Requirements Specification

HoverSprite Organic Sprayer Management System

Introduction

HoverSprite is a start-up Vietnamese agricultural solutions company based in the Southwest Provinces. It specializes in organic fertilizer and pesticide spraying services for farmers, utilizing a fleet of 15 spraying drones to serve orchards, cereal fields, and vegetable farms. Currently, *HoverSprite* relies on a manual telephone-based system for order processing. Farmers place orders by calling the company's call center, where receptionists record the request and dispatch drone sprayers.

The sprayers then travel to the farm location, program the drones for spraying, and report completion via a video call chat app. Additionally, the receptionist gathers farmer feedback during the call. This process relies heavily on paper and spreadsheets for data management, limiting scalability and efficiency.

Recognizing the potential to expand market share and better serve farmers, *HoverSprite* seeks to digitize its services. Your software development team is tasked with building a web application to address these needs. The application will provide functionalities for:

- Customer registration and sign-in
- Spray booking and order management
- Service delivery
- Customer feedback and service quality rating

By implementing this web application, *HoverSprite* aims to **streamline their operations**, **enhance customer experience**, and **facilitate service expansion** in the future. Noticeably, the company wants the web application to provide service to **all farmers <u>regardless of their technical competency</u>**.

The requirements of the HoverSprite Application are divided into four levels:

- **Level 0**: Design the Data Model, Architecture, and User Experience of the entire system.

- Level 1: Develop the four services to a **minimally functional degree**

- Level 2: Provide the four services with **<u>enhanced</u> usability**, **efficiency**, **responsiveness** (to different devices) **maintainability**, and **security**

- Level 3: Provide the four services with **intuitive interactions**, **outstanding user experience** and **convenience**, **excellent responsiveness** (to smartphones and PCs), **well-protected security** measures, **highly maintainable** and **extensible architecture**, and enabling **registration and login** with **Facebook**, **Gmail,** or a **registered HoverSprite account**.

You are highly recommended to **analyze the service requirements of all Levels 1, 2, and 3** to decide the **<u>target</u> complexity**, **capability**, **flexibility**, and **goals** of your system architecture and application.

Level 0 Requirements

In this section, you are required to document the design decisions on the **Data Model**, **System Architecture**, and **User Experience** in a **25-page report**. The page count does NOT include your cover page and table of contents.

R01: Design Conceptual Data Model for Database

Develop an Entity Relationship Model (ER Model) to store information necessary for fulfilling the requirements. Include this ER model in your project report.

- **Conceptual Modeling:** Design a high-level representation of your database schema using appropriate concepts and entities relevant to your system's requirements.
- **Data Requirements:** Ensure the model accommodates all necessary data elements and relationships essential for system functionality.
- **Documentation:** Briefly explain the design ideas and purposes of entities and relationships depicted in your conceptual data model. **Justify the advantages and potential drawbacks of your design.**

R02: Present Architecture Diagram of the Entire System

Create an architecture diagram illustrating the entire system using either UML Component Diagram (Option 1) or C4 Model Container and Component Diagrams (Option 2).

Option 1: UML Component Diagram

Use UML Component Diagram to depict the architecture of your system.

Back-end:

- **Spring Boot Architecture:** Illustrate how Spring Boot components (Controllers, Services, Repositories, Models) are organized and interact to deliver services.
- **Utilities Classes and Spring Configuration:** Show helper classes and configurations used in your back-end (e.g., Spring Security, Token Generator, Cookie Configurer).

Front-end (if using React):

- **Component Breakdown:** Describe React components used in your web application.
- **Interaction with Back-end:** Detail how front-end components communicate with back-end services (e.g., REST API calls).

Front-end (if not using React):

- **Page Interaction:** Show webpages and their interactions with user inputs and back-end services.

Justification:

- **Architecture Design Rationale:** Provide a concise justification for your architectural decisions. **Justify the advantages and shortcomings of your design, and state why do you accept these trade-offs?**

- **Component Roles:** Describe the role and responsibilities of key components (back-end controllers, front-end components) within your system.

Note:

- You can draw the component diagram showing the back-end and front-end modules of the entire system **in one page** or depict the back-end and front-end architectures **in two diagrams**.

- In the latter option, only show the component on the other subsystem that directly interact with the currently focused subsystem. For example, if you are drawing the front-end architecture and express that the **User Authentication** Service in the front-end communicates with the **Authentication** module of the back-end, show only the AuthenticationController in the back-end subsystem as the recipient of the request.

Option 2: C4 Model Component Diagram

Use C4 Model Component and Container Diagrams to depict the architecture of your entire system.

Components Coverage:

- **Back-end and Front-end Components:** Represent components as described in Option 1.

- **Component Descriptions:** Include brief descriptions within each component regarding their purpose and functionality.

General Requirements for Both Options:

- **Clarity and Conciseness:** Ensure the diagrams are clear, concise, and effectively communicate the system architecture.

- **Alignment with Requirements:** Demonstrate how your architecture meets the specified system requirements and design principles.

- **Architectural Rationale:** Concisely justify the architectural choices made for each component**. Justify the advantages and shortcomings of your design, and state why do you accept these trade-offs?**

These tasks aim to evaluate your ability to design a robust data model and articulate a coherent system architecture, essential skills for software engineering projects.

R03: User Interface Design Aligns with 10 Nielsen Heuristics

This requirement ensures the user interface (UI) design of the system adheres to the 10 Nielsen Heuristics for UX Design. The heuristics provide established principles for creating user-friendly and intuitive interfaces. Following these principles will enhance the overall user experience of the system.

Deliverables:

- A comprehensive justification of how the UI design decisions across different pages of the system address the 10 Nielsen Heuristics.
- Each heuristic should be explained in the context of the system's UI design, with specific examples and justifications provided.
- Evidence can include screenshots of the relevant components or wireframes to illustrate how the design elements support the heuristics.

Additional Notes:

- Consider including a table that maps each Nielsen Heuristic to specific UI design elements or functionalities within the system. This can provide a clear overview of how the heuristics are addressed throughout the UI.
- The evaluation will assess the comprehensiveness of the analysis and the strength of the evidence provided to support the claims.

This requirement ensures your group focuses on creating a user-centered design that adheres to best practices in user experience design.

R04: Frequent Usage of GitHub

Your team must use GitHub as the only tool to store your deliverables, including **diagrams**, **reports**, **database**, and **code base**. You are required to use GitHub in a professional, iterative, and active approach by frequently pushing your primitive, partially completed, or finalized deliverables into the repositories, and use appropriate communication language in your repository.

Additional Note:

1) NOT Using GitHub frequently during the project **results in up to 10 points penalty** of your assessment grade**.**

2) In **your report, submit the proof of GitHub contribution** throughout the project duration

3) In **Canvas, submit the zip file of your GitHub repository.**

Missing either 2) or 3) results in two points penalty. Missing both 2) and 3) results in four points penalty.

Level 1 Requirements

General Requirements:

- The application interface should be intuitive and user-friendly for farmers with varying technical skills.

- The system should be accessible on desktops and usable in mobile devices.

RA1: Customer Registration and Sign-in

1) Account Registration

- The system shall allow **FARMERS** to register for an account using their **full name**, written in Vietnamese order, i.e., LastName MiddleName1 MiddleNameN FirstName, **phone number, email address**, and **home address**.

- Password creation and confirmation must be implemented. Password must contain at least **one capital letter** and **one special character**.

- **The database must store the hashed password, NOT the raw password.**

- A basic user profile section should be included to capture essential **FARMER** information (e.g., full name, phone number, home address).

- The system must store the **full name**, **phone number, address**, **expertise**, and **email** of **RECEPTIONISTS/SPRAYERS.**

  - The **domain part** of their email **must be hoversprite.com.** For example, nguyen.van.luong@hoversprite.com

  - **Expertise** must be one of the following values: **Apprentice**, **Adept**, **Expert**

2) Sign-in Protocol

- The system shall offer a sign-in functionality using the registered phone number/email address and password.

- The front-end must sends the username and password to the back-end **using the Basic Auth mechanism**.

RB1: Spray Order Booking and Management

1) Spray Order Booking

- The system shall provide a UI for **FARMERS/RECEPTIONISTS** to request for a spray order. **FARMERS** who have technical limitation directly phone the call center of *HoverSprite*, where **RECEPTIONISTS** receive phone calls and register the spray order.

- The system shall display available spraying sessions for **FARMERS/RECEPTIONISTS** to select. Every day of the week, *HoverSprite* offers **12 spraying sessions** at **six 1-hour time slots**. Each time slot has **two bookable spraying sessions**. There are <u>four morning time slots</u> from **04:00 to 08:00**, i.e., 04:00 – 05:00; 05:00 – 06:00; 06:00 – 07:00; 07:00 – 08:00. The two afternoon time slots are from **16:00 to 18:00**. **Fully booked time slots are disabled and cannot be selected**.

- **FARMERS/RECEPTIONISTS** shall be able to specify the following information:
  - **Type of crop**: Fruit, Cereal, or Vegetable
  - Farmland **area** (in decare**,** with 1 decare = 0.1 hectare = 1000m^2**)**
  - The desired **date** AND **time** for the service. Date format is **dd/mm/yyyy**, time format is **hh:mm**, with the hour value ranges from 00 to 24.
  - You must show **both Lunar Calendar AND Gregorian Calendar** dates, as both dates are important for agricultural activities in Vietnam.

- The system shall display the total cost with a unit price of **30,000 VND/decare**.

- **FARMERS/RECEPTIONISTS** shall be able to confirm and submit the service request. **Only cash payment is applied in Level 1**.

- The system sends an email to confirm the booking is successful and provide the information on the **date** (both Gregorian and Lunar calendar date), **time**, **location**, **farmland size** and **total cost** in the email. The email must also include an appreciation to the **FARMERS** for their trust and choice in using the *HoverSprite* service.

2) Spray Order Management

- A basic order management section shall be available for **FARMERS/RECEPTIONISTS** to view the status and details of their spray orders. A **FARMER** can only see order details made by themselves. **RECEPTIONISTS** can see order created by any **FARMER/RECEPTIONIST** in the system.

- When viewing the list of orders, **Completed** orders (See **RC1)** are placed at the bottom of the list. Each order must show their rating details if applicable (See **RD1**).

RC1: Service Delivery

- The system shall allow **FARMERS/RECEPTIONISTS** to track the status of booked spraying order:

  - **Pending**: The order from a **FARMER** was successfully sent to HoverSprite, yet the **RECEPTIONISTS** has not accepted the order.

  - **Cancelled:** The **RECEPTIONISTS** cancelled the order made by **FARMERS.**
    The system sends an email to the **FARMER** to inform that their order **is cancelled**.

  - **Confirmed:** The **RECEPTIONISTS** accepted the order made by **FARMERS**; **orders made by RECEPTIONISTS are initially set as Confirmed**.

    When the order is confirmed, the system sends an email to the **FARMER** to inform that their booking is successful, and *HoverSprite* is assigning technicians to process their order.

  - **Assigned:** The **RECEPTIONISTS** assigns the order to at least one **SPRAYER** who will conduct the service at the **FARMER**'s field. If there is an **Apprentice** **SPRAYER**, the system must ask the receptionist to choose another **Adept** or **Expert** **SPRAYER.**

    The system sends an email to the **FARMER** to inform that their order is assigned to **SPRAYER(S)**. Include the full name of the **SPRAYER(S)** conducting the service in this email. Additionally, an email is sent to each assigned **SPRAYER** with information about the full name, location, and phone number of the **FARMER.**

- **In Progress:** The **SPRAYER** confirmed from their *HoverSprite* account that they started working on the order and is on the way to conduct the spray.

- **Completed:** Both of the following conditions must apply:

  1) The **FARMER** confirmed from their *HoverSprite* account that the spray finished.

  2) **One** assigned **SPRAYER** confirmed that they received the cash payment from their *HoverSprite* account. The system must show the **total cost** of the service, and an optional **receive amount**, and **change amount** at this stage to help the **SPRAYER** computes the change to return to the **FARMER**.

  For example, if the **total cost** is 1,020,000 VND, the **FARMER** pays 1,050,000 VND, thus the **receive amount** is 1,050,000 and the **change** is 30,000 VND.

  The system sends an email to the **FARMER** to inform that their order is assigned to **SPRAYER(S)**. Include the full name of the **SPRAYER(S)** conducting the service.

RD1: Customer Feedback and Service Quality Rating

- The system shall provide a mechanism for **FARMERS** to submit feedback **after the spraying service is Completed.**

- The feedback must provide **a text input field** and a **multiple-choice rating** for overall satisfaction, with 1 as Very Unsatisfied to 5 as Very Satisfied.

- The system must store the collected feedback for internal analysis by *HoverSprite*.

RE1: System Architecture

- The back-end architecture must be designed using a **layer-based** or **module-based Spring Boot architecture**.

- If React is NOT used, there must be a **separation** between page content and style.

Level 2 Requirements

General Requirements:

- **GR21**: The application interface should be intuitive and user-friendly for farmers, **especially for the one with very basic technical skills.**

- **GR22**: The system shall be **responsive** on **desktops** and **smartphones**, with the **content being customized to each device type.**

- **GR23:** Both the front-end and back-end system operates **under the HTTPS protocol.** The system can use self-signed certificates.

RA2: Enhanced Customer Registration and Sign-in

The system supports the requirements in **RA1** and the following extra requirements:

1) Secure Sign-in Protocol

- The system **must NOT use Basic Auth** for authentication. Instead, the system **must apply JSON Web Signature** for authentication and authorization purposes.

- The authentication and authorization information are NOT stored in **Local Storage**, **Session Storage**, or **non-HTTP-only cookies** to prevent cross site scripting attacks.

2) Input Validation

- Besides the password, the system must validate the **full name**, **phone number**, and **email address at the back-end** based on the following rules:

  - A word in a **full name** can have **two non-adjacent capitalized letters**. For example: McKinley, McDonald

  - Full Name of **FARMERS** accepts **BOTH acuted** and **non-acuted** Vietnamese format. E.g., both Trần Bách Nghệ and Tran Bach Nghe are acceptable.

  - Phone number must start with **0** or **+84**, followed by **nine** or **ten digits,** and **can include space**. For example: 0862 123 456, +84 862 123 456, or 0862123456.

  - Email must have a valid local part and the domain part must end with **.com** or **.vn**.

RB2: Enhanced Spray Booking and Order Management

The system supports the **RB1** requirements and the following extra features:

1) Flexible Date Time Specification

- Use a DateTimePicker to help set the desired date of the spray service.

- The DateTimePicker must allow both **date selection** and **typing**. In other words, expert users can type a date, e.g., 15/03/2024, and the value is still accepted by the system.

2) Automatic Suggestion for Sprayer Assignment

- After a successful booking, the system must automatically and optimally suggest the assignment of **SPRAYERS** to process the order based on the following rules:
  - Prioritize **SPRAYERS** who has not accepted any request in the same week
  - Prioritize **Apprentice SPRAYERS**
  - Every **Apprentice SPRAYER** must be accompanied by one **Adept** or **Expert SPRAYERS**
  - One **Expert SPRAYER** is sufficient to handle a request.
  - At most two **SPRAYERS** can be assigned to a request.

3) Online Payment

- Support VISA or MasterCard payment. You can use the card number provided by the sandbox or developer tool. **Your system is not required to demonstrate online payment with real card.**

RC2: Enhanced Service Delivery

The system supports the requirements in **RC1** and the following extra requirements:

- Provide a **Notification Box** feature for **FARMERS** to display notifications when a request is created, cancelled, confirmed, assigned, and completed.

- Provide a **Notification Box** feature for **SPRAYER** to display notifications when a request is assigned.

- The **Notification Box** receives notifications from the server **in real-time**. The **FARMERS/ SPRAYERS** do NOT need to refresh or reload the page to see the notifications.

RD2: Enhanced Customer Feedback and Service Quality Rating

The system supports the requirements in **RD1** and the following extra requirements:

- Besides the **Overall** rating, provide a 1-to-5 Rating to the **attentive**, **friendly**, and **professional** aspect of the **SPRAYER** team.

RE2: Enhanced System Architecture

- The back-end architecture must be designed using a **modular Spring Boot Architecture**.

- For every back-end module, only its **Models** and **Services are accessible to other modules.** The **Controller** and **Repository** of any module are accessible from within its module, but not accessible to the outside modules.

  For example: A **Booking** module handling requests from **FARMERS** to request the **SPRAYER** information from a **Scheduling** module, then the Service of a **Booking** module must access only the Service and Model of the **Scheduling** module to handle the request.

- **If React is NOT used,** every page must separate **HTML content, CSS styling, event handlings,** and **back-end requests** into different files.

- **If React is used,** every component must separate **content** and **back-end requests** into different files. Each component must be cohesive and does not declare the properties or styling of any child component that can be reused.

  For example: an **Order** Management Page shows all requests that a **FARMER** ordered. The **Page** Component shall NOT define the representation, i.e., HTML elements and styling, of each **Order** Component. Instead, the **Page** Component shall pass the **Order** data as parameters when creating each **Order** Component. The **Order** representation is defined inside the **Order** Component.

Level 3 Requirements

The following tasks emphasize robust software engineering principles including modularity, encapsulation, and secure authentication mechanisms, essential for developing scalable and maintainable software systems.

RA3: Advanced Customer Registration and Sign-in

The system supports the **RA2** requirements and the following extra features:

- Validate the **full name**, **phone number**, **email** and **password** of **FARMER** at **both the front-end and back-end subsystems**.
- Implement OAuth 2 for authentication and authorization, allowing **FARMERS** to log in **using either their Gmail, Facebook,** or ***HoverSprite* account**. Integration with the TWO external OAuth 2 provider is mandatory.
    - **Authenticate FARMER** using OAuth 2 through Google or Facebook account.
    - Implement OAuth 2 authorization flow to create an internal *HoverSprite* account with the **FARMER** access level when the **FARMER** successfully signs in with Google or Facebook account for the first time.

RB3: Advanced Spray Booking and Order Management

This enhancement aims to provide a visual and intuitive way for farmers and receptionists to view and select available spraying sessions across a week. By incorporating both Gregorian and Lunar calendar dates, the system acknowledges and respects cultural preferences while improving accessibility and usability.

As a result, the system must support the **RB2** requirements and the following extra features:

1) Display Available Time Slots in a Weekly Calendar Table

- When choosing the desired **date** and **time**, display a table format with 7 columns representing each day of the week (Monday to Sunday). Each column header should include:
    - **Day name** (e.g., Monday, Tuesday, etc.)
    - **Gregorian Calendar date** (dd/mm/yyyy format)
    - **Lunar Calendar date** (dd/mm/yyyy format) with a smaller font size compared to the Gregorian date to differentiate it clearly.
- The table should have **6 rows** representing the six 1-hour time slots available for spraying sessions. **Clearly indicate a separation** between morning and afternoon time slots to avoid confusion.
- Allow **FARMERS** or **RECEPTIONISTS** to view and book available slots directly in the table.

- If a **RECEPTIONISTS** books the timeslot, the system shall prompt for the telephone number first.

- If there exists a **FARMER** with the given telephone number, the system shall automatically fill in the **full name** and **location** of the **FARMER** to provide convenience.

- If there is NO **FARMER** having the phone number, the front-end shall inform the phone number is not found to indicate that the **RECEPTIONIST** need to manually fills in the **FARMER** data.

- Once a slot is booked, it should be visually marked as unavailable for subsequent bookings

- Ensure the table is **responsive** and **user-friendly**, optimizing for both smartphone and PC displays.

2) Paginated or Lazy-loaded Order Management Page

- On the Order Management page, offers Pagination or Lazy loading of multiple spray orders.

RC3: Advanced Service Delivery

This enhancement concerns the user experience of **SPRAYERS** by offering them an optimal route planning. Furthermore, HoverSprite also aims to deliver **real-time update** to keep **FARMERS** and **SPRAYERS** constantly informed of the order status.

As a result, the system must support the **RC2** requirements and the following extra features:

1) Route Planning

- Enable **SPRAYERS** to request the system for planning an **optimal route** that minimizes the travelling distance to all assigned **FARMERS** fields in the same day.

- The route planning algorithm must consider the **current position** of the **SPRAYER** and the locations of all destinations, i.e., the fields of **FARMERS** assigned to the **SPRAYER**.

- The system must show the optimal route from the current position of the **SPRAYER** to the locations of all **FARMER** fields **on a Map API**. The optimal route can be shown either from within your application, or at the website of the Map API. You can consider using OpenStreetMap or Google Maps API.

2) Dual Confirmation of Completed Spray Session

- After finish spraying, a **SPRAYER** uses the *HoverSprite* Web Application to confirm that the service has been delivered to the **FARMER**. The system must then generate a QR Code and show it **on the screen of the SPRAYER**. When the **FARMER** scans this QR code from within their HoverSprite Web Application, the system receives the confirmation from the **FARMER** and sets the status of the order to **Completed**.

  Note: we should use a One-Time Password sent to the **FARMER'S** mobile phone, as this approach is more usable to **FARMER** with very basic technical knowledge. Yet, we deliberately limit the scope of this use case due to physical, complexity, and time constraints.

RD3: Advanced Customer Feedback and Service Quality Rating

- The system must support the RD2 requirements and enable **FARMERS** to upload **at most five images** into their comment to supply evidence or proof that they want to address in the textual feedback.

RE3: Design Modular Monolith Architecture

Design a modular monolith architecture using Spring Boot, ensuring strict module encapsulation and interface-based communication.

- **Modular Design:** Architect the system as a modular monolith where each module provides services through interfaces only, preventing direct access to internal components (**Controller**, **Service**, **Repository**, **Entity** Model) by other modules.

- **Interface-Based Interaction:** Modules can only access services through defined interfaces, ensuring loose coupling and adherence to module boundaries.

  Example: If Module A requires data from Module B, Module B provides a specific interface that Module A can use to access necessary functionality or data in a controlled manner.

- **Entity Model Access:** Each module exposes a read-only, public object model containing a subset of entity attributes required by other modules. For instance, a module may provide a simplified object containing only necessary attributes to meet the needs of other modules. This design promotes data privacy and encapsulation.

  Example*:* A **UserProfile** module contains a UserProfileModel with 20 attributes, but when an **AppointmentBooking** module requires a **UserProfile** with first name, last name, thumbnail image, and telephone, the **UserProfile** module provides a service returning a **read-only** UserProfileBasicPersonalData object, which is accessible inside the **AppointmentBooking** module. As a result, the **AppointmentBooking** Service <u>can only access the four attributes</u> in the UserProfileBasicPersonalData, but not other attributes of the UserProfileModel Entity Object.

Contribution Requirements

Every team member is expected to contribute into **ALL** of the following tasks by producing recognizable deliverables or providing substantial feedback:

1. Design Data Model

2. Design System Architecture

3. UX/UI Design of **ALL** Requirements

4. Documentation

5. Database Construction

6. Front-end Development

7. Back-end Development

For development tasks, each member is expected to **contribute either in individual or pair-programming**. The amount of fair contribution is determined by the complexity of the supported features and average workload of the entire team.

Penalty

P1) In your report, **briefly describe HOW you implement the RE3 requirements**. **Missing the description** will lead to **2 points deduction** of your total mark.

P2) Specify the requirements that your system supported by filling in the sheet **ImplementationSummary.xlxs** in the assignment instruction. **False declaration** of provided features results in up to **2 points deduction.**

P3) NOT Using GitHub frequently during the project **results in up to 10 points penalty** of your assessment grade**.**

P4) In your report**, submit the proof of GitHub contribution** throughout the project duration

P5) In **Canvas,** submit the **zip file of your GitHub repository.**

Missing either P4) or P5) results in two points penalty. Missing BOTH 2) and 3) results in four points penalty.

End of Requirement Specification