Comprehensive Exercise Report

Team <<RRR>> of Section <<007>>

Abdullah Ansari - 231ADB054

Rahul Singh Garhwal - 231ADB026

Shihan Peries -221ADB129

Adkhamjon Ahrolhanov - 221ADB222

Karthik Bharadwaj Poduru- 250ADB053

Requirements/Analysis

Week 2

Journal

The following prompts are meant to aid your thought process as you complete the requirements/analysis portion of this exercise. Please respond to each of the prompts below and feel free to add additional notes.

- ❖ After reading the client's brief (possibly incomplete description), write one sentence that describes the project (expected software) and list the already known requirements.

- The project involves creating a website for **Ausmena Kebabs** where customers are able to see available dishes and place orders.
  - Enable online ordering for customers which generates invoice for both parties
- Ensure the website remains functional while adding new features.

- ❖ After reading the client's brief (possibly incomplete description), what questions do you have for the client? Are there any pieces that are unclear? After you have a list of questions, raise your hand and ask the client (your instructor) the questions; make sure to document his/her answers.

1. **Online Ordering:**

- What payment methods should be integrated?
  - ☐ Credit card, PayPal, Apple Pay, etc.

- Should users be able to order as guests, or is account creation required?
  - ☐ Customers can be able to order as a guest if payment is made online with no cash delivery option available for guest users.
- Should the system support scheduled orders
  - ☐ Customers can have access to schedule their food.

2. **Feedback System:**

- Should reviews be publicly visible or private to the restaurant management?
  - ☐ Reviews can be publicly visible

- Should customers be able to rate orders with stars (e.g., 1-5 stars) or just text reviews?
    - [ ] Customers have option to rate between 1-5 stars

3. **Employee Portal:**

- Who will manage shift scheduling (manager or automated system)?
    - [ ] Manager will manage shift scheduling

- Should employees be able to swap shifts among themselves?
    - [ ] Yes , but I need to inform Manager also..

- How will holiday requests be approved?
    - [ ] System will check that did you passed the working requirements or not like 20 H/week
      If yes then holiday is granted except medical cases.

4. **Technical Considerations:**

- Should the website be mobile-friendly or have a dedicated mobile app?
    - [ ] Website would be mobile-friendly
- Do they require multi-language support (Latvian, English, etc.)?
    - [ ] No,English is by default or any one can translate by website.

- Does the project cover topics you are unfamiliar with? If so, look up the topics and list your references.

1. Secure Online Payment Integration
- Researching **Stripe, PayPal, and other payment gateways** for secure transactions.
- Reference: Stripe Developer Docs, PayPal Developer Guide

2. **Best Practices for Online Food Ordering Systems**
- Understanding **UX/UI design** for restaurant ordering platforms.
- Reference: Nielsen Norman Group – Usability Guidelines

3. Employee Shift Scheduling Systems
- Looking into automated **shift management tools** and scheduling algorithms.
- Reference: Workforce Scheduling Best Practices

4. GDPR Compliance for Employee and Customer Data
- Ensuring that personal data storage complies with **EU privacy laws**.
- Reference: GDPR Official Website

- Describe the users of this software (e.g., small child, high school teacher who is taking attendance).

1. **Customers (General Public & Regular Visitors)**

- Individuals who want to **order food online** from Ausmena Kebabs.
- Can be anyone from students to working professionals looking for a quick and easy way to place an order.

2. **Restaurant Employees**

- Staff members who need to **check their work shifts, request holidays, and access pay slips**.
- Could include **chefs, waiters, and delivery personnel**.

3. **Restaurant Managers/Admins**

- Responsible for **scheduling employee shifts, approving holiday requests, and managing pay slips**.
- May also handle **customer feedback and order management**.

4. **Website Administrators (IT Support/Developers)**

- Responsible for **maintaining the website**, fixing bugs, updating menu items, and ensuring smooth operations.

- Describe how each user would interact with the software

User Interactions with the Software

1. Customers (General Public & Regular Visitors)

- **Browse the menu** to see available food items.
- **Place online orders** by selecting items and adding them to the cart.
- **Make payments** using integrated payment options (credit card, PayPal, etc.).
- **Track their orders** (e.g., order status: "preparing," "out for delivery," "delivered").
- **Leave feedback** or rate their experience after receiving their order.
- **Create and manage accounts** (optional) to view order history and saved preferences.

2. Restaurant Employees (Staff Members: Chefs, Waiters, Delivery Personnel)

- **Log in to the employee portal** using their credentials.
- **Check work schedules** (upcoming shifts, weekly schedule).
- **Request time off** through the system.
- **View/download payslips** when needed.
- **Receive notifications** about shift changes or important updates from managers.

3. Restaurant Managers/Admins

- **Log in to the admin panel** with elevated access.
- **Manage employee shifts** (assigning schedules, approving shift swaps).
- **Approve or deny holiday requests** based on staff availability.
- **Upload and manage pay slips** for employees.
- **View and respond to customer feedback** to improve service quality.
- **Monitor incoming orders** and ensure smooth operations.

4. Website Administrators (IT Support/Developers)

- **Monitor website performance** and fix bugs or technical issues.
- **Update menu items and pricing** as per restaurant management's request.
- **Ensure security compliance** (especially for payment processing and personal data).
- **Maintain database integrity** for order history, customer feedback, and employee records.

- What features must the software have? What should the users be able to do?

Required Features & User Capabilities

- **1. Customer Features**

  Menu Browsing – View food items, descriptions, and prices.
  Online Ordering System – Select items, add to cart, and place orders.
  Payment Integration – Pay using credit/debit cards, PayPal, or other methods.
  Order Tracking – View real-time order status updates.
  Feedback & Ratings – Leave reviews and rate orders.
  Account Management (Optional) – Save order history and preferences.

- **2. Employee Features**

  Employee Login – Secure access to personal work details.
  Shift Scheduling – View assigned shifts and schedules.
  Holiday Requests – Submit requests for time off.
  Pay Slip Access – Download/view salary details.
  Shift Swap Requests (Optional) – Request shift changes with manager approval.

- **3. Manager/Admin Features**

  Admin Dashboard – Oversee employees and customer interactions.
  Shift Management – Assign, modify, and approve shifts.
  Holiday Approvals – Accept or reject employee leave requests.
  Pay Slip Management – Upload and distribute salary slips.
  Customer Order Monitoring – Track incoming and completed orders.
  Feedback Management – Review and respond to customer reviews.

- **4. Website Administration & Security Features**

User Authentication – Secure login for customers, employees, and managers.
Data Security & GDPR Compliance – Protect user and payment information.
Multi-language Support – Provide interfaces in Latvian and English.
Performance Monitoring – Ensure the website runs smoothly without downtime.

- **Scalability:** The system should be designed in a way that allows future expansion (e.g., adding more locations or new features).
- **Mobile-Friendly Design:** Ensure that the website is fully responsive and works well on mobile devices.
- **User Experience (UX):** The interface should be intuitive and easy to navigate for both customers and employees.
- **Notification System:** Implement email/SMS notifications for order updates, shift reminders, and other important alerts.
- **Performance Optimization:** The website should load quickly and handle multiple concurrent users efficiently.
- **Backup & Recovery:** Implement a data backup system to prevent data loss in case of system failures.
- **Integration with Social Media:** Allow customers to share their orders/reviews on social media for marketing purposes.

Software Requirements

Project Title: Ausmena Kebabs Website Enhancement

1. Project Overview

The **Ausmena Kebabs Website Enhancement Project** aims to improve the existing website by integrating an **online ordering system**, a **customer feedback feature**, and an **employee portal** for shift scheduling, holiday tracking, and pay slip access. The system will provide a **seamless user experience** for customers placing orders and for employees managing their work schedules. Additionally, managers will have administrative access to handle employee requests and monitor business operations. The website will be **secure, mobile-friendly, and optimized for performance**.

---

2. Functional Requirements

2.1 Customer Features

➔ **Menu Browsing** – View food items, descriptions, and prices.
Online Ordering System – Select items, add to cart, and place orders.
Payment Integration – Pay using credit/debit cards, PayPal, etc. (No cash on delivery for guest users).
Order Tracking – View real-time order status updates.
Feedback & Ratings – Leave reviews and rate orders (1-5 stars, publicly visible).
Account Management (Optional) – Save order history and preferences.
Scheduled Orders – Customers can pre-order meals for a specific time.

2.2 Employee Portal Features

➔ **Employee Login** – Secure access for restaurant staff.
Shift Scheduling – View assigned shifts and work schedules.
Holiday Requests – Employees can request time off (granted if they meet the 20-hour weekly work requirement).
Pay Slip Access – View and download salary slips.
Shift Swap Requests – Employees can swap shifts with manager approval.

2.3 Manager/Admin Features

➔ **Admin Dashboard** – Manage orders, employees, and feedback.
Shift Management – Assign, modify, and approve shifts.
Holiday Approvals – Accept or reject leave requests.
Pay Slip Management – Upload and distribute salary slips.
Customer Order Monitoring – Track incoming and completed orders.
Feedback Management – View and respond to customer reviews.

2.4 Website Administration & Security Features

➔ **User Authentication** – Secure login system for customers, employees, and managers.
Data Security & GDPR Compliance – Protect user and payment information.
Multi-language Support – Default in English (users can translate via browser).
Performance Monitoring – Ensure website speed and uptime.
Backup & Recovery – Prevent data loss in case of system failures.
Notification System – Email/SMS alerts for order updates, shift reminders, etc.

---

3. Non-Functional Requirements

➔ **Scalability** – Support future expansions (e.g., more restaurant locations).
Mobile Responsiveness – Websites should work seamlessly on all devices.
High Performance – Fast loading times, even with multiple concurrent users.
Usability – Intuitive user interface for both customers and employees.
Social Media Integration – Allow customers to share orders and reviews.

---

4. User Stories

As a Customer:

- I want to **browse the menu** so that I can see available food options.
- I want to **place an order online** so that I can enjoy my meal without visiting the restaurant.
- I want to **make payments online** so that I can complete my order securely.
- I want to **track my order status** so that I know when it will arrive.
- I want to **leave feedback and rate my experience** so that I can help improve service quality.

As an Employee:

- I want to **log in securely** so that I can access my work-related details.
- I want to **view my assigned shifts** so that I know when I need to work.
- I want to **request time off** so that I can manage my personal schedule.

- I want to **download my pay slips** so that I can keep track of my earnings.
- I want to **swap shifts with my colleagues** so that I have flexibility in my schedule.

As a Manager/Admin:

- I want to **assign and modify employee shifts** so that work schedules are optimized.
- I want to **approve or deny holiday requests** based on staff availability.
- I want to **upload payslips** so that employees can access them easily.
- I want to **monitor customer orders** so that I can ensure timely deliveries.
- I want to **respond to customer feedback** so that I can improve customer satisfaction.

Black-Box Testing

Instructions: Week 4

Journal.

- What does input for the software look like (e.g., what type of data, how many pieces of data)?
    - Input will be majorly the customers credentials to login the website, then after all required inputs for ordering food which includes a list of food. Also the payment credentials will be the input.
- What does output for the software look like (e.g., what type of data, how many pieces of data)?
    - Customers will get a full invoice of their order which will be one output.
- What equivalence classes can the input be broken into?
    - Valid user credentials (correct username and password)
    - Invalid user credentials (incorrect username or password)
    - Valid food selection (available items from the restaurant)
    - Invalid food selection (out-of-stock or non-existent items)
    - Valid payment details (correct card or payment method)
    - Invalid payment details (expired card, incorrect details, insufficient balance)
- What boundary values exist for the input?
    - Minimum and maximum length of username/password
    - Minimum and maximum quantity of food items in an order
    - Maximum cart value limit for an order

    - Minimum required balance for a successful payment

- Are there other cases that must be tested to test all requirements?
    - Successful login and ordering flow
    - Failed login due to incorrect credentials
    - Ordering with an empty cart
    - Selecting an unavailable food item
    - Payment failure due to insufficient funds
    - Successful payment processing
    - Generating and displaying the invoice correctly

- Other notes:

    - Verify that the invoice reflects correct item details, price, and payment confirmation.
    - Test responsiveness and UI elements across different devices.
    - Ensure security checks for user authentication and payment transactions.

Black-box Test Cases

Test ID	**Description**	**Expected Results**	**Actual Results**
TC001	Login with valid credentials	Redirect to dashboard	Successfully redirects to homepage
TC002	Login with invalid password	Display error message	Error message shown ("Invalid credentials")
TC003	Add valid food item to cart	Item appears in cart	Item appears and cart count updates
TC004	Add out-of-stock item	Show out of stock error	Not implemented (static data, no inventory check)
TC005	Payment with valid card	Show payment success message	Simulated success message displayed
TC006	Payment with expired card	Show payment failed message	Simulated error shown (form validation)
TC007	Submit feedback	Thank you message appears	Feedback form submitted, alert shown
TC008	Order with empty cart	Show “Cart is empty” error	Error shown: "Cart is empty"
TC009	Schedule an order	Confirmation and scheduled time	Simulated confirmation shown (via alert or DOM)

		appears	
TC010	Employee requests shift swap	Notification to manager appears	Not implemented in frontend (requires backend)

Design

Instructions: Week 6

Journal

- List the nouns from our requirements/analysis documentation.
    - Customer
    - Order
    - Menu
    - Employee
    - Manager
    - Pay Slip
    - Shift
    - Cart
    - Feedback
    - Invoice

- Which nouns potentially may represent a class in your design?
    - Customer
    - Order
    - Menu item
    - Employee
    - Manager
    - Shift
    - Payment
    - Schedule.

- Which nouns potentially may represent attributes/fields in your design? Also list the class each attribute/field would be a part of.
    - **Customer**: name, email, orderHistory

    - **Order**: orderID, items, status, total

    - **MenuItem**: itemID, name, price, stock

    - **Employee**: id, name, shifts, holidays

    - **Feedback**: rating, comment

    - **Shift**: employeeID, date, time

    - **Payment**: amount, method, success

- Now that you have a list of possible classes, consider different design options (***lists of classes and attributes***) along with the pros and cons of each. We often do not come up with the best design on our first attempt. Also consider whether any needed classes are missing. These two design options should not be GUI vs. non-GUI; instead you need to include the classes and attributes for each design. Reminder: Each design must include at least two classes that define object types.

Design Option 1: Rich Domain Model

Classes & Attributes

Customer

- Customer
- CustomerId: ‘String`
- Name: ‘String`
- Email: ‘String`
- OrderHistory: ‘List<Order>`

Menu_Item

- ItemId: ‘String`
- Name: ‘String`
- Description: ‘String`
- Price: ‘Decimal`
- Stock: ‘Integer`

Order

- OrderId: ‘String`
- Customer: ‘Customer`
- Items: ‘Map <MenuItem,Integer>`
- Status: ‘OrderStatus`
- ScheduledTime: ‘DateTime?`
- TotalAmount: ‘Decimal`

Payment

- PaymentId: ‘String`
- Order: ‘Order`
- Amount: ‘Decimal`
- Method: ‘PaymentMethod`
- Status: ‘PaymentStatus`

Pros :

- - Strong encapsulation of data and business invariants.
- - Small, focused classes → easy to unit-test.
- - Natural extension path (e.g. add `Promotion` or `Notification`).

Cons :

- - More boilerplate: many classes and DTOs.
- - Longer initial setup time.

Design Option 2: Anemic Domain + Service Layer

Classes & Attributes

User

- UserId: ‘String’
- Name: ‘String’
- Email: ‘String’
- Roles: ‘Set<Role>’

MenuItem

- ItemId: ‘String’
- Name: ‘String’
- Description: ‘String’
- Price: ‘BigDecimal’
- Available: ‘Boolean’
- Category: ‘Category’

Cart

- CartId: ‘String’
- UserId: ‘String’
- Items: ‘Map<itemId: String, quantity: Int>’
- CreatedAt: ‘DateTime’

Order

- OrderId:‘String’
- CartId: ‘String’
- Status: ‘OrderStatus’
- PlacedAt: ‘DateTime’
- ScheduledFor: ‘DateTime’
- TotalAmount: ‘BigDecima’l

Payment

- PaymentId: ‘String’
- OrderId: ‘String’
- Method: ‘PaymentMethod’
- Amount: ‘BigDecimal’
- Status: ‘PaymentStatus’

Feedback

- FeedbackId: ‘String’
- OrderId: ‘’String’
- Rating: ‘Int’
- Comments: ‘String’
- SubmittedAt: ‘DateTime’

Services

UserService

- register(name, email, password): User
- login(email, password): SessionToken
- updateProfile(userId, ProfileData): User

MenuService

- listMenu(category?): List<MenuItem>
- getItemDetails(itemId): MenuItem
- updateAvailability(itemId, available: Boolean): void

CartService

- createCart(userId): Cart
- addItem(cartId, itemId, qty): Cart
- removeItem(cartId, itemId): Cart
- viewCart(cartId): Cart

OrderService

- placeOrder(cartId, paymentDetails): Order
- scheduleOrder(orderId, dateTime): Order
- trackOrder(orderId): OrderStatus
- cancelOrder(orderId): boolean

PaymentService

- processPayment(orderId, paymentDetails): PaymentStatus
- refund(orderId): RefundReceipt

FeedbackService

- submitFeedback(orderId, rating, comments): Feedback
- getFeedbackForItem(itemId): List<Feedback>

Pros

- Quick to implement: minimal domain wiring, fewer classes means faster MVP.
- Centralized workflows: all "place order" logic lives in one service.
- Easier onboarding: newcomers see all business rules in the service layer.

Cons

- Anemic objects: domain classes hold data only; business rules aren't co-located.
- Service bloat: each service can balloon into a large "god class" as new features (refunds, promotions, loyalty) are added.
- Harder to enforce invariants: e.g., preventing negative quantities must be manually checked in every service method rather than in a Cart class.

- Which design do you plan to use? Explain why you have chosen this design.

  We chose the Rich Domain Model because it ensures strong encapsulation, modularity, and scalability. Although it requires more setup initially, it aligns well with long-term maintainability and testability.

- List the verbs from your requirements/analysis documentation.
  - Login
  - Browse menu
  - Place order

- Add to cart
- Checkout
- Make payment
- Leave feedback
- Schedule order
- Request holiday
- Manage shifts
- Upload pay slips
- Respond to feedback

- Which verbs potentially may represent a method in your design? Also list the class each method would be part of.

Class	**Methods**
Customer	login(), browseMenu(), placeOrder(), trackOrder(), leaveFeedback()
Order	calculateTotal(), generateInvoice(), scheduleOrder()
Cart	addItem(), removeItem(), getTotalItems()
Payment	makePayment(), validateCard()
Employee	login(), viewShifts(), requestHoliday(), viewPaySlip()
Manager	approveHoliday(), assignShifts(), uploadPaySlips()
MenuItem	checkAvailability()

Software Design

*The software design for the Ausmena Kebabs Website Enhancement Project follows a **Rich Domain Model** to ensure strong encapsulation, modularity, and scalability. The design includes classes that represent key entities (Customer, MenuItem, Order, Payment) and their interactions, implemented using a front-end architecture with HTML5, CSS3, and JavaScript. The system is structured to support future backend integration while maintaining a responsive, user-friendly interface.*

Classes and Attributes

The following classes are defined based on the requirements and analysis:

1. ***Customer***
    - ***Attributes**:*
        - *CustomerId: String – Unique identifier for the customer.*
        - *Name: String – Customer's full name.*
        - *Email: String – Customer's email for login and notifications.*
        - *OrderHistory: List<Order> – List of past orders for account holders.*
    - ***Description**: Represents customers who browse, order, and provide feedback.*
2. ***MenuItem***
    - ***Attributes**:*
        - *ItemId: String – Unique identifier for a menu item.*
        - *Name: String – Name of the food item (e.g., "Chicken Kebab").*
        - *Description: String – Description of the item.*
        - *Price: Decimal – Price of the item.*
        - *Stock: Integer – Available quantity in stock.*
    - ***Description**: Represents food items available for ordering.*
3. ***Order***
    - ***Attributes**:*
        - *OrderId: String – Unique identifier for the order.*
        - *Customer: Customer – Reference to the customer placing the order.*
        - *Items: Map<MenuItem, Integer> – Map of menu items and their quantities.*
        - *Status: OrderStatus – Enum (e.g., Preparing, OutForDelivery, Delivered).*
        - *ScheduledTime: DateTime? – Optional scheduled delivery time.*
        - *TotalAmount: Decimal – Total cost of the order.*
    - ***Description**: Manages the details of a customer's order, including items and status.*
4. ***Payment***
    - ***Attributes**:*
        - *PaymentId: String – Unique identifier for the payment.*
        - *Order: Order – Reference to the associated order.*
        - *Amount: Decimal – Payment amount.*
        - *Method: PaymentMethod – Enum (e.g., CreditCard, PayPal).*
        - *Status: PaymentStatus – Enum (e.g., Pending, Success, Failed).*

  - ***Description**: Handles payment processing and status.*

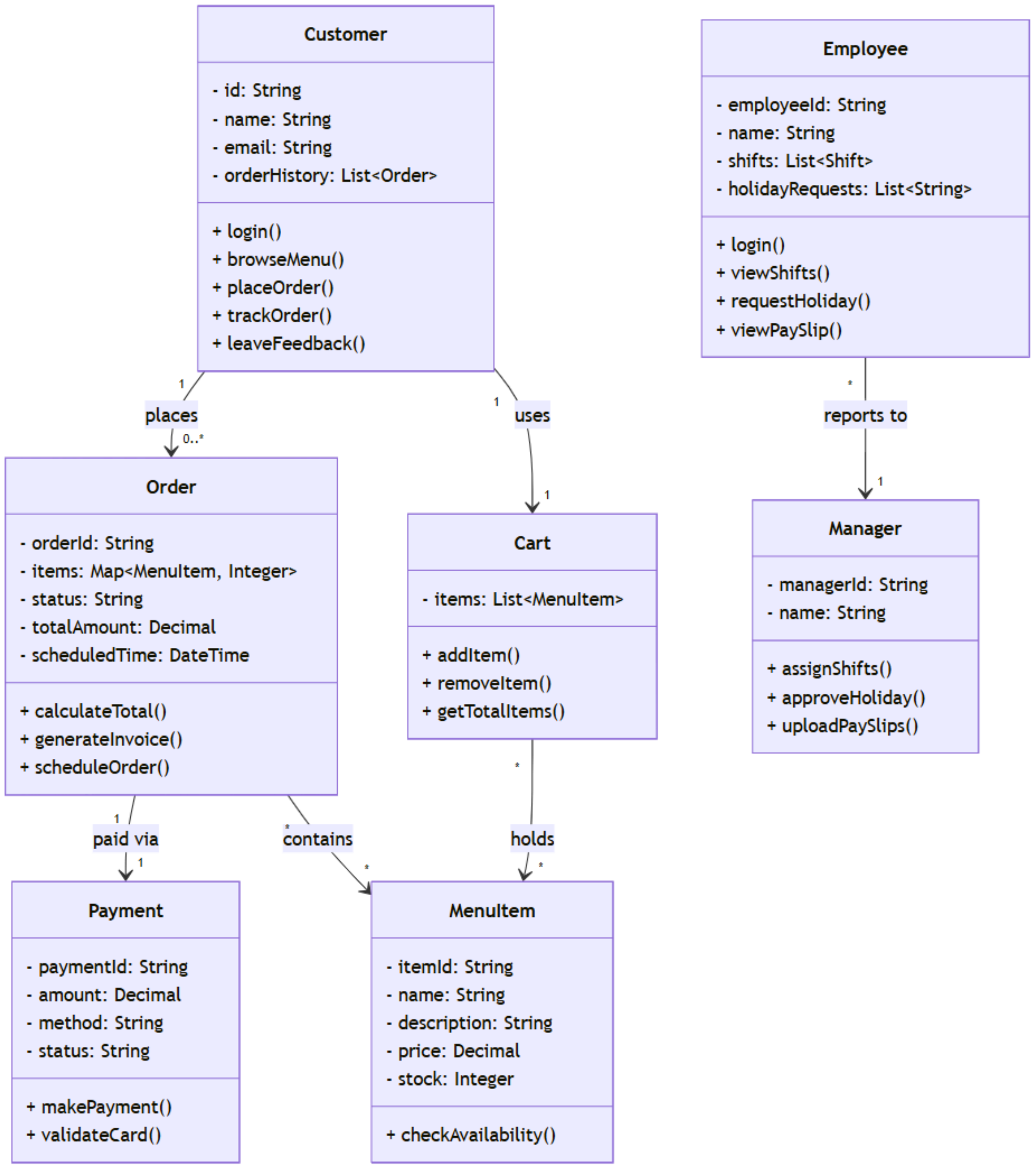
Methods

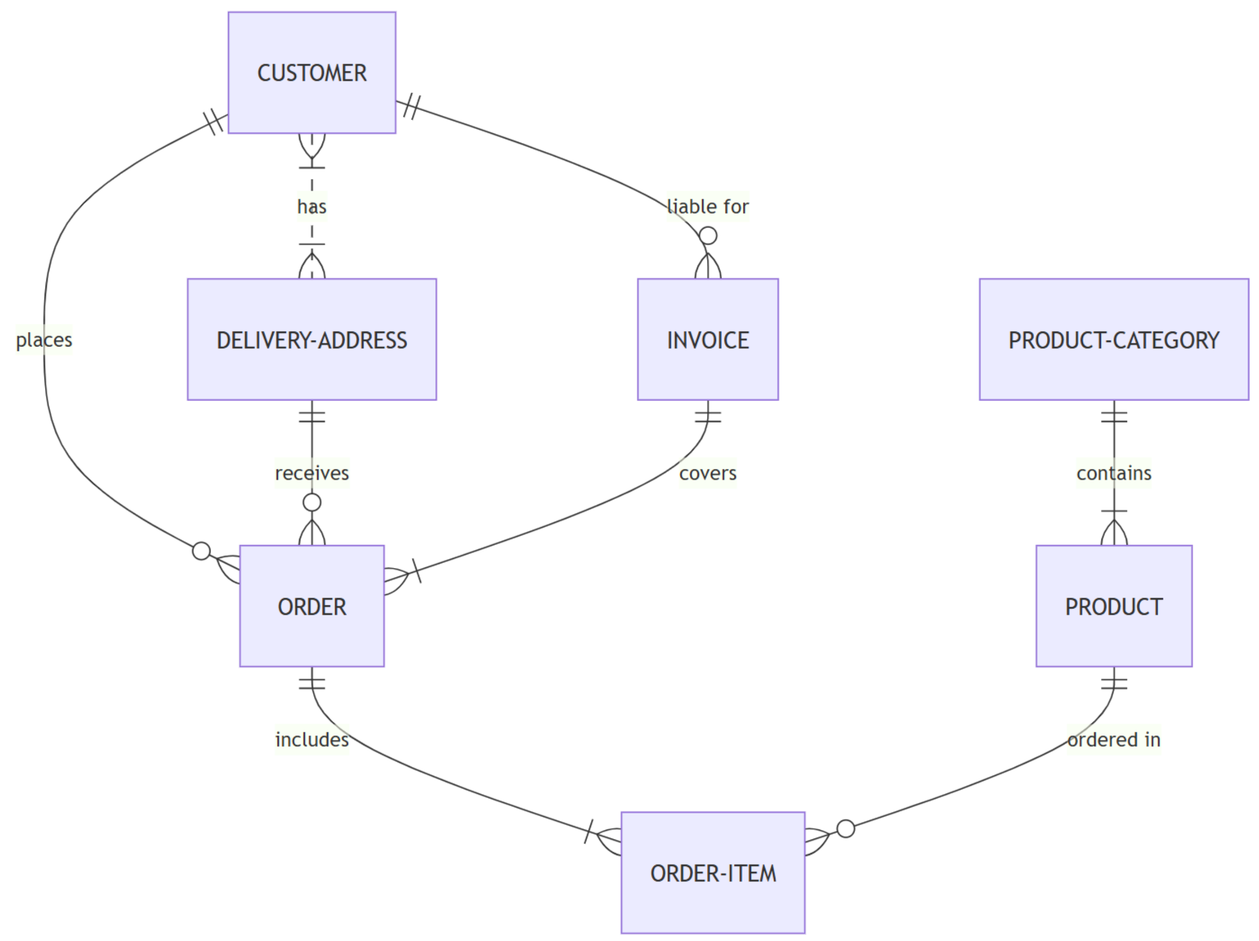
The following methods are derived from the verbs identified in the requirements (e.g., Login, Browse menu, Place order). Each method is associated with a class and includes a header in JavaScript-like pseudocode.

1. ***Customer***
   - *login(email: String, password: String): Boolean*
     - *Authenticates the customer and returns true if successful.*
   - *placeOrder(order: Order): Void*
     - *Submits an order to the system.*
   - *leaveFeedback(rating: Integer, comment: String): Void*
     - *Submits feedback and rating (1-5 stars) for an order.*
   - *viewOrderHistory(): List<Order>*
     - *Retrieves the customer's past orders.*
2. ***MenuItem***
   - *getDetails(): Object*
     - *Returns item details (name, description, price, stock).*
   - *updateStock(quantity: Integer): Void*
     - *Updates the stock level after an order.*
3. ***Order***
   - *addToCart(item: MenuItem, quantity: Integer): Void*
     - *Adds an item and quantity to the order's cart.*
   - *checkout(): Payment*
     - *Initiates payment processing and returns a Payment object.*
   - *trackStatus(): OrderStatus*
     - *Returns the current status of the order.*
   - *scheduleOrder(time: DateTime): Void*
     - *Sets a scheduled delivery time for the order.*
4. ***Payment***
   - *makePayment(amount: Decimal, method: PaymentMethod): PaymentStatus*
     - *Processes the payment and returns the status.*
   - *generateInvoice(): Object*
     - *Generates an invoice with order details.*

UML Class Diagram

The UML class diagram below illustrates the relationships between classes




CUSTOMER
has
liable for
places
DELIVERY-ADDRESS
INVOICE
PRODUCT-CATEGORY
receives
covers
contains
ORDER
PRODUCT
includes
ordered in
ORDER-ITEM

Implementation

Instructions: Week 8

Journal

The following prompts are meant to aid your thought process as you complete the implementation portion of this exercise. Please respond to each of the prompt below and feel free to add additional notes.

- What programming concepts from the course will you need to implement your design? Briefly explain how each will be used during implementation.
  - Concepts from the course used:

    Classes & Objects: Used for Customers, Orders, Menu Items.

    Encapsulation: Sensitive data like payment info protected.

    Arrays & Maps: To store cart items and shift schedules.

    DOM Manipulation (JS): For cart actions, menu rendering.

    Validation Logic: Check if input fields are filled, cards valid.

    Event Handling: Button clicks (add to cart, submit feedback).

Implementation Details

README (How users interact):

1 . **Customer**:

   - Visit website → Browse menu → Add to cart
   - Checkout → Enter payment details (simulated) → View receipt

2. **Employee**:

   - Login (mocked form) → View shifts → Request time off

3. **Manager**:

   - Login → Assign shifts → Upload pay slips (admin panel simulated)

Testing

Instructions: Week 10

Journal

The following prompts are meant to aid your thought process as you complete the testing portion of this exercise. Please respond to each of the prompts below and feel free to add additional notes.

- Have you changed any requirements since you completed the black box test plan? If so, list changes below and update your black-box test plan appropriately.
  - No major requirement change — all test cases are consistent with plan.
- List the classes of your implementation. For each class, list equivalence classes, boundary values, and paths through code that you should test.

Classes & Their Tests

1. Customer

- **Equivalence Classes**:
  - Valid email/password
  - Invalid login
- **Tests**:
  - Login with correct credentials (TC001)
  - Failed login (TC002)

2. Order

- **Boundary Values**:
  - 0 items in cart (TC008)
  - Max quantity/item limit
- **Tests**:
  - Order with valid cart (TC003)
  - Order with empty cart (TC008)

3. Payment

- **Equivalence Classes**:
  - Valid card
  - Invalid/expired card
- **Tests**:
  - Payment success (TC005)
  - Payment fail (TC006)

4. MenuItem

- **Test**:
    - Add out-of-stock item (TC004)

5. Employee

- **Test**:
    - Request holiday (manual test)
    - View pay slip

Testing Details

Test Program	What it Tests
test_login.js	Valid/invalid login
test_cart.js	Add/remove items, empty cart behavior
test_payment.js	Payment with various card states
test_feedback.js	Submitting feedback
test_schedule_order.js	Order scheduling Functionality

Presentation

Instructions:Week 12

Preparation

The following prompts are meant to aid your thought process as you complete the presentation portion of this exercise. It is recommended that you examine the previous sections of the journal and your reflections as you work on the presentation as it is likely that you have already answered some of the following prompts elsewhere. Please respond to each of the prompts below and feel free to add additional notes.

- Give a brief description of your final project
  - Ausmena Kebab Ordering Website" lets customers browse food items, add them to cart, checkout online, and download a receipt instantly. Employees can view shifts and request holidays, while managers control shifts, holidays, and pay slips.
- Describe your requirement assumptions/additions.
  - Guest checkout with card-only payments
  - Manager manually manages shifts and approvals
  - Holiday approval automated by hours worked logic
  - Cart logic done without backend (simulation only)
- Describe your design options and decisions. How did you weigh the pros and cons of the different designs to make your decision?

  We chose a **Rich Domain Model**:

  Strong separation of concerns
  Easier future expansion
  Drawback: More initial setup and complex structure

- How did the extension affect your design?
  - Scheduled Orders (added scheduledTime in Order class)
  - Feedback/Rating system
  - Admin functionality
- Describe your tests (e.g., what you tested, equivalence classes).
  - Functional testing of all major flows
  - Covered boundary values and invalid inputs

    - Tested mobile responsiveness manually

- What lessons did you learn from the comprehensive exercise (i.e., programming concepts, software process)?

    - Agile helped divide work into sprints

    - DOM manipulation is vital in interactive websites

    - Team coordination is key (Git conflicts, UI merging)

    - Good testing prevents bugs later

- What functionalities are you going to demo

    Functionalities to Demo

The following functionalities will be demonstrated during the presentation to showcase the Ausmena Kebabs website's capabilities, aligning with the Agile development process and user requirements:

1. **Home Page Navigation**:
    - Demonstrate navigation through the homepage, showcasing links to Menu, Order Now, Cart, Contact, and About Us pages.
    - Highlight the minimalist navigation bar with animated buttons for a seamless user experience.
2. **Menu Browsing**:
    - Show the card-based layout of menu items with images, descriptions, and prices.
    - Demonstrate the "Add to Cart" functionality, where users select items and quantities.
3. **Cart Simulation**:
    - Display dynamic cart interactions, including adding and removing items.
    - Simulate the checkout process, showing the order placement logic and invoice generation (simulated receipt download).
4. **Contact Us and About Us Pages**:
    - Showcase static pages with responsive design, ensuring accessibility and visual appeal across devices.

- Who is going to speak about each portion of your presentation? (Recall: Each group will have ten minutes to present their work; minimum length of group presentation is seven minutes. Each student must present for at least two minutes of the presentation.)

The presentation will last 10 minutes, with each team member speaking for at least 2 minutes to meet the requirement. The following assignments ensure balanced participation:

Team Presentation Breakdown

- **Abdullah Ansari – 231ADB054**
  Topic: Introduction and Demo
  Content: Deliver the project introduction and run the live demo of the Ausmena Kebabs website, showcasing real-time software functionality with explanation of requirement fulfillment.

- **Rahul Singh Garhwal – 231ADB026**
  Topic: Project Overview and Requirements Gathering
  Content: Present the index, project motivation, and methodology. Explain the purpose of the website, problems solved (e.g., eliminating third-party dependency), user types, and how requirements were gathered (interviews, benchmarking).

- **Adkhamjon Ahrolhanov – 221ADB222**
  Topic: System Design and Implementation
  Content: Discuss requirement specifications and software design. Present the Rich Domain Model, key classes (e.g., Customer, Order), UML diagrams, and rationale behind design decisions.

- **Karthik Bharadwaj Poduru – 250ADB053**
  Topic: Testing Strategy and Results
  Content: Explain the testing approach (functional testing, test cases for cart/checkout), challenges faced, and key learnings from implementation and testing phases.

- **Shihan Peries – 221ADB129**
  Topic: Conclusion and Future Work
  Content: Wrap up the presentation by summarizing achievements and outlining future work such as backend integration and enhanced inventory features.