Inventra – Intelligent Inventory Management System

Presented By

Christina J

MODULE 1 -AUTHENTICATION MODULE

1. AuthController-Handles HTTP Requests
2. AuthService-Business logic and validation
3. LoginUser-User entity/model
4. UserRepository-Used for Database operations
5. JWTUtility-Token generation &verification
6. EmailService (for forgot password)-Send password reset to emails

MODULE 2-PRODUCT/INVENTORY MANAGEMENT

1. Product
2. ProductRepository
3. ProductService
4. ProductController

MODULE 3:INVENTORY/ALERTS

AlertService Class
NotificationService Class

In this module monitors the minimum stock levels. When the stock<threshold then the alert appers on the dahboard.This mainly helps to avoid the stock-out situations.

PROBLEMS IN THE EXISTING SYSTEM

CURRENT CHALLENGES

- MANUAL PRODUCT ENTRY
- NO REAL-TIME STOCK VISIBILITY
- FREQUENT STOCKOUTS AND OVERSTOCK
- NO EXPIRY DATE TRACKING
- LACK OF CENTRALIZED CONTROL

Module 1-Authentication Module/role based login

- This mainly allows a user to log into the system.
- The system checks the username and password,and if the user is valid,it allows access to the application.

Module 1-SIGN-UP(Register user)

Purpose

- Create a new user account
- Only Admin can register employees

AuthController – Signup

```
CLASS AuthController

  FUNCTION signup(request)
userDetails = request data
result = AuthService.registerUser(userDetails)
RETURN result
  END FUNCTION
END CLASS
```

1.SIGN-UP(Register User)

AUTHSERVICE-SIGNUP

```
CLASS AuthService

   FUNCTION registerUser(userDetails)

existingUser = UserRepository.findByUsername(userDetails.username)

       IF existingUser exists THEN
RETURN "User already exists"
       END IF

encryptedPassword = encrypt(userDetails.password)

newUser = create User
newUser.username = userDetails.username
newUser.password = encryptedPassword
newUser.role = userDetails.role
newUser.email = userDetails.email

UserRepository.save(newUser)

RETURN "Signup successful"
   END FUNCTION
END CLASS
```

Step-by-step Explaination

1. Check if username already exists
2. Encrypt password for security
3. Create new user object
4. Save user in database
5. Return success message

SIGN-IN(LOGIN USER)

PURPOSE:

Authenticate user

Generate secure token

AUTHCONTROLLER – SIGNIN

```
CLASS AuthController
   FUNCTION signin(request)
username = request.username
password = request.password
token = AuthService.authenticate(username, password)
RETURN token
   END FUNCTION
END CLASS
```

SIGN-IN(LOGIN USER)

AUTHSERVICE- SIGNIN

```
CLASS AuthService

   FUNCTION authenticate(username, password)

       user = UserRepository.findByUsername(username)

       IF user does not exist THEN
RETURN "Invalid username"
       END IF

       IF password matches encrypted password THEN
token = JWTUtility.generateToken(user)
RETURN token
       ELSE
RETURN "Invalid password"
        END IF
   END FUNCTION
END CLASS
```

FORGOT PASSWORD

PURPOSE:

Authenticate user

Generate secure token

AUTHCONTROLLER

```
CLASS AUTHCONTROLLER

    FUNCTION FORGOTPASSWORD(REQUEST)
        EMAIL = REQUEST.EMAIL
RESULT = AUTHSERVICE.PROCESSFORGOTPASSWORD(EMAIL)
RETURN RESULT
    END FUNCTION
END CLASS
```

FORGOT PASSWORD

AUTHSERVICE-FORGOT PASSWORD LOGIC

```
CLASS AUTHSERVICE

   FUNCTION processForgotPassword(email)

       user = UserRepository.findByEmail(email)

       IF user does not exist THEN
RETURN "Email not registered"
       END IF

resetToken = generateResetToken()
       save resetToken with user

EmailService.sendResetLink(email, resetToken)

RETURN "Password reset link sent"

   END FUNCTION

END CLASS
```

EMAILSERVICE-Reset Email

```
CLASS EmailService

    FUNCTION sendResetLink(email, token)
create reset password link using token
        send email to user
    END FUNCTION


END CLASS
```

RESET PASSWORD

AUTHCONTROLLER

```
CLASS AuthController

   FUNCTION resetPassword(token,
newPassword)
result = AuthService.resetPassword(token,
newPassword)
RETURN result
   END FUNCTION

END CLASS
```

RESET PASSWORD

AUTHSERVICE

```
CLASS AUTHSERVICE

   FUNCTION RESETPASSWORD(TOKEN, NEWPASSWORD)

       USER = FIND USER BY RESET TOKEN

       IF TOKEN INVALID OR EXPIRED THEN
RETURN "INVALID TOKEN"
       END IF

ENCRYPTEDPASSWORD = ENCRYPT(NEWPASSWORD)
USER.PASSWORD = ENCRYPTEDPASSWORD

       CLEAR RESET TOKEN
       SAVE USER

RETURN "PASSWORD RESET SUCCESSFUL"
END FUNCTION
END CLASS
```

FLOW

SIGNUP
UI → CONTROLLER → SERVICE → REPOSITORY → DATABASE

SIGNIN
UI → CONTROLLER → SERVICE → JWT TOKEN → UI

FORGOT PASSWORD
UI → CONTROLLER → SERVICE → EMAIL SERVICE → USER

PRODUCT/INVENTORY MANAGEMENT

PRODUCT(MODEL)-STORES PRODUCT DETAILS
PRODUCTREPOSITORY-ONLY PERFORMS DB QUERIES
PRODUCT SERVICE-BUSSINESS LOGIC LAYER
PRODUCTCONTROLLER-REQUEST-HANDLING LAYER

MODULE-2(PRODUCT/INVENTORY

PURPOSE

- Manage products
- Track stock levels
- Prevent duplicate items

* SKU(Stock keeping unit):It’s mainly work as an unique identifier to prevent duplicates.
* Also works in the Stock Tracing Real-time updates on inventory levels
* MinStockLEvel:Threshold for low stock alerts

PRODUCT CLASS(Entity)

```
 CLASS Product
   productId
   sku
   name
   category
   supplier
   unitPrice
stockQuantity
minStockLevel
END CLASS
```

EXPLAINATION:

sku ensures uniqueness
stockQuantity changes frequently
minStocklevel may used for alert

PRODUCTCONTROLLER CLASS

```
 CLASS ProductController
   FUNCTION addProduct(productData)
ProductService.addProduct(productData)
   END FUNCTION

   FUNCTION stockIn(productId, quantity)
ProductService.increaseStock(productId, quantity)
   END FUNCTION

   FUNCTION stockOut(productId, quantity)
ProductService.decreaseStock(productId, quantity)
   END FUNCTION
END CLASS
```

EXPLAINATION:

- Receives UI requests
- Sends work to service layer
- Does not touch database directly

PRODUCTSERVICE CLASS

```
CLASS ProductService

   FUNCTION addProduct(productData)
       IF product SKU not exists THEN
save product
       ELSE
   RETURN "Duplicate Product"
       END IF
     END FUNCTION

   FUNCTION increaseStock(productId, quantity)
product.stock += quantity
       save product
TransactionService.log("STOCK_IN", productId, quantity)
AlertService.checkLowStock(product)
   END FUNCTION
FUNCTION decreaseStock(productId, quantity)
       IF product.stock >= quantity THEN
product.stock -= quantity
save product
TransactionService.log("STOCK_OUT", productId, quantity)
AlertService.checkLowStock(product)
       ELSE
RETURN "Insufficient Stock"
       END IF
   END FUNCTION
END CLASS
```

Explaination:

Central inventory logic
Updates stock
Logs transactions
Triggers alerts

PRODUCTCONTROLLER CLASS

```
CLASS ProductRepository

    FUNCTION findById(productId)
RETURN product
    END FUNCTION

    FUNCTION save(product)
        store product in database
    END FUNCTION

END CLASS
```

PRODUCTCONTROLLER CLASS

```
CLASS AlertService

    FUNCTION checkLowStock(product)
        IF product.stock < product.minStockLevel THEN
NotificationService.sendAlert(product)
        END IF
    END FUNCTION

END CLASS
```

Explanation

- Automatically triggered after stock change
- Works silently in background

NotificationService CLASS

```
CLASS AlertService
 FUNCTION checkLowStock(product)
 IF product.stock < product.minStockLevel THEN
NotificationService.sendAlert(product)
 END IF
 END FUNCTION
END CLASS
```

Explanation

- Automatically triggered after stock change
- Works silently in background

Explanation

- Uses communication channels
- Improves operational efficiency

MODULE 3:INVENTORY ALERTS

Product(reused)-Comes from module-2
Alert(model)-Stores triggered alerts
AlertRepository-only reads/writes DB
AlertService-Contains bussiness rules
AlertController-Handles UI/API requests

Module 3

The alert module is responsible for monitoring inventory conditions and notifying users automaticaly when undefined rules are violated.
The alert module continuously monitors inventory conditions and automatically notifes users about low,expiry,reorder and critical inventory situations.

Module 3:Inentory/Alerts

```
CLASS AlertService
 FUNCTION checkLowStock(product)
 IF product.stock < product.minStockLevel THEN
NotificationService.sendAlert(product)
 END IF
 END FUNCTION
END CLASS
```

Explanation

- Automatically triggered after stock change
- Works silently in background

NotificationService Class

```
CLASS NotificationService
FUNCTION sendAlert(product)
show dashboard alert
send email or SMS
 END FUNCTION
END CLASS
```

Explanation

- Uses communication channels
- Improves operational efficiency

NotificationOverflow

- Product stock updated
- Alert Service checks quantity
- Low stock detected
- Alert created
- Alert Repository saves alert
- Notification Service sends Email / SMS
- User receives notification
- Alert visible on dashboard

THANK YOU