Comprehensive Exercise Report

Team DIP392-group_M.R.D.D.CJ of Section 001

Muhammad Ali Tursunmurodov 213AEB032, Danylo Karpov 220AIB025, Oleksandr Shushyn 220ADB055, Shasti Chandrakumar Jayalakshmi 221ADB015, Dat Phu Huynh 221ADB182, Riyad Isgandarov 221ADB170, Illia Kuzub 221ADB236.

NOTE: You will replace all placeholders that are given in <<>>

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.

    - Project is responsive web-based app for Bakht Estate, helping users see pricing info and book online appointments. After booking, the system sends automatic email confirmation.

        - The website must display a clear and updated service price list.
        - Users should be able to book appointments through the website.
        - The system must send automatic email confirmations upon booking.
        - The interface must be accessible via desktop and mobile devices.
        - Simple navigation with minimal interaction complexity.
        - Client-side (Bakht Estate) should be able to receive booking details.

- 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.

    - Q: Should clients be able to choose a time slot or just submit a request?
    - A: Clients should select from available time slots shown in the form.

    - Q: Is an admin panel required in this phase for the business side?
    - A: No, not mandatory now. Admin features can be handled manually.

    - Q: Should bookings be stored for later view or only emailed?
    - A: Bookings should be stored on the backend and also emailed.

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

    - FastAPI email-sending integration
    - Calendar input UX on Angular forms

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

    - General customers of Bakht Estate (non-technical users)
    - Bakht Estate business staff (passive recipients of booking emails)

- Describe how each user would interact with the software
    - Customers: Open the website → View price list → Select service → Pick date/time → Submit form → Receive confirmation email

    - Staff: Receive appointment request via email and update scheduling manually

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

    - Service price list display
    - Booking form (name, service, date/time, email)
    - Automatic email confirmation system
    - Mobile responsiveness
    - Data handling with backend security

- Other notes:

    - The project is intentionally kept simple to meet a 20–30 hour development timeline.
    - Email delivery must be tested across devices and email providers.
    - UX must be clear enough for older users, as the agency caters to a wide demographic.

Software Requirements

Bakht Real Estate Booking Platform is a web-based scheduling system made for real estate agency, Bakht Estate. The platform solves old manual booking methods, which use phone calls and emails. The new system shows the service price list and allows online booking through a simple, responsive interface. After booking, the system sends automatic confirmation emails to the client. Backend, built with Fast API, manages appointment data safely and keeps connection between Angular frontend and email service. Solution gives more customer-friendly experience, lowers admin work, and builds a modern business image.

Functional Requirements:

ID	Requirement Description
1	The system shall display a price list of services on the main webpage.
2	The system shall provide a booking form for users to schedule appointments.
3	The system allows users to select a service, date, and time for their booking.
4	The system shall validate all form inputs before submission.
5	The system shall send an automatic confirmation email to the user upon successful booking.
6	The system shall store booking information securely on the backend.
7	The system should support access from mobile and desktop browsers.
8	The system shall allow customization of available services and prices (manual at this stage).

Non-Functional Requirements:

ID	Requirement Description
NFR1	The system should provide a responsive UI for mobile and desktop devices.
NFR2	The backend shall be developed using Fast API and integrated with SMTP for email notifications.
NFR3	The system shall maintain a response time under 2 seconds for form submission and confirmation.
NFR4	The system shall be protected against common web vulnerabilities (e.g., input sanitization, HTTPS encryption).
NFR5	The system shall store data in a reliable format (e.g., SQLite or JSON) during development.

Example User Stories are:

1. As a customer, I want to view a list of available real estate services and prices so that I can choose the right option.

2. As a customer, I want to book an appointment online by selecting a service and time slot so that I don’t have to call or email.

3. As a customer, I want to receive a confirmation email immediately after booking so I feel assured that my request was successful.

4. As the business owner, I want to receive customer booking details via email so I can manage appointments manually.

Black-Box Testing

Instructions: Week 4

Journal

Remember: Black box tests should only be based on your requirements and should work independent of design.

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

- What does input for the software look like (e.g., what type of data, how many pieces of data)?

The primary input is the appointment booking form, which includes the following fields:

  - Name and Surname
  - Email address
  - Selected service
  - User's choice about buys or sell
  - Phone number

- What does the output for the software look like (e.g., what type of data, how many pieces of data)?

  - A confirmation message shown on the front
  - A confirmation email sent to the user containing the booking details
  - Backend data storage of the booking entry

- What equivalence classes can the input be broken into?

Input Field	Valid Class	Invalid Class
Name	Any non-empty alphabetic string	Empty, numeric, or special characters only
Email	Properly formatted email	Missing @, domain, or empty
Service	One of the predefined options	Not in list, empty
Date	Valid future date	Past date, invalid format, empty
Time	Valid time in working hours	Non-time format, outside of hours, empty

- What boundary values exist for the input?

  - Name: Minimum 1 character, maximum 100 characters.
  - Email: Must be at least in the form a@b.c ; max 254 characters.
  - Date: Must not be in the past. Minimum is the current date.
  - Time: Start of business (e.g., 09:00) and end of business.
  - Service selection: Must match the exact string from list.

- Are there other cases that must be tested to test all requirements?

  - Submitting the form with one or more fields empty.
  - Valid submission to ensure confirmation messages appear and email is sent.

    - Cross-browser compatibility for form submission.
    - Testing email deliverability to different providers

- Other notes:

    - Black-box tests check what system should do, without looking at code or how it’s built.
    - Test cases created using only functional needs, keeping tests clear and fair.
    - Form checks expected on both frontend and backend to make sure correct behaviour across all parts.

Black-box Test Cases

Use your notes from above to complete the black-box test plan section of the formal documentation by writing black box test cases (other than actual results since no program currently exists). Remember to test each equivalence class, boundary value, and requirement.

Test ID	Description	Expected Results	Actual Results
1	Submit booking form with valid name, email, service, and date	Booking accepted, confirmation email sent	Booking accepted, confirmation email received
2	Submit booking form with empty required fields	Form shows error messages, booking not accepted	Error messages displayed, booking not accepted
3	Enter email without "@" or domain (e.g., "useremail.com")	Error shown for email field	Email field highlighted with error message

Design

Instructions: Week 6

Journal

Remember: You still will not be writing code at this point in the process.

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

- List the nouns from your requirements/analysis documentation.
    - Client, service, booking, email, form, date, platform, appointment, backend, frontend, confirmation, system, interface
- Which nouns potentially represent a class in your design?
    - Client
    - Booking
    - Service
    - Email (or Email Service)
    - System (as main controller or manager class)
    - Appointment
- Which nouns potentially may represent attributes/fields in your design? Also list the class each attribute/field would be a part of.
    - Client: name, email
    - Booking: bookingID, client, service, date, time
    - Service: serviceID, serviceName, price
    - EmailService: recipient, subject, messageBody
    - Appointment: status, timestamp
- 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 classes needed 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.

  Option 1

    - Classes:
    - Client: name, email
    - Booking: bookingID, client, service, date, time
    - Service: serviceID, name, price

Pros:

- Clean separation of data

- Easier unit testing
- More flexible for future features (e.g., multiple services per booking)

Cons:

- Requires more backend logic to manage connections between classes
- Slightly more complex integration with frontend

Option 2

- Classes:
- Booking Manager handles all booking logic, stores all data
- Email Service sends confirmation
- Booking: name, email, service, date, time (flat structure)

Pros:

- Simpler to implement
- Easier for small projects or MVP
- Quick connection to frontend

Cons:

- Less scalable
- Harder to separate logic for reuse
- Potential data duplication

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

  We use client-server architecture with a RESTful API. The frontend (Angular) interacts with the backend (Fast API),

- List the verbs from your requirements/analysis documentation.

  - View
  - Submit
  - Select
  - Book
  - Send

- Which verbs potentially may represent a method in your design? Also list the class each method would be part of.
  - PriceController (Backend)
  - BookingController (Backend)
  - send_confirmation_email()

- Other notes:

  - Design keeps logic modular, which improves testability.

Software Design

Class	***Method***	***Field(s)***
BookingController	*submit_booking(data)*	*-*
EmailService	*send_confirmation_email(to)*	*email_server, template*
FormValidator	*validate_input(form_data)*	*-*
DatabaseHandler	*save_booking(data)*	*db_path, connection*

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.

    - Functions: Used to define routes and business logic.
    - Classes: Used for modular components like email and database.
    - Validation: Ensures data integrity.
    - APIs: Enables front-end-backend communication (FastAPI).

- Other notes:

    - Used Python typing for method signatures.
    - Applied try/except for error handling in backend.

Implementation Details

The backend of the Bakht Real Estate project was built using FastAPI – a modern, streamlined ControlPython framework for quickly and securely building APIs, which was capable of running independently of the backend code. Its job was to get the booking data from the frontend form, validate the form, save the data, and send a confirmation email to the user for a reservation. The backend was built with a simple structure implementing clean-code practices. It has a controller for handling API requests (BookingController), a one service for form-checking, a service for sending email, and a service for saving booking information. This allowed the overall application to be more testable and extensible. Up to the user on the frontend, the client-side then would send over an HTTP request which was also written in Angular, with a clean and mobile-friendly interface, observable with fast refreshes and no timing issues to demonstrate on any device. The overall system was created with simplicity and ease of use in mind, and quick and easy end-user experience that met the clients request of a modern booking process without excessive complexity.

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, we have not made any changes to the original requirements. All planned features including form validation, booking submission, and email confirmation were implemented as described

- List the classes of your implementation. For each class, list equivalence classes, boundary values, and paths through code that you should test.

    - BookingController
        - Valid/invalid form submissions
        - Empty fields, max input length
        - Success, input error, server error

- Other notes:
    - Manual testing was done using browser.
    - All API endpoints were tested under normal and edge cases.
    - Email delivery confirmed via test Gmail account.

Testing Details

Test Name	Description	Expected Result	Actual Result
`test_valid_booking`	Submits valid booking data (all fields correct)	Booking accepted, email sent	Pass
`test_empty_fields`	Leaves one or more form fields empty	Form rejected, error shown	Pass
`test_invalid_email`	Submits invalid email format (e.g., "user@")	Form rejected, validation error	Pass

- All black-box tests were executed using both API tools and browser UI.
- Emails were confirmed via inbox.
- Form behavior was consistent across Chrome, Firefox, and mobile.

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
    - Our project is a web-based booking system for Bakht Real Estate that lets users view service prices and schedule appointments online. It replaces the outdated phone/email process with a fast and user-friendly interface.
- Describe your requirement assumptions/additions.
    - No admin panel was required at this stage.
    - Email confirmation is mandatory for every booking.
- Describe your design options and decision. How did you weigh the pros and cons of the different designs to make your decision?
    - We chose a client-server architecture using Angular (frontend) and FastAPI (backend) and this gave us a clean separation of logic and made integration simple
- How did the extension affect your design?
    - No big features were added, but the system was made ready for future growth. New tools like online payments or admin dashboard can be added later without changing main code.
- Describe your tests (e.g., what you tested, equivalence classes).
    - We tested form input, email sending, and data saving. Test cases covered correct and incorrect inputs, edge values, and errors (like email server not working).
- What lessons did you learn from the comprehensive exercise (i.e., programming concepts, software process)?
    - How to connect frontend and backend using REST APIs
    - How to use FastAPI in practice and send emails with SMTP
    - Why input validation, user messages, and clean code structure are important
    - How to plan real-world projects, work in a team, and write clear documentation
- What functionalities are you going to demo?
    - Viewing the service price list

    - Submitting a booking form
    - Receiving confirmation email
    - Booking storage (backend simulation)
    -

- 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.)

    - Illia Kuzub (Introduction & Project Overview)
    - Danylo Karpov (Agile Methodology & Requirements Analysis)
    - Shasti Chandrakumar Jayalakshmi (Requirements & Architecture)
    - Oleksandr Shushyn (Frontend Design & UI/UX)
    - Dat Phu Huynh (Backend Integration & Testing)
    - Riyad Isgandarov (Challenges & Lessons Learned)
    - Muhammad Ali Tursunmurodov (Demo, Conclusion & Future Work)

- Other notes:

    - Slides will be organized by project phase (Problem → Design → Demo → Lessons). Screenshots, short video clips, and live form demo will be included.