INVENTRA –

INTELLIGENT INVENTORY MANAGEMENT SYSTEM

Presented by:
Momina Shaik

Problem Statement

- **The Challenge:** Businesses (retail, warehouses, e-commerce) struggle with inefficient stock tracking, leading to overstocking or stockouts.
- **Security Risk:** Without a robust system, inventory data is vulnerable to unauthorized access, leading to data inconsistency or theft.
- **Need:** A secure, automated, and role-based full-stack Java application to centralize inventory operations.

Existing Problem

- **Manual Tracking:** Many SMEs still use spreadsheets or paper-based logs, which are prone to human error.

- **Lack of Security:** Standard legacy systems often lack role-based access control, meaning any user can modify sensitive purchase orders or sales records.

- **No Real-time Visibility:** Decisions are made based on outdated data rather than live stock updates or automated alerts.

Proposed System (Inventra)

- **Smart Automation:** A Java-based system that offers real-time stock updates and low-stock alerts.

- **Role-Based Access:** High-level security through Spring Security to ensure only authorized personnel (Admin, Manager, Staff) can perform specific actions .

- **Comprehensive Features:** Integrated modules for products, suppliers, purchase orders, sales, and detailed analytics .

Modules Description

- **Module 1:** Authentication & Security: Handles User Signup, Sign-in, and Password recovery using JWT tokens.
- **Module 2:** Product & Stock Management: CRUD operations for inventory, including categorization and expiry tracking .
- **Module 3:** Supplier & Purchase Orders: Managing vendor directories and tracking PO status (Pending/Received) .
- **Module 4:** Sales & Analytics: Processing sales, generating invoices, and viewing business performance dashboards .

Architecture

- **Frontend:** HTML5, CSS3, and Bootstrap for a responsive UI.
- **Backend:** Spring Boot framework utilizing Java 17/21 for robust logic.
- **Database:** MySQL for persistent data storage.
- **API Layer:** RESTful APIs for seamless communication between the client and server.
- **Security:** Spring Security with JWT (JSON Web Token) for stateless authentication.

Classes in Module

The Authentication module for Inventra is implemented using a clean, layered architecture to separate concerns and improve maintainability.

Controller Layer: Handles incoming web requests and navigation logic.

- **AuthController:** Manages user login and registration flows **.**
- **DashboardController:** Handles access to the main user dashboard.
- **ResetPasswordController:** Dedicated controller for handling password recovery requests.

Service Layer: Contains the core business logic of the application .

- **AuthService:** Validates user credentials, manages registration rules, and processes password resets

Model Layer: Defines the data structure.

- **User:** An entity class that represents the user data stored in the database**.**

Repository Layer: Handles database communication.

- **UserRepository:** An interface that allows the application to perform CRUD operations on the MySQL database.

Resources & Templates: The frontend views that interact with these classes**.**

- **HTML files** like **login.html, signup.html, and dashboard.html** provide the user interface.

Pseudocode (Part 1: Sign-up)

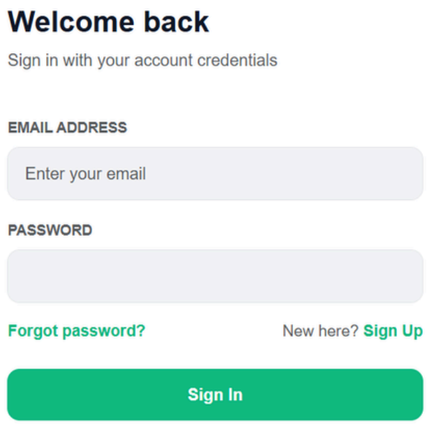
Controller Layer

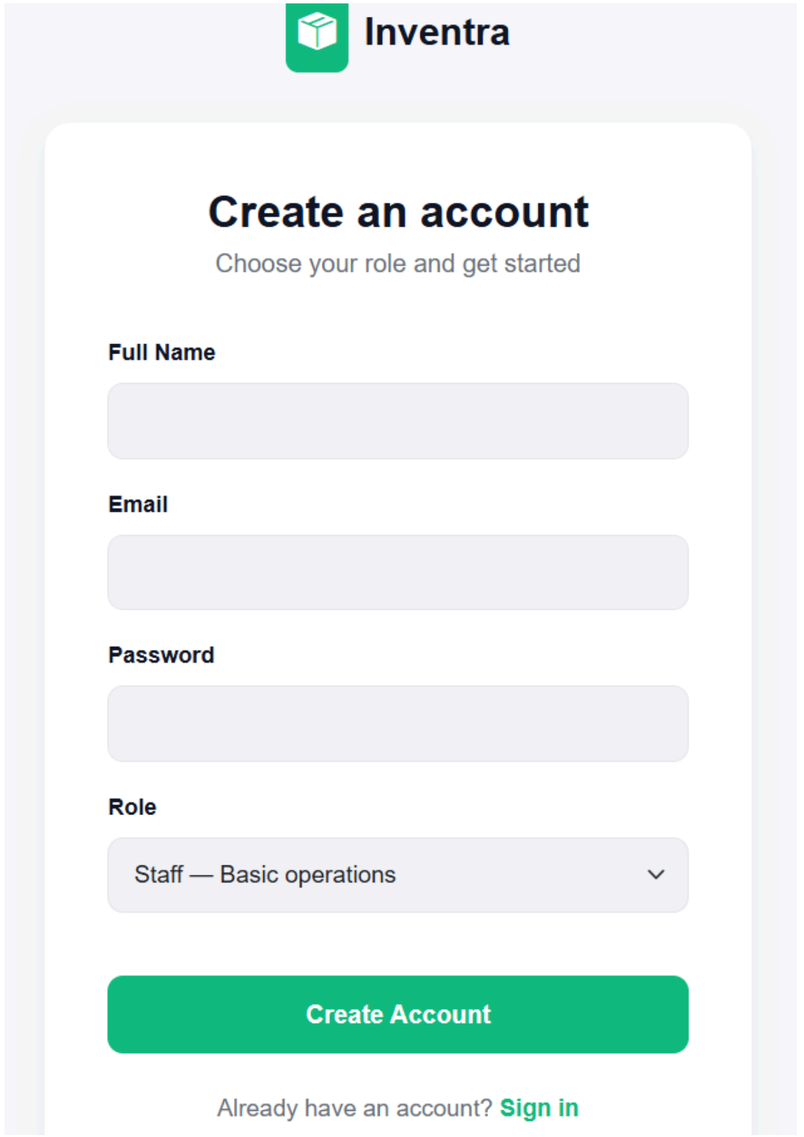
```
FUNCTION signupSubmit(user_data):
    // Attempt to register the user via the AuthService
    RESULT = authService.register(user_data)

    IF RESULT is FALSE:
        // Case: User already exists
            ADD "Username or Email already exists" to Error Message
        STAY on "signup" page and show error

    ELSE:
        // Case: Registration successful
            ADD success message: "Congratulations! Account created successfully."
        REDIRECT user to the "login" page (root URL)
    END IF
END FUNCTION
```

Service Layer

```
FUNCTION register(user):
    // Step 1: Validation
      IF userRepository finds that username ALREADY EXISTS OR
       userRepository finds that email ALREADY EXISTS:
        RETURN FALSE
    END IF

    // Step 2: Set Default Values
    IF user.role is empty:
       SET user.role to "STAFF"
    END IF

    // Step 3: Persistence
    SAVE user to the database using userRepository

    RETURN TRUE
END FUNCTION
```

Screenshots - SignUp





Pseudocode (Part 2: Sign-in)

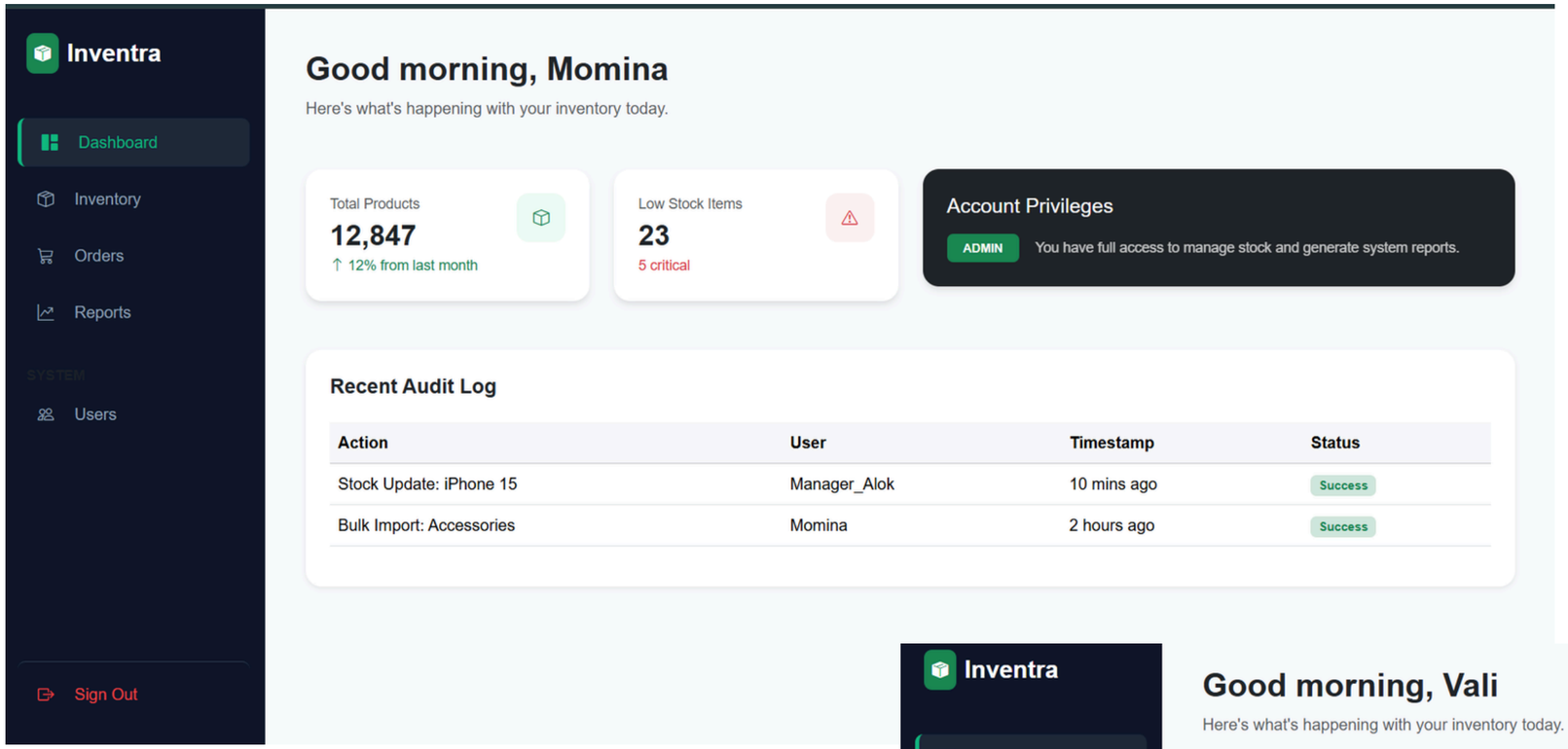
Controller Layer

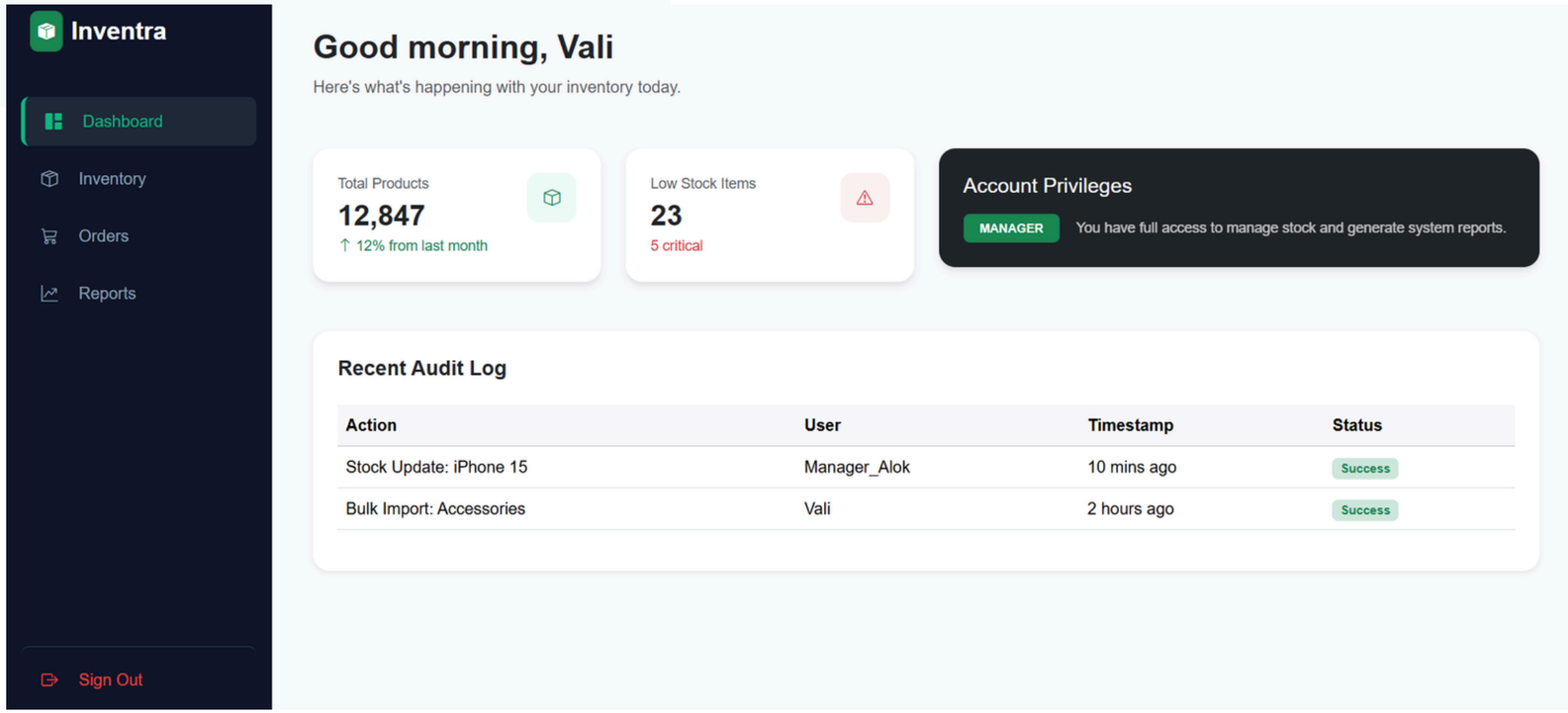
```
FUNCTION login(email, password, session):
    // Step 1: Request Authentication from the Service
    USER_OBJECT = authService.authenticateByEmail(email, password)

    // Step 2: Validate the Result
    IF USER_OBJECT is NULL:
        // Case: Credentials don't match
        ADD "Invalid email or password" to Error Message
        RETURN the "login" view to show the error

    ELSE:
        // Case: Credentials are correct
        // Step 3: Establish Session
        STORE USER_OBJECT in the current HTTP Session as "user"

        // Step 4: Redirection
        REDIRECT the browser to the "/dashboard" route
    END IF
END FUNCTION
```

Service Layer

```
FUNCTION authenticateByEmail(email, password):
    // Step 1: Look for the email in the database
    OPTIONAL_USER = userRepository.findByEmail(email)

    // Step 2: If user exists, check the password
    IF OPTIONAL_USER exists AND OPTIONAL_USER.password EQUALS password:
        RETURN the USER_OBJECT
    ELSE:
        RETURN NULL
    END IF
END FUNCTION
```

Screenshots-SignIn





Pseudocode (Part 3: Password Recovery)

Controller Layer

```
FUNCTION processForgotPassword(email):
    // Step 1: Ask service to handle the request
    IF authService.resetPasswordRequest(email) is TRUE:
        // Case: Email exists and mail was triggered
            ADD "A reset link has been sent to your registered email" to Message
    ELSE:
        // Case: Email not found in MySQL
        ADD "Email not found in our system" to Error Message
    END IF

     RETURN the "forgot-password" view to show the result
END FUNCTION
```

Reset Action

```
FUNCTION updatePassword(email, newPassword):
    // Step 1: Find the user again to ensure they still exist
    OPTIONAL_USER = userRepository.findByEmail(email)

    IF OPTIONAL_USER exists:
        // Step 2: Update the password field
        SET OPTIONAL_USER.password = newPassword
        // Step 3: Save changes to the database
        userRepository.SAVE(OPTIONAL_USER)
    END IF
END FUNCTION
```

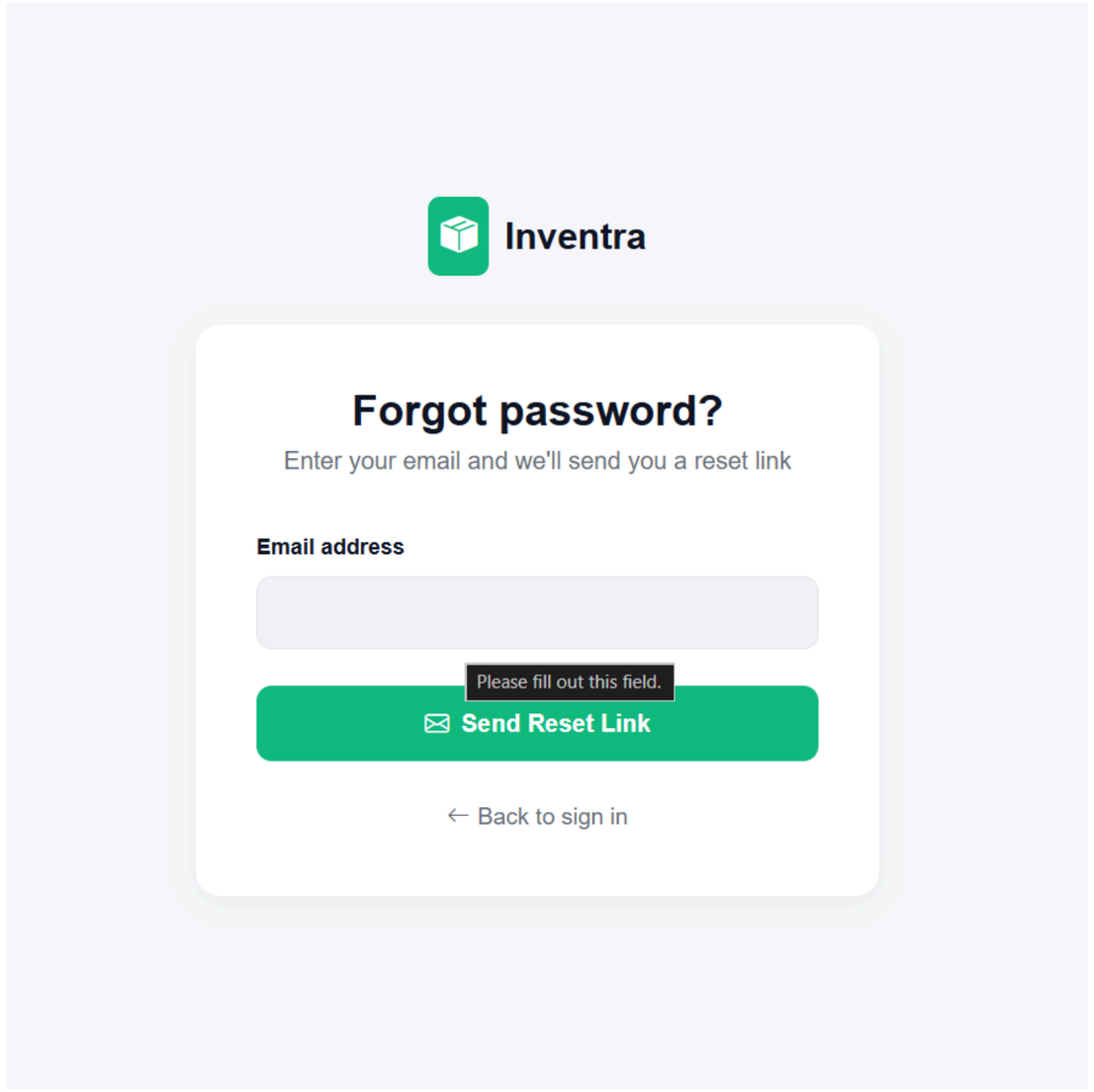
Pseudocode (Part 3: Password Recovery)

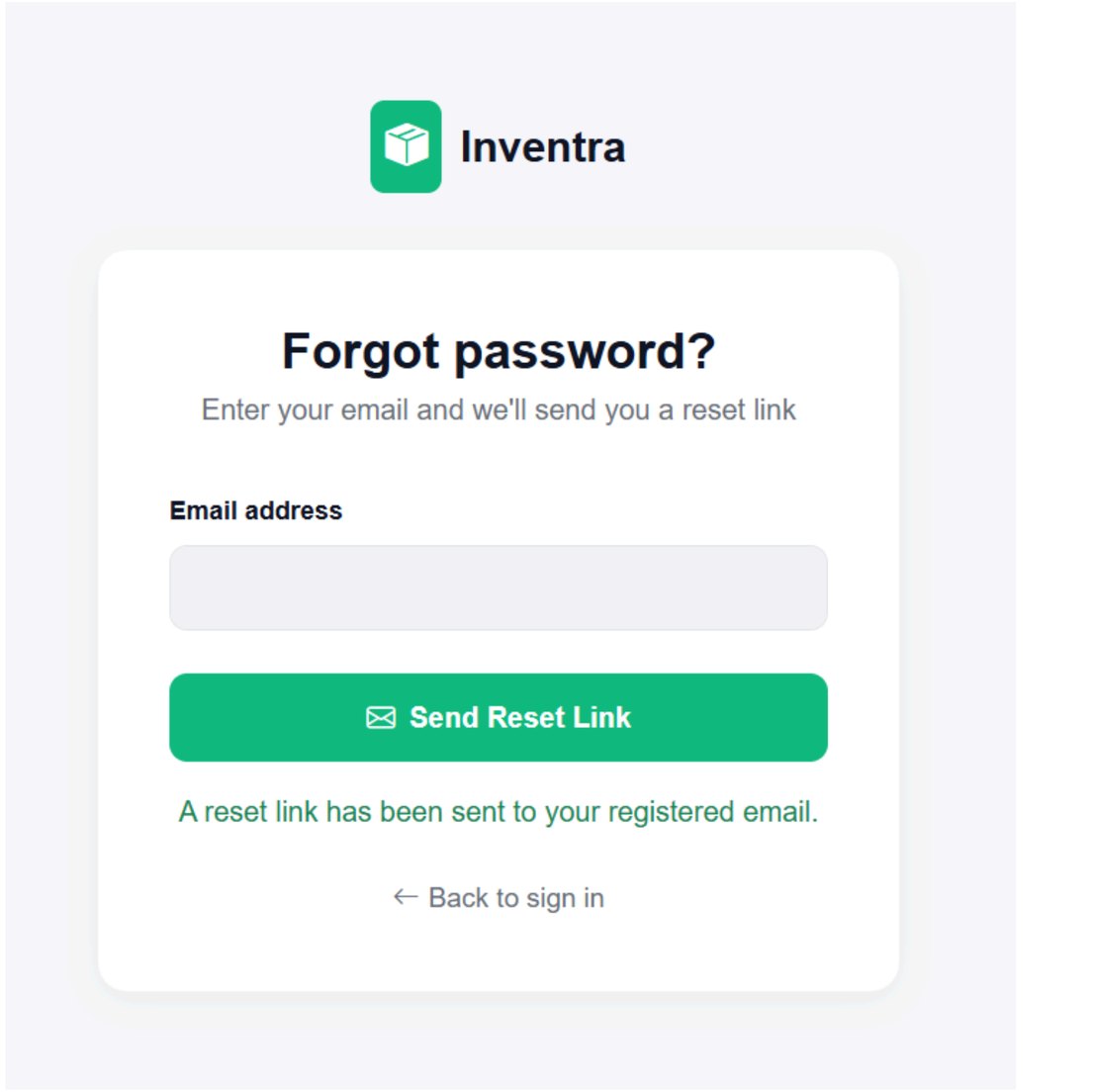
Service Layer

```
FUNCTION resetPasswordRequest(email):
    // Step 1: Search for user in repository
    OPTIONAL_USER =
userRepository.findByEmail(email)

    IF OPTIONAL_USER exists:
        // Step 2: Trigger the email helper
        CALL sendResetEmail(email)
        RETURN TRUE
    ELSE:
        RETURN FALSE
    END IF
END FUNCTION
```

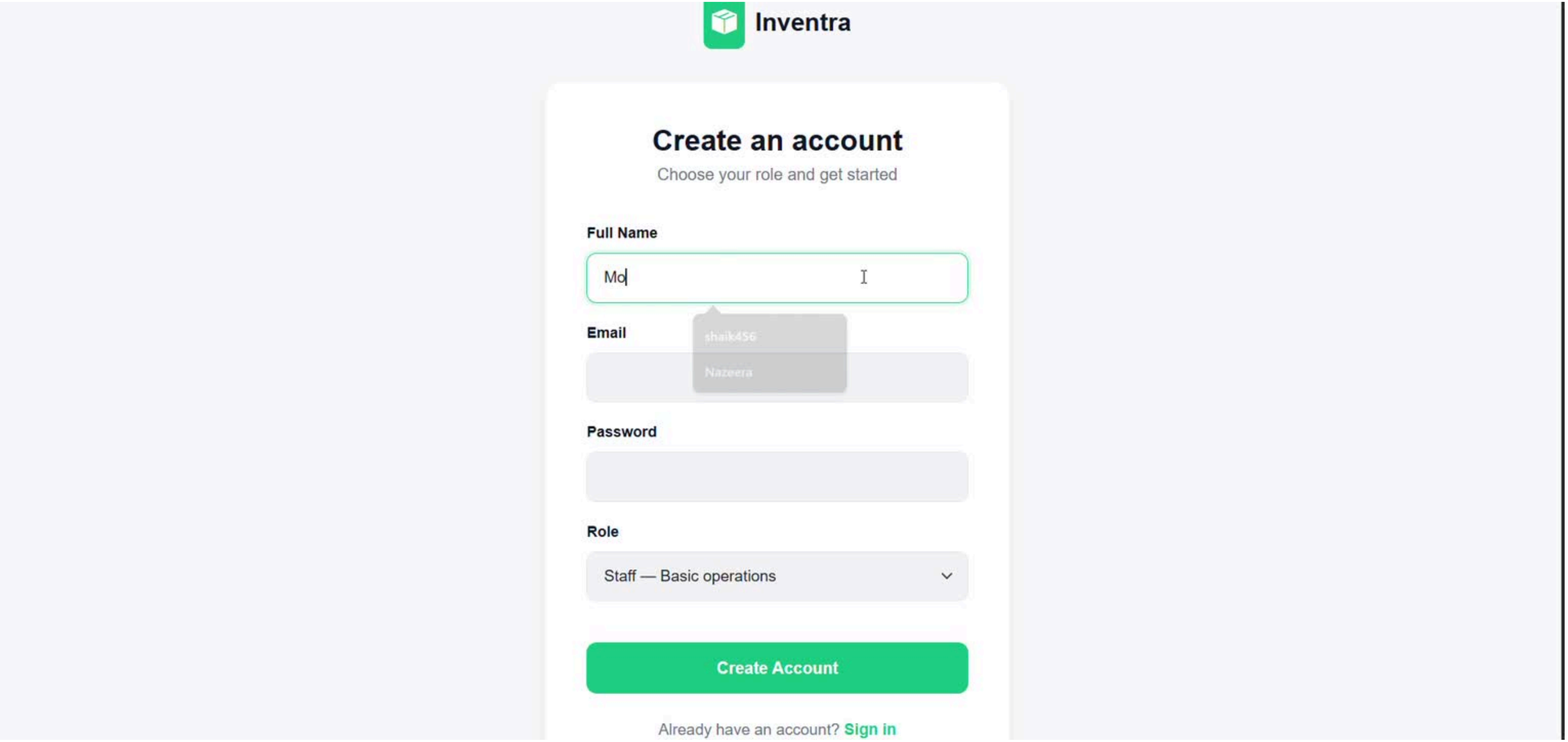
```
FUNCTION sendResetEmail(toEmail):
    TRY:
        CREATE a new Mail Message
        SET SENDER to "your-email@gmail.com"
        SET RECEIVER to toEmail
        SET SUBJECT to "Inventra - Password Reset
Request"
        SET TEXT to "Link: http://localhost:8080/reset-
password?email=" + toEmail

        // Step 3: Send via JavaMailSender
        mailSender.SEND(message)
    CATCH Error:
        LOG "Error sending email"
    END TRY
END FUNCTION
```

Screenshots - Forgot Password





Demo Video


Inventra
Create an account
Choose your role and get started
Full Name
Mo
shaik456
Nazeera
Email
Password
Role
Staff — Basic operations
Create Account
Already have an account? Sign in

Conclusion

- **Security Foundation:** Successfully established a secure entry point for Inventra using Spring Security.
- **Robustness:** Implemented password encryption and JWT tokens to prevent unauthorized data exposure.
- **Scalability:** The architecture allows for easy addition of more specific user roles as the system grows .

Thank You