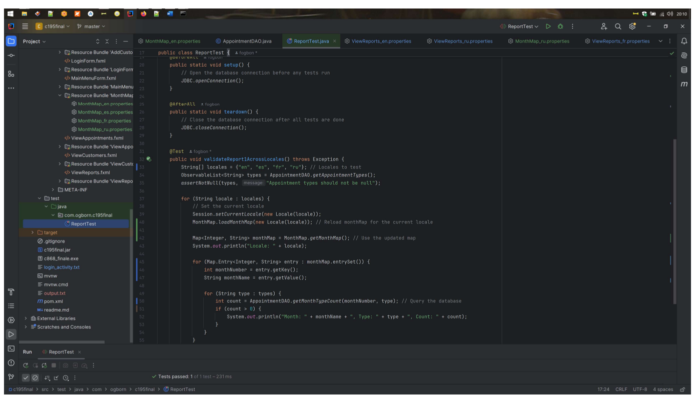
```
Month: DECEMBER, Type: fadsadsffsad, Count: 1
Locale: es
Month: ENERO, Type: asdfsafdfsd, Count: 1
Month: ENERO, Type: afsdasfdfadsasfd, Count: 1
Month: MAYO, Type: Planning Session, Count: 1
Month: MAYO, Type: De-Briefing, Count: 1
Month: DICIEMBRE, Type: fadsadsffsad, Count: 1
Locale: fr
Month: JANVIER, Type: asdfsafdfsd, Count: 1
Month: JANVIER, Type: afsdasfdfadsasfd, Count: 1
Month: MAI, Type: Planning Session, Count: 1
Month: MAI, Type: De-Briefing, Count: 1
Month: DÉCEMBRE, Type: fadsadsffsad, Count: 1
Locale: ru
Month: ЯНВАРЬ, Type: asdfsafdfsd, Count: 1
Month: ЯНВАРЬ, Type: afsdasfdfadsasfd, Count: 1
Month: МАЙ, Type: Planning Session, Count: 1
Month: МАЙ, Type: De-Briefing, Count: 1
Month: ДЕКАБРЬ, Type: fadsadsffsad, Count: 1
Connection closed!

Process finished with exit code 0
```

Screenshots:


MonthMap_en.properties
AppointmentDAO.java
ReportTest.java
ViewReports_en.properties
ViewReports_ru.properties
MonthMap_ru.properties
ViewReports_fr.properties
ReportTest
Tests passed: 1 of 1 test – 231 ms

```
january=JANUARY
february=FEBRUARY
march=MARCH
april=APRIL
may=MAY
june=JUNE
july=JULY
august=AUGUST
september=SEPTEMBER
october=OCTOBER
```

```
C:\Users\m.u.l.e\scoop\apps\intellij-jbr17\current\bin\java.exe ...
Connection successful!
Locale: en
Month: JANUARY, Type: asdfsafdfsd, Count: 1
Month: JANUARY, Type: afsdasfdfadsasfd, Count: 1
Month: MAY, Type: Planning Session, Count: 1
Month: MAY, Type: De-Briefing, Count: 1
Month: DECEMBER, Type: fadsadsffsad, Count: 1
Locale: es
Month: ENERO, Type: asdfsafdfsd, Count: 1
Month: ENERO, Type: afsdasfdfadsasfd, Count: 1
Month: MAYO, Type: Planning Session, Count: 1
Month: MAYO, Type: De-Briefing, Count: 1
Month: DICIEMBRE, Type: fadsadsffsad, Count: 1
Locale: fr
Month: JANVIER, Type: asdfsafdfsd, Count: 1
Month: JANVIER, Type: afsdasfdfadsasfd, Count: 1
Month: MAI, Type: Planning Session, Count: 1
Month: MAI, Type: De-Briefing, Count: 1
Month: DÉCEMBRE, Type: fadsadsffsad, Count: 1
Locale: ru
Month: ЯНВАРЬ, Type: asdfsafdfsd, Count: 1
Month: ЯНВАРЬ, Type: afsdasfdfadsasfd, Count: 1
Month: МАЙ, Type: Planning Session, Count: 1
Month: МАЙ, Type: De-Briefing, Count: 1
Month: ДЕКАБРЬ, Type: fadsadsffsad, Count: 1
Connection closed!
```


MonthMap_en.properties
AppointmentDAO.java
ReportTest.java
ViewReports_en.properties
ViewReports_ru.properties
MonthMap_ru.properties
ViewReports_fr.properties
ReportTest
Tests passed: 1 of 1 test – 231 ms

```java
public class ReportTest {
    public static void setup() {
        // Open the database connection before any tests run
        JDBC.openConnection();
    }

    @AfterAll
    public static void teardown() {
        // Close the database connection after all tests are done
        JDBC.closeConnection();
    }

    @Test
    public void validateReport1AcrossLocales() throws Exception {
        String[] locales = {"en", "es", "fr", "ru"}; // Locales to test
        ObservableList<String> types = AppointmentDAO.getAppointmentTypes();
        assertNotNull(types, message: "Appointment types should not be null");

        for (String locale : locales) {
            // Set the current locale
            Session.setCurrentLocale(new Locale(locale));
            MonthMap.loadMonthMap(new Locale(locale)); // Reload monthMap for the current locale

            Map<Integer, String> monthMap = MonthMap.getMonthMap(); // Use the updated map
            System.out.println("Locale: " + locale);

            for (Map.Entry<Integer, String> entry : monthMap.entrySet()) {
                int monthNumber = entry.getKey();
                String monthName = entry.getValue();

                for (String type : types) {
                    int count = AppointmentDAO.getMonthTypeCount(monthNumber, type); // Query the database
                    if (count > 0) {
                        System.out.println("Month: " + monthName + ", Type: " + type + ", Count: " + count);
                    }
                }
            }
```

Conclusion:

The test was successful in exposing poor implementation of month mapping in the ViewReports form, as the first test iteration required a HashMap within the test itself. The revised implementation, adding a MonthMap helper class, along with AppointmentDAO and ViewReports updates, successfully addresses locale independence and eliminates the need for repetitive localized mapping in the test code. The test plan was executed effectively, validating the application's functionality across multiple locales.

- **source code and executable file**

Source Code:

The source code resides in the /src/ folder within the project directory.

Executable files:

Choose one of the following executables, based on the current environment. The .exe file is included for convenience, since the .jar requires CLI usage to run.

c868final.jar: Located in the root of the project directory. If regenerated with intelliJ, the updated file will be placed in the /out/artifacts/c868final_jar/ folder. Can be launched from command line using **java -jar c868final.jar**

c868final.exe: Located in the root of the project directory. This .exe *is not signed by a certificate authority* and *therefore may cause a false positive* with an aggressive antivirus.

- **link to where web app is hosted with HTML code (if applicable)**
  Not applicable

- **user guide for setting up and running the application for maintenance purposes**
  Please refer to the README.md file, located in the project directory.

- **user guide for running the application from a user perspective**
  Please refer to the user_guide.pdf file, located in the project directory.