جامعة الأمير سلطان
PRINCE SULTAN
UNIVERSITY

The Consultant! Website

Department of Computer & Information Sciences

SE322 Project deliverable 1

Oct 7th, 2023

Instructor: Dr. Reem Alsuhaibani

Section: 1163

Prepared By:

Sarah AlJurbua 220410528

Fatemah Algarni 217410226

Nour Mohammed 220410494

Sara AlMutabagani 219410047

Yara Almuslimany 221410310

Table of Content

1. Introduction

A groundbreaking platform poised to transform the realm of consultancy. This innovative online solution is designed to streamline and expedite the process of conducting consultancy meetings over the Internet. With a relentless focus on simplicity, speed, and efficiency, "The Consultant" offers a user-friendly interface and robust architecture to ensure that online meetings for consultancy services are both accessible and hassle-free. In this document, we delve into the core features and design principles that make "The Consultant" an industry game-changer, shedding light on its potential to redefine the consultancy experience for professionals and clients alike.

1.1. Purpose

The primary purpose of "The Consultant" website is to facilitate and enhance the consultancy experience by providing a seamless and efficient platform for online meetings. Our main objectives include offering easy accessibility, fostering clear communication, and ensuring reliability for both consultants and clients. By prioritizing user-friendliness and speed, we aim to empower professionals to connect and collaborate effortlessly, ultimately driving the success of consultancy services in the digital age.

1.2. Scope

The scope of "The Consultant" website encompasses a comprehensive range of features and functionalities dedicated to optimizing the online consultancy experience. This platform will provide a user-friendly interface for scheduling and conducting virtual meetings, ensuring compatibility with various devices and browsers. It will support secure data transmission and storage, maintaining client confidentiality. "The Consultant" will also incorporate communication tools like chat and video conferencing to facilitate real-time interactions between consultants and clients. The website's architecture will also be designed for scalability and maintainability to accommodate potential future enhancements and adaptations. This project's scope will encompass the development, testing, and deployment of the website, as well as ongoing maintenance to ensure its reliability and efficiency in the long term.

1.3. Definitions, Abbreviations, and Acronyms

Term/Abre./Acro.	Definition
Repository	Computer storage for maintaining data or software packages
Counselor	A person trained to give guidance on personal, social, or psychological problems

1.4. References

- *Logical View - UML. (n.d.). cplusoop.com.* https://www.cplusoop.com/uml/module2/logical-view-uml.php
- *4+1 Architectural View Model in Software. (n.d.). Medium.* https://medium.com/javarevisited/4-1-architectural-view-model-in-software-ec407bf27258
- *Identify Stakeholders. (n.d.). Community Tool Box.* https://ctb.ku.edu/en/table-of-contents/participation/encouraging-involvement/identify-stakeholders/main
- *Collaboration Diagram - Process View* https://app.diagrams.net/
- *Creately. (n.d.). App.creately.com. Retrieved October 6, 2023, from* https://app.creately.com/d/LWWUadkxkDo/edit

2. Design stakeholders and concerns

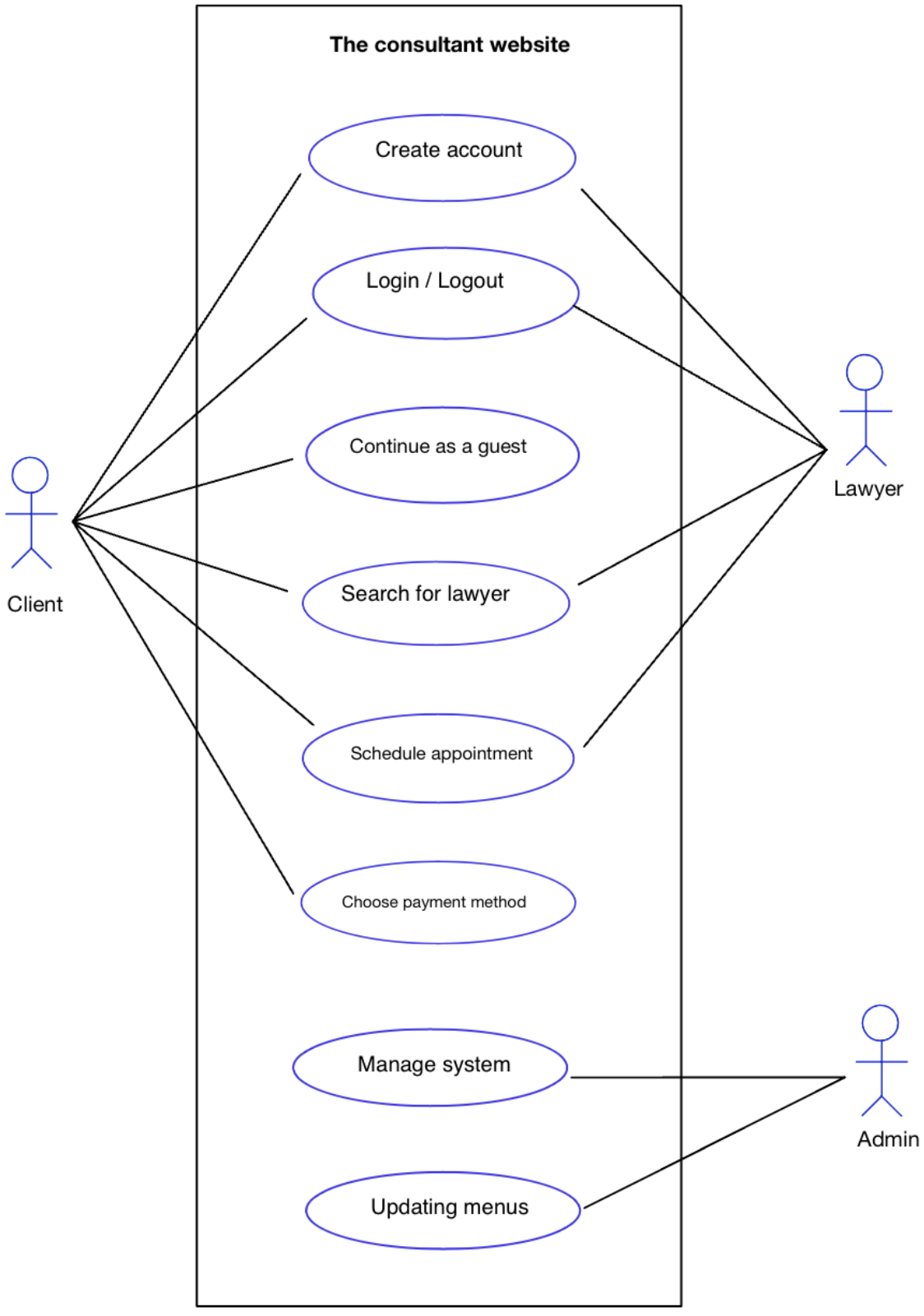
2.1. Design stakeholders and their concerns

- Clients want to chat with lawyers in a private environment.
- Lawyers want to schedule appointments with their clients in an efficient way.
- Admins want to maintain and update the system effectively.
- Designers want a well-implemented SRS document.
- Developers want well-structured architecture designs.
- The project manager needs to submit the required project within the time.

2.2. Design views and relationships to design concerns

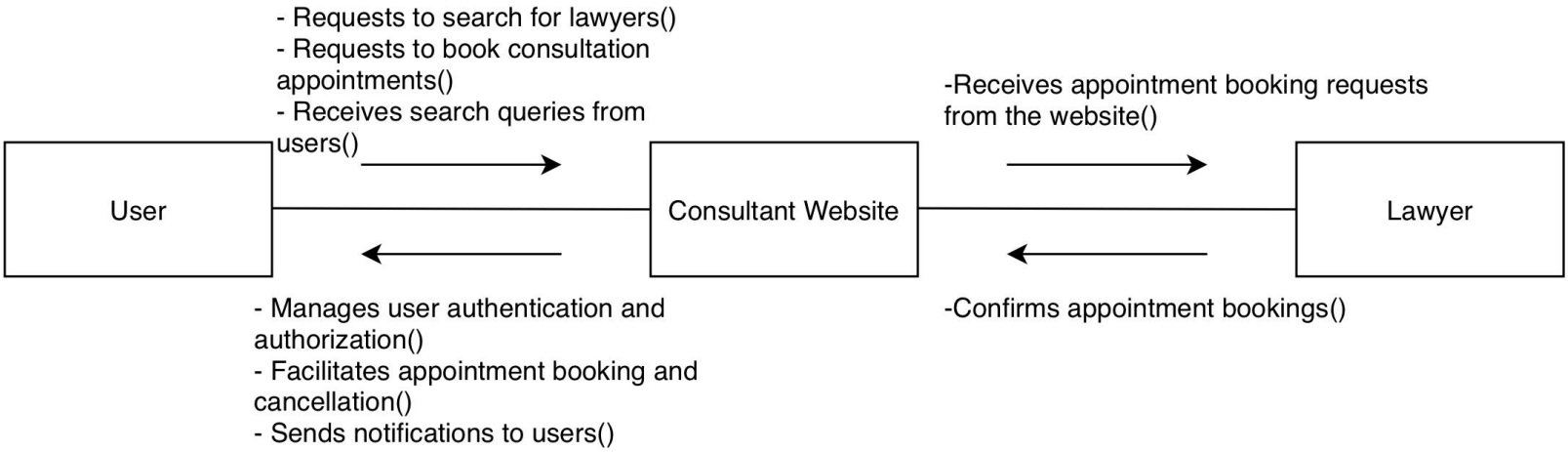
2.2.1. User view

2.2.1.1. Use Case Diagram

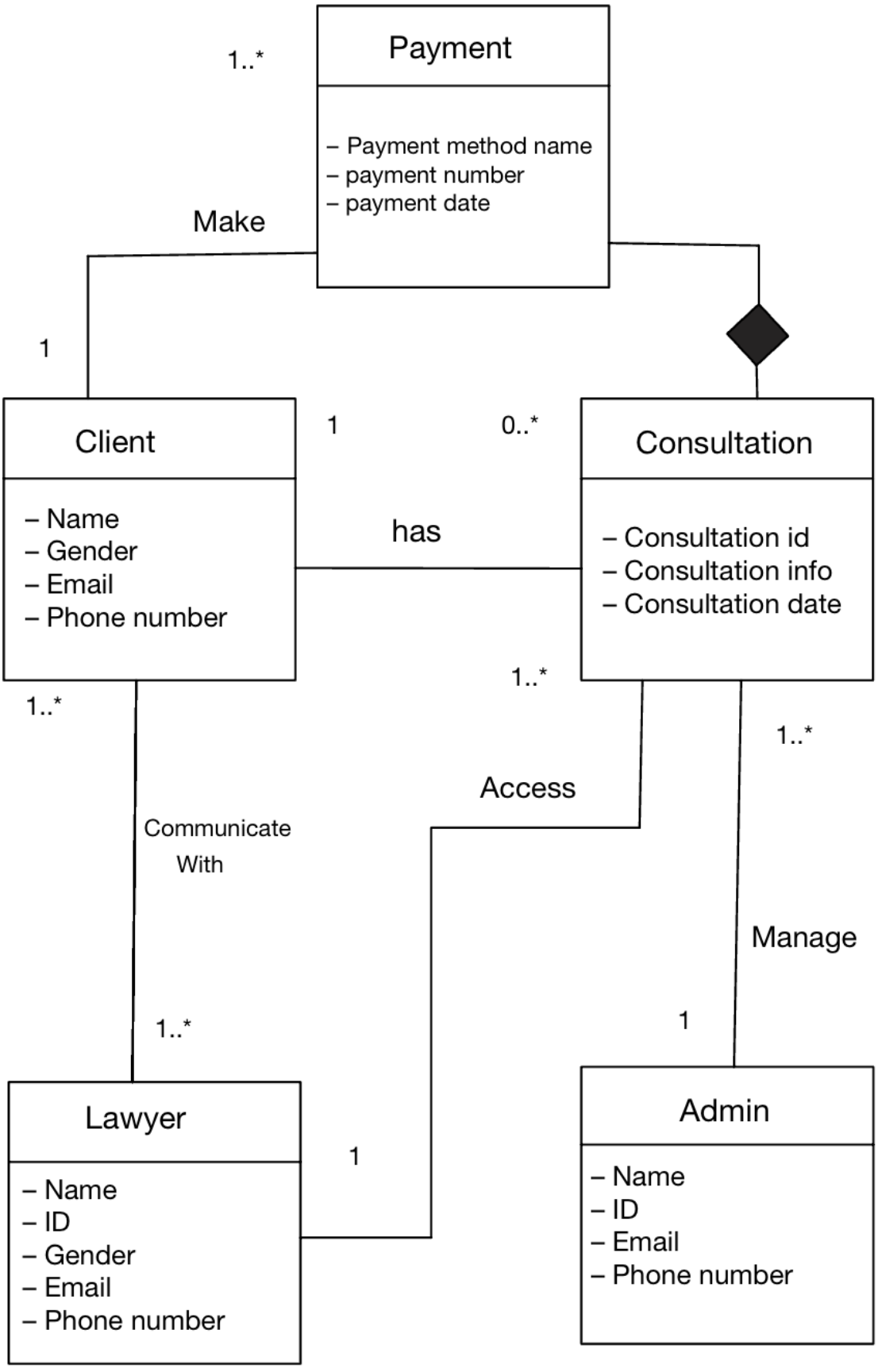


2.2.2. Logical View

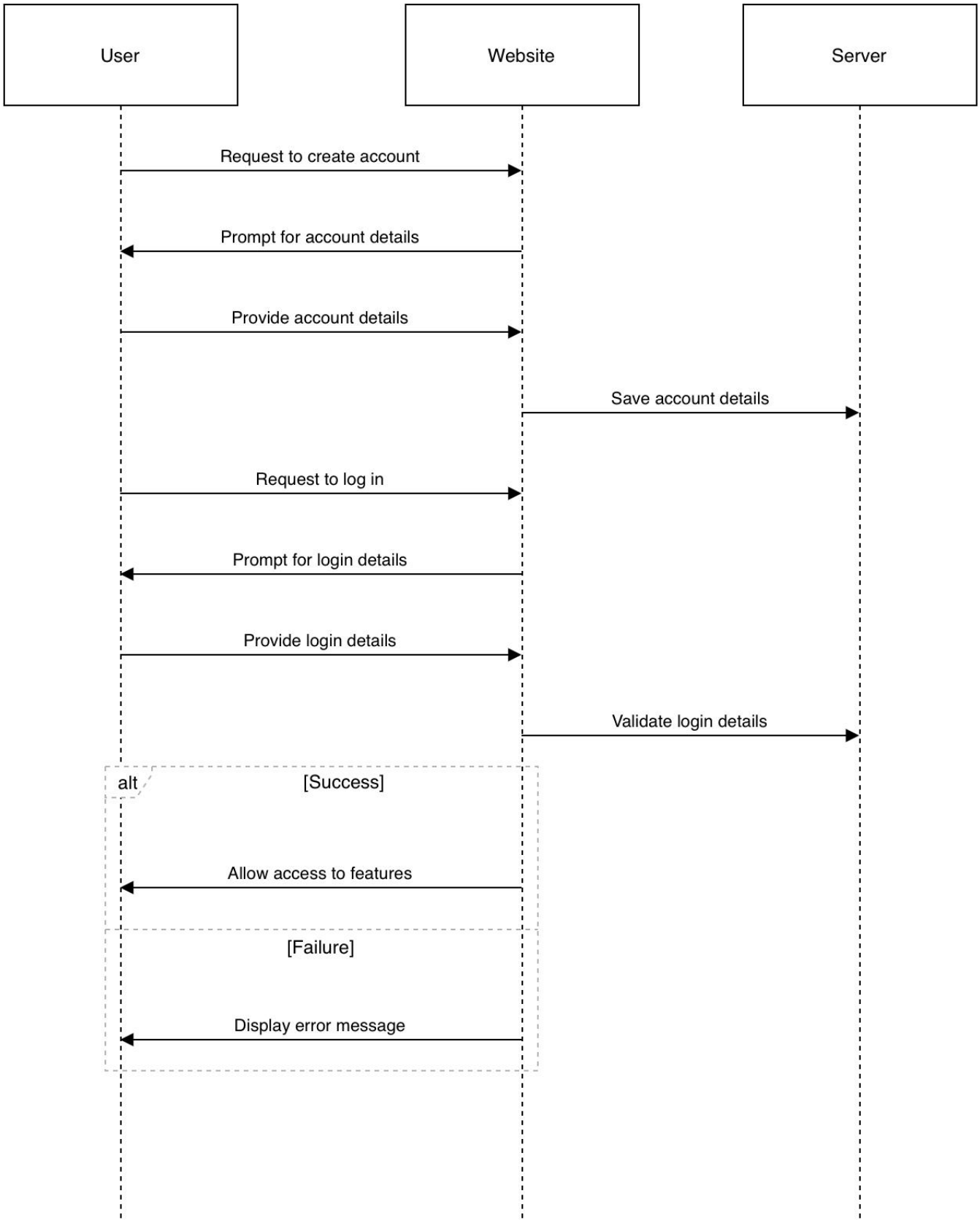
2.2.2.1. Collaboration Diagram



2.2.2.2. Class Diagram

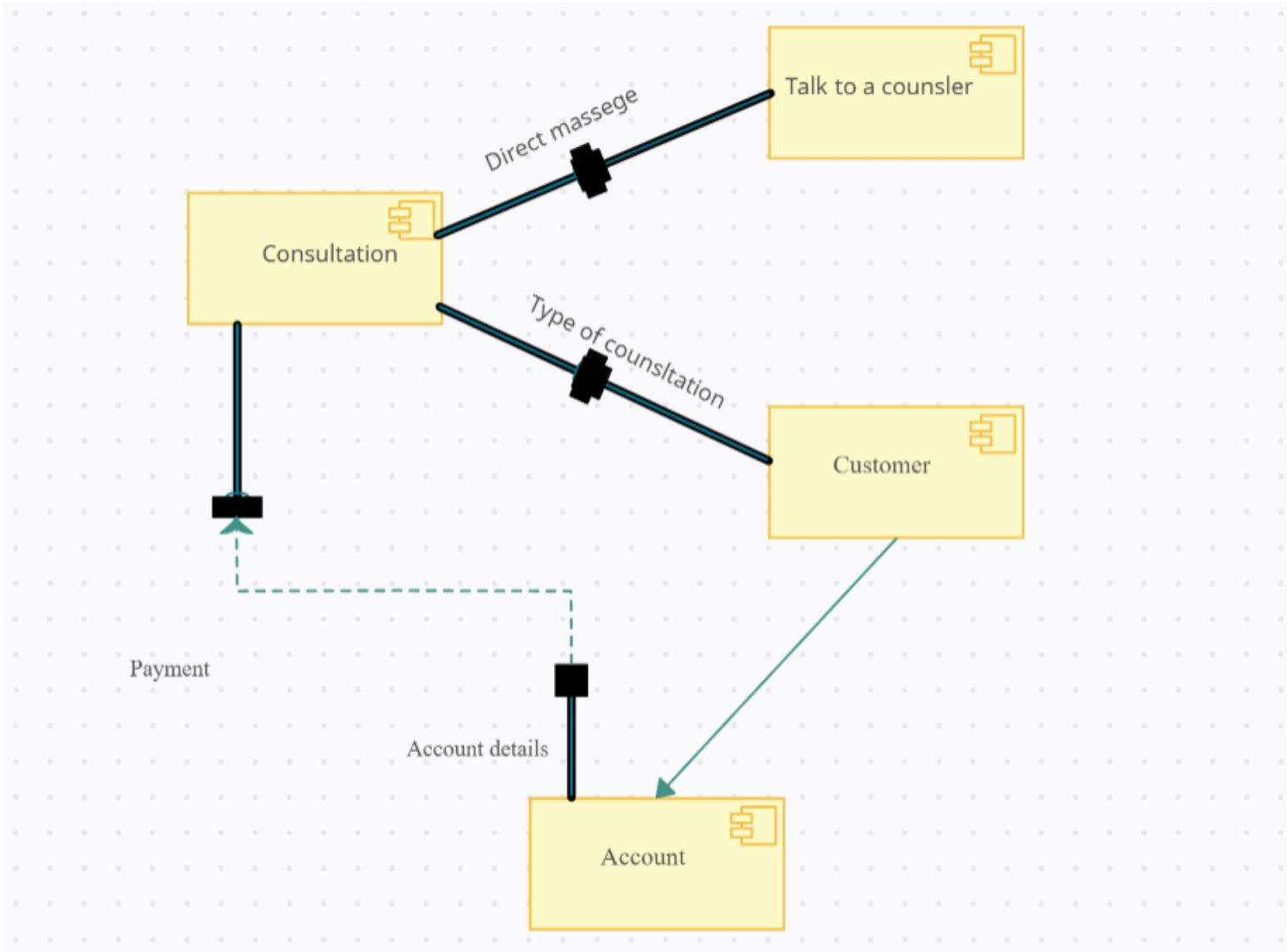


2.2.2.3. Sequence Diagram

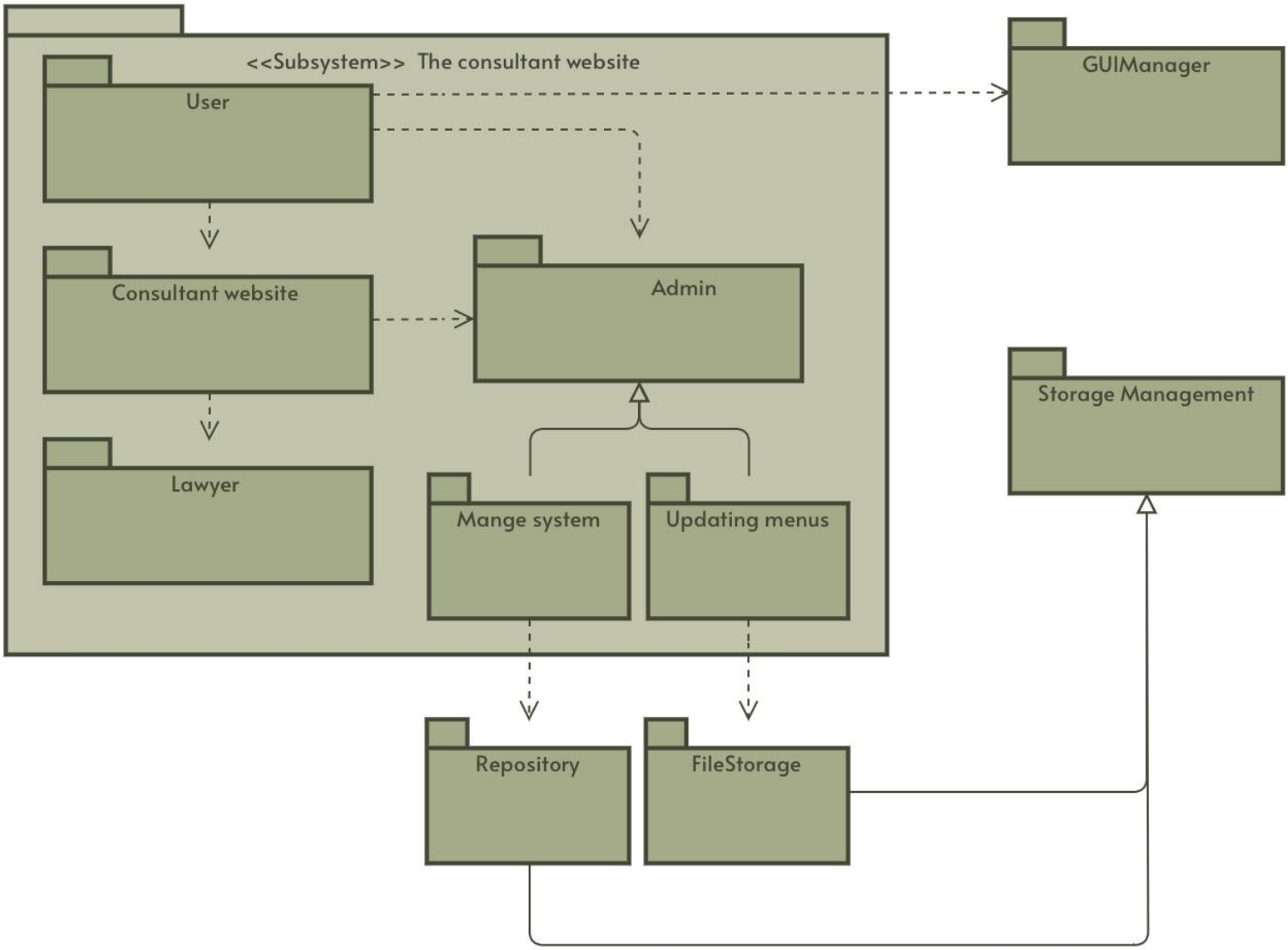


2.2.3. Implementation View

2.2.3.1. Component Diagram

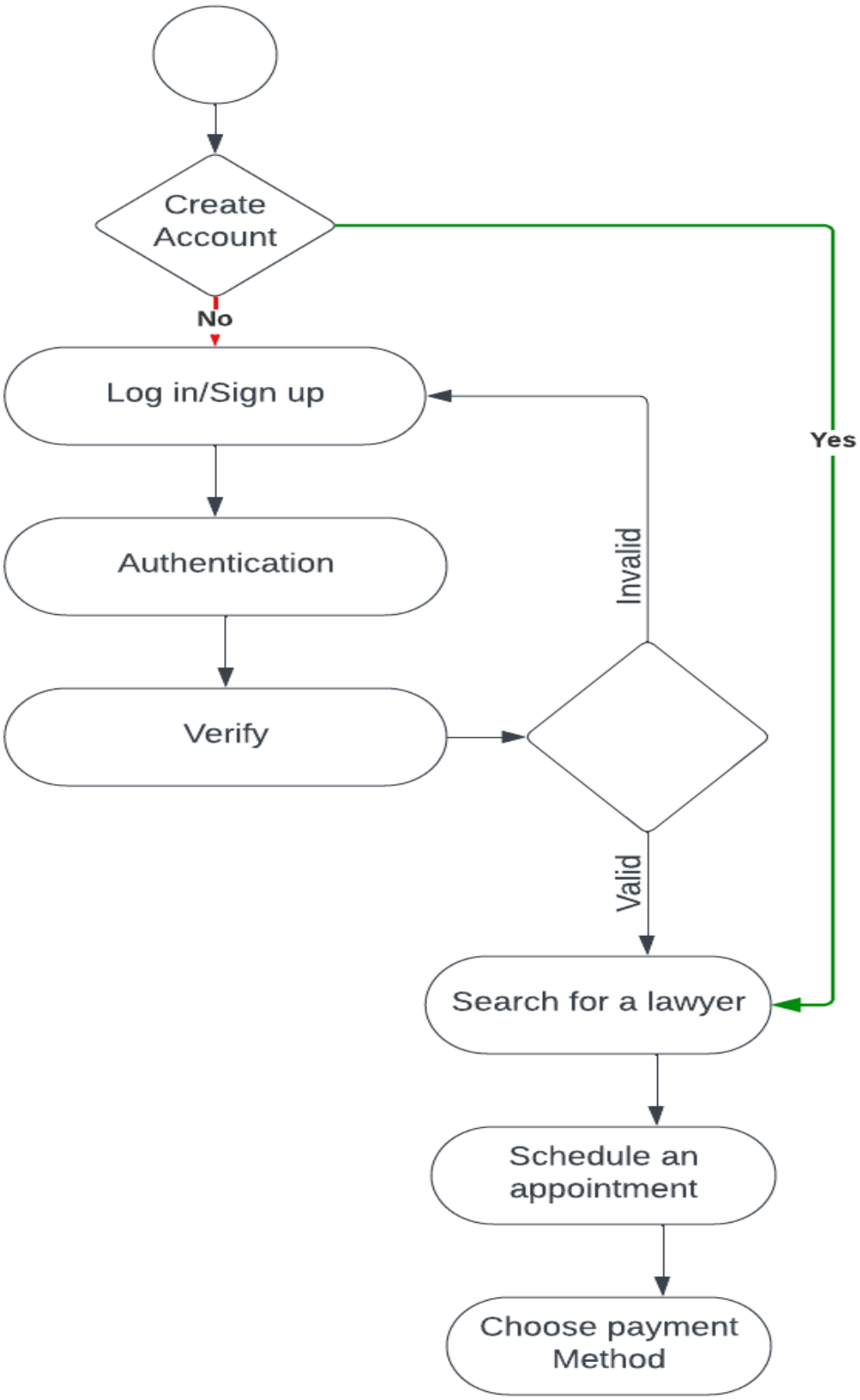


2.2.3.2. Package Diagram

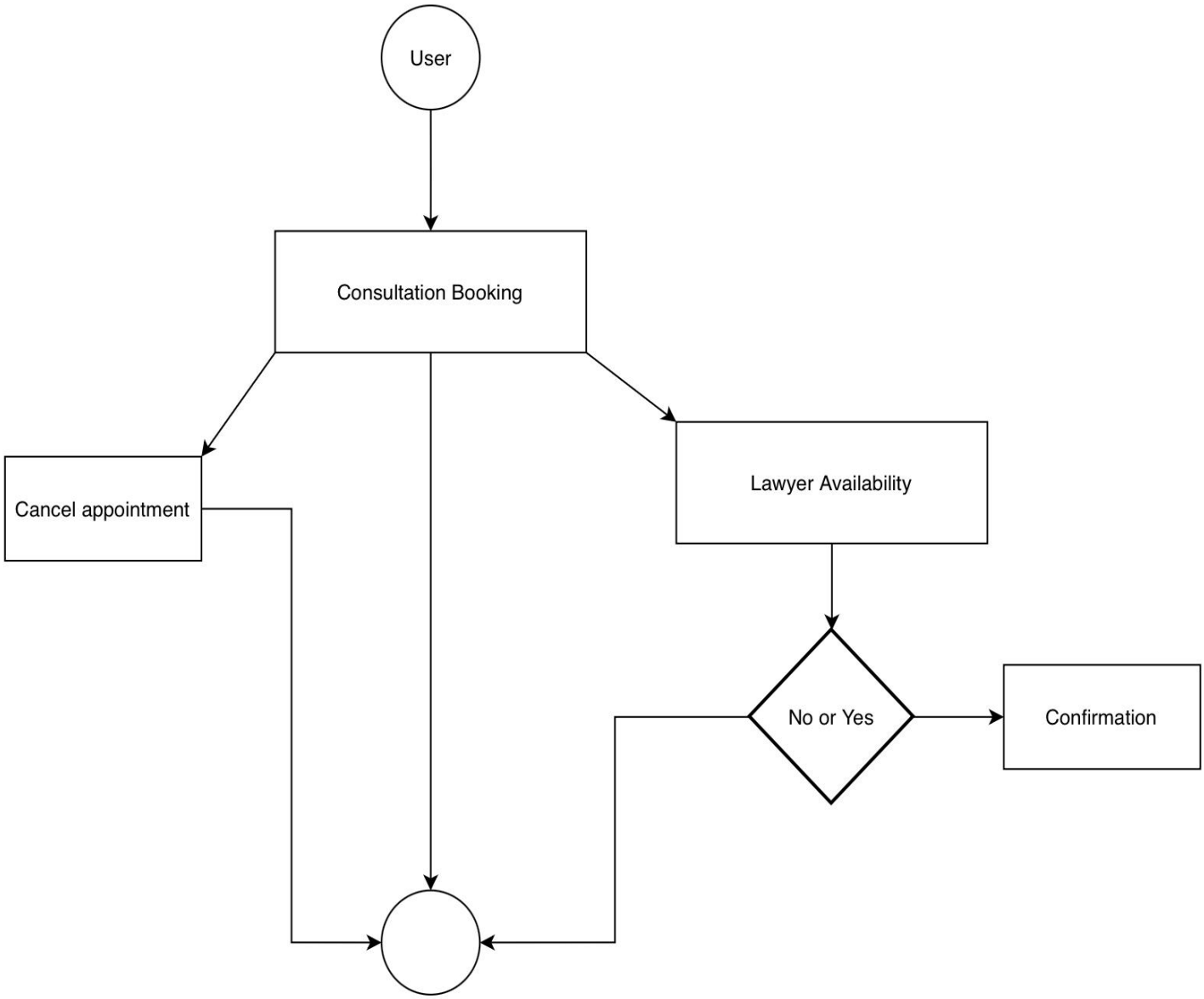


2.2.4. Process View

2.2.4.1. Activity Diagram

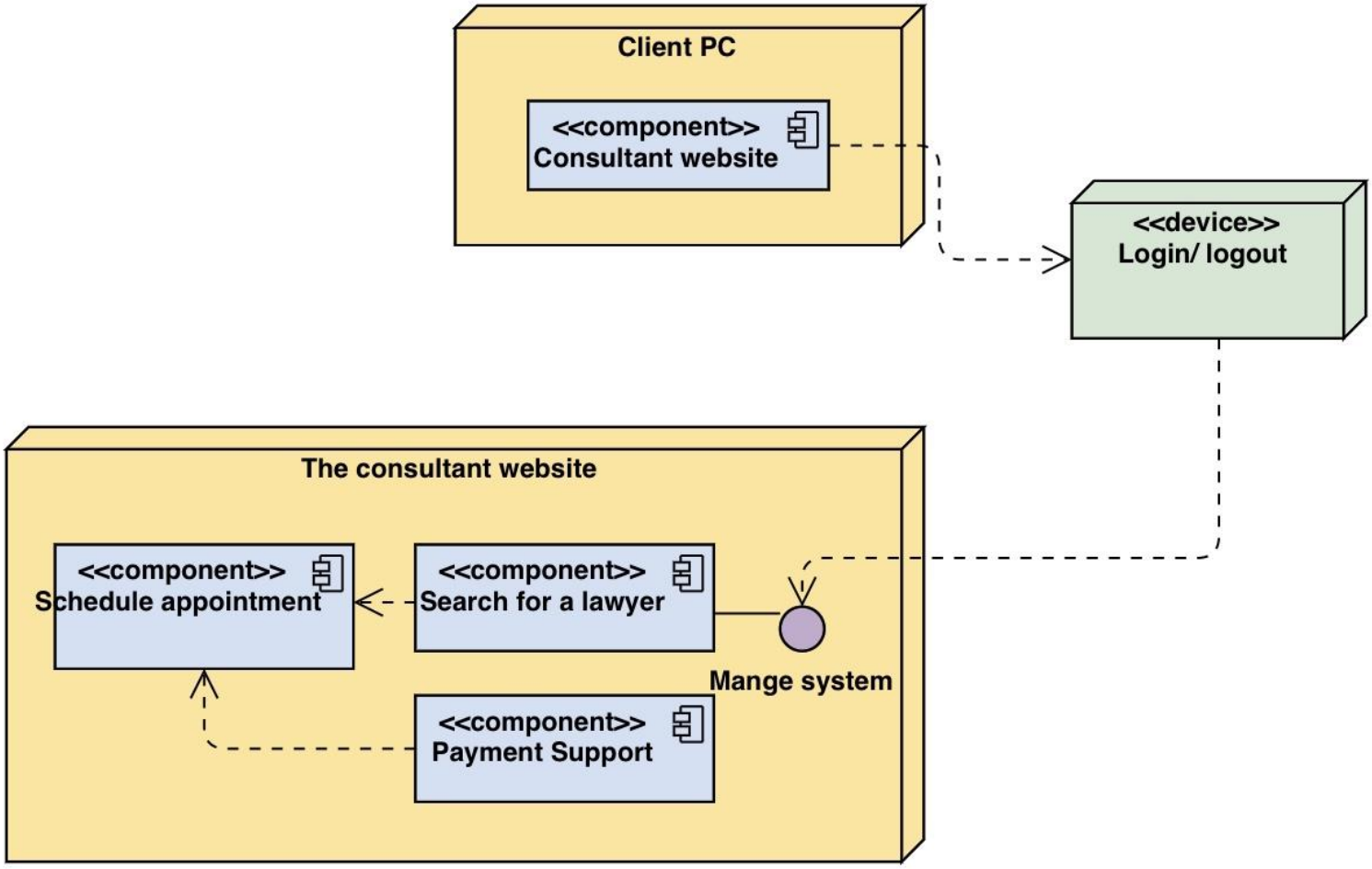


2.2.4.2. Interaction Overview Diagram



2.2.5. Physical View

2.2.5.1. Deployment Diagram



3. Additional

3.1. Individual Contribution

Name	Individual Contribution
Sarah Aljurbua	all #1, activity diagram
Fatemah AlGarni	Collaboration diagram, Sequence diagram, Interaction overview diagram
Yara Almuslimany	Component diagram
Nour Fatoom	Use case diagram, class diagram
Sara Almutabagani	Deployment diagram, package diagram

جامعة الأمير سلطان
PRINCE SULTAN
UNIVERSITY

The Consultant! Website

Department of Computer & Information Sciences

SE322 Project deliverable 2

Nov 11th, 2023

Instructor: Dr. Reem Alsuhaibani

Section: 1163

Prepared By:

Sarah AlJurbua 220410528

Fatemah Algarni 217410226

Nour Mohammed 220410494

Sara AlMutabagani 219410047

Yara Almuslimany 221410310

Table of Content

4. Software Architecture Description

4.1. Overview of Software Architecture

We need to develop a quality attribute scenario to choose the most suitable software architecture.

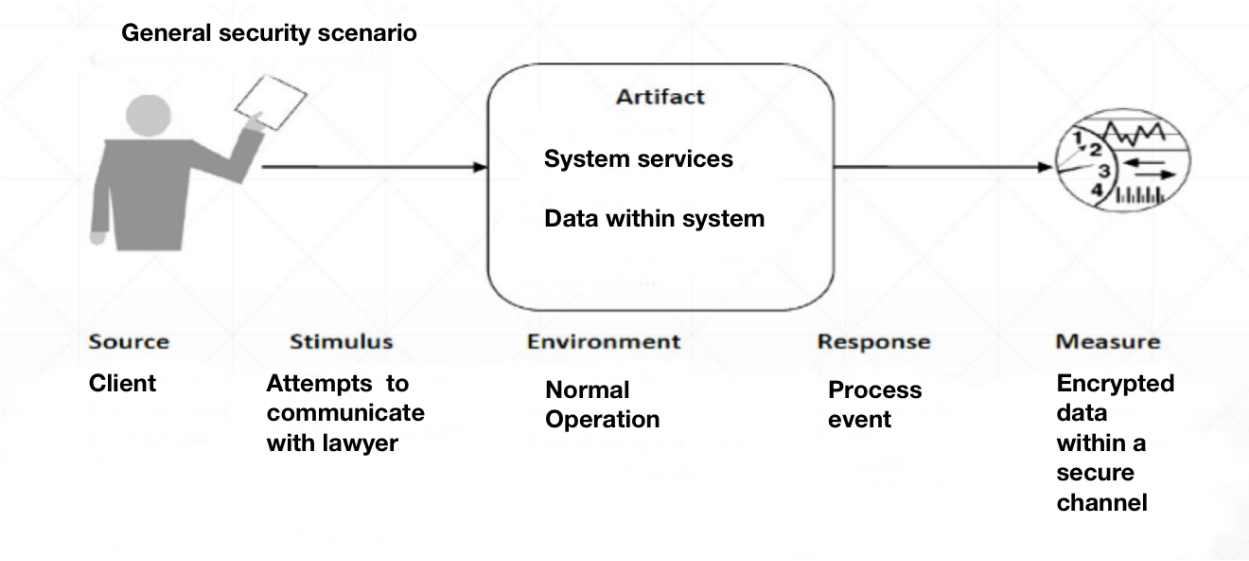
The 4 quality attributes that we picked are:

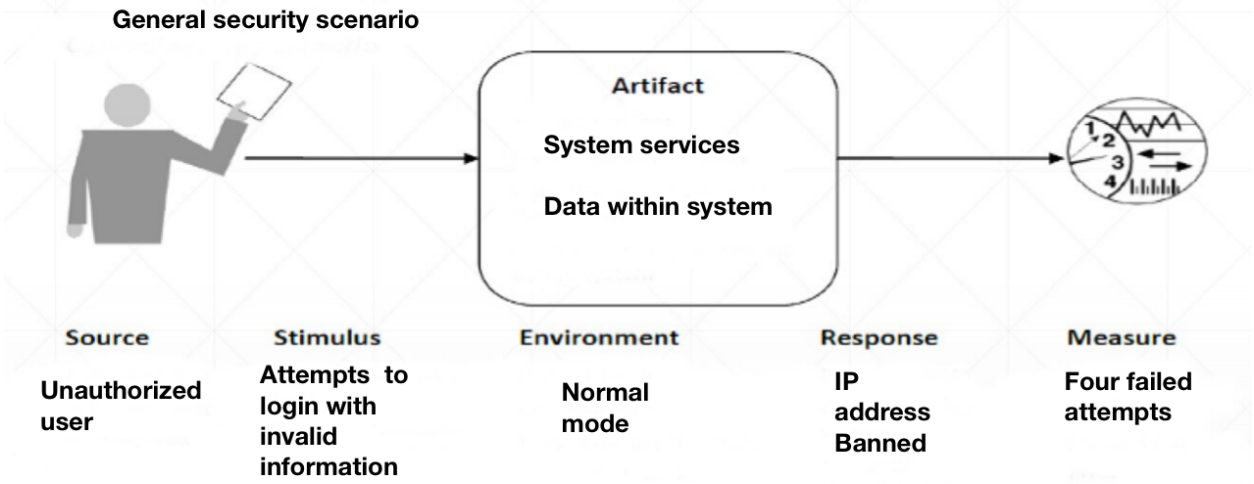
- Security.
- Availability.
- Modifiability.
- Usability.

4.1.1. Security

•Emphasizes secure data transmission and storage to protect client information, ensuring confidentiality.

•Implements encryption protocols and access controls to safeguard sensitive data.



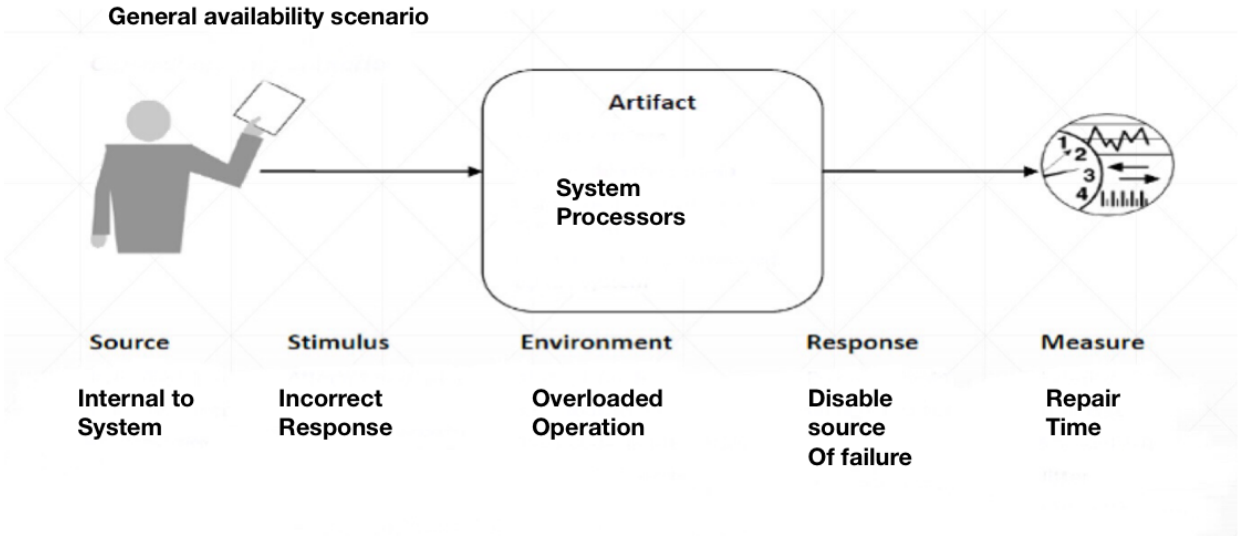


Security tactics:

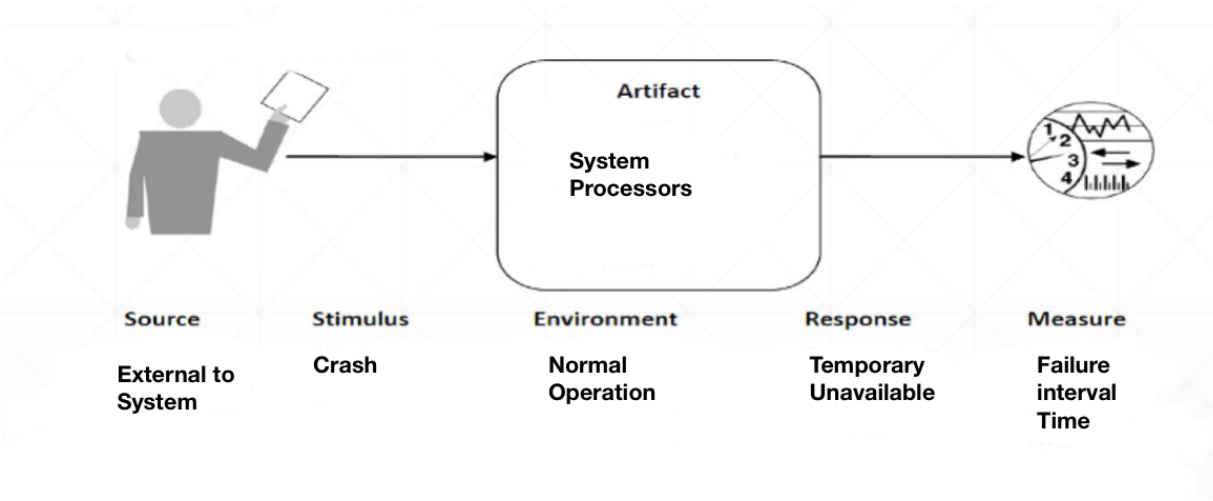
- Encrypt and Backup data in a secure database.
- Authenticate users. The system should not allow a valid login after 4 failed attempts.
- Ensure having secure hosting services on the website.

4.1.2. Availability

•Ensures high availability by employing redundant servers and failover mechanisms to prevent service disruptions.
•May use load balancing and scalable infrastructure to handle increased user demand without downtime.



General availability scenario

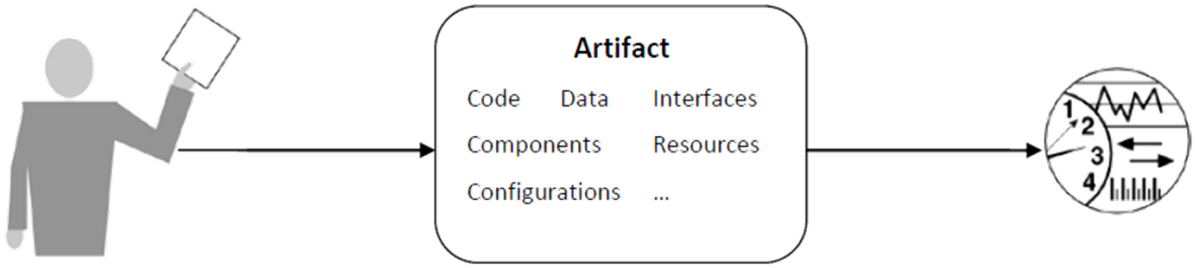


Availability tactics:

- Applying active and passive redundancy to the system
- Using detect, recover, and prevent failure approach.
- Temporarily removing a component of the system out of service.

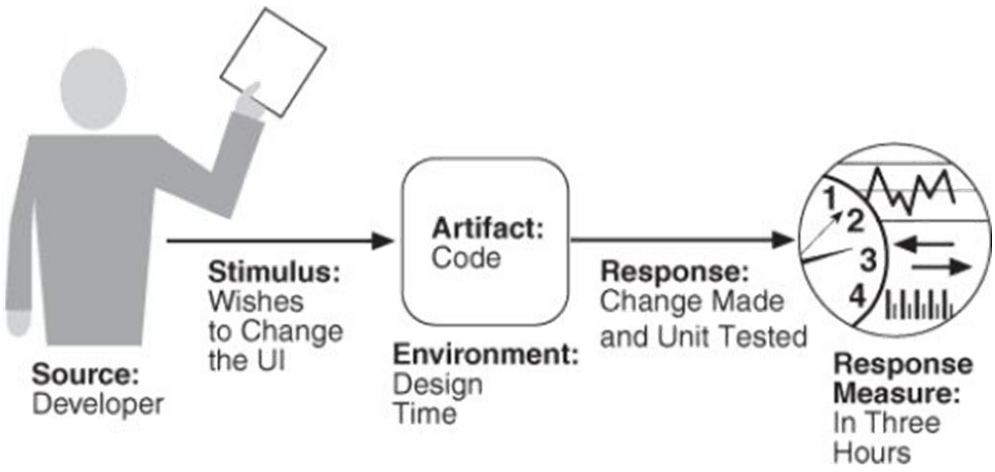
4.1.3. Modifiability

General modifiability scenario



Source	Stimulus	Environment	Response	Measure
End-user	Add / delete / modify functionality, quality attribute, capacity or technology	Runtime	Make modification	Cost in effort
Developer		Compile time	Test modification	Cost in money
System-administrator		Build time	Deploy modification	Cost in time
		Initiation time		Cost in number, size, complexity of affected artifacts
		Design time		

Sample modifiability scenario



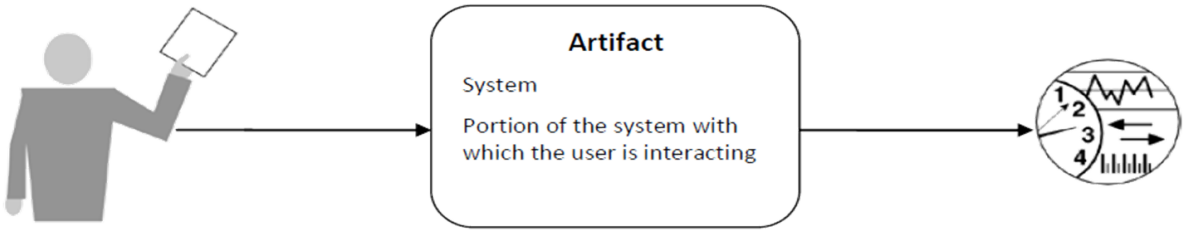
Modifiability Tactics:

- Manage code changes, with the development team.
- The architecture should be scalable.
- Develop an application that includes unit tests and integration tests.

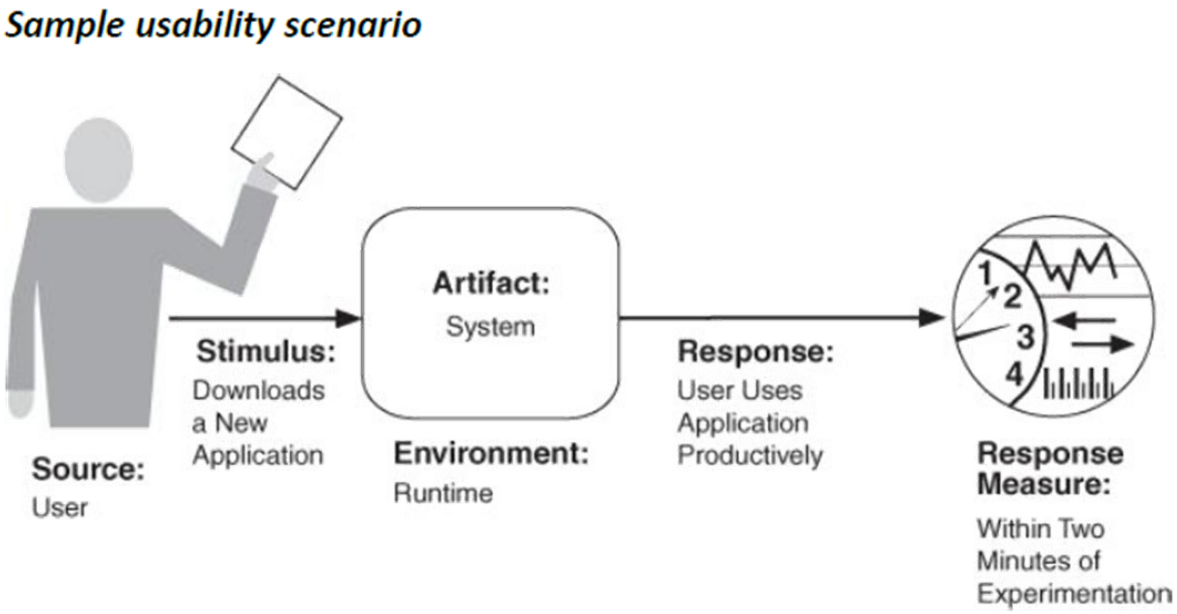
4.1.4. Usability

•The platform focuses on a user-friendly interface, emphasizing simplicity, ease of use, and intuitive navigation.

General usability scenario



Source	Stimulus	Environment	Response	Measure
End user (possibly special role)	Use the system efficiently	Runtime	Provide features needed	Task time
	Learn to use the system	Configuration time	Anticipate the user's needs	Number of errors
	Minimize impact of errors			Number of tasks accomplished
	Adapt the system			User satisfaction
	Configure the system			Gain of user knowledge



Usability Tactics:

- Users can cancel any booking from/by any consultant.
- Make user onboarding that guides new users to the feature and how to book an appointment.
- Users can edit settings by the provided options for personalization.

4.2. Overview of Software Architecture Via 4+1 Views

4.2.1. User View

•Provides an interface that allows users (consultants and clients) to schedule, join, and conduct consultancy meetings seamlessly.
•Offers a straightforward user experience, enabling effective communication and interaction during virtual meetings.

4.2.2. Logical View

•Illustrates the structural design, defining the functionalities and relationships between different components within the system.
•Describes the logic behind data flow, process coordination, and the interaction between various modules.

4.2.3. Implementation View

•Focuses on the implementation details, showcasing the software components, frameworks, and technologies used to build the platform.

•May include information on databases, APIs, communication protocols, etc.

4.2.4. Process View

•Explores the system from a process or workflow perspective, depicting how different processes work together to enable the consultancy meetings.
•Describes how tasks are carried out, including scheduling, data transmission, and user communication.

4.2.5. Physical View

•Describes the physical infrastructure of the system, detailing the hardware, servers, and network configuration that support the platform's operations.
•Discusses the deployment environment and how the system is distributed across different physical components.

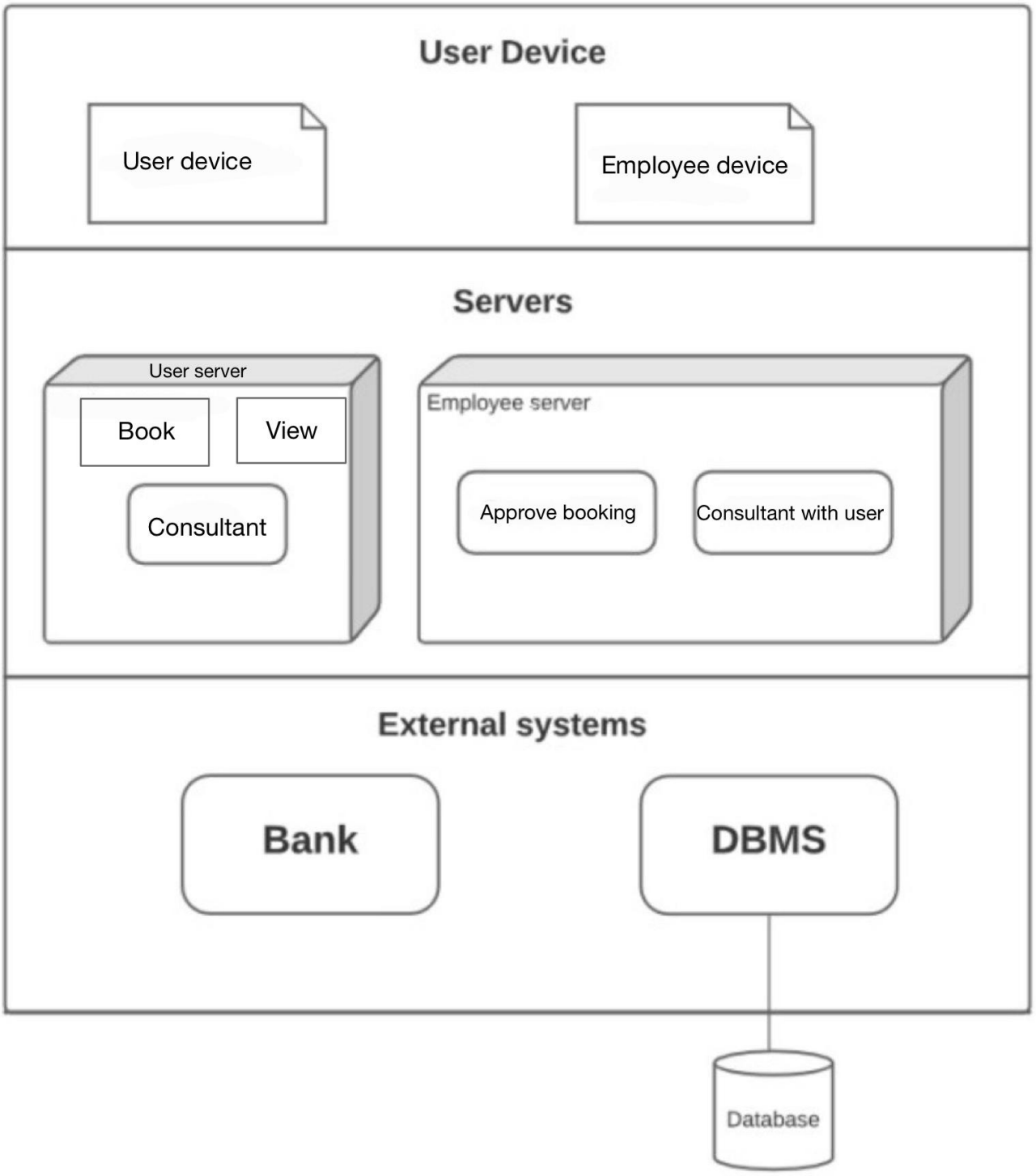
4.3. Architectural information that is relevant to multiple views

4.3.1. Overview

For this type of system, the most appropriate architectural style is the 3-tier architecture since the system is divided into three parts. Furthermore, each tier runs on its own infrastructure, each tier can be updated or scaled as needed without impacting the other tiers, and each tier can be developed simultaneously by a separate development team.

1- Presentation tier: deals with the user interface which handles the user and admin.
2- Logic tier: is in charge of all operations in the system.
3- Data management tier: this tier is responsible for retrieving and storing data.

4.3.2. Architectural Style Diagram



The Consultant! Website

Department of Computer & Information Sciences

SE322 Project deliverable 3

Dec 4th, 2023

Instructor: Dr. Reem Alsuhaibani

Section: 1163

Prepared By:

Sarah AlJurbua 220410528

Fatemah Algarni 217410226

Nour Mohammed 220410494

Sara AlMutabagani 219410047

Yara Almuslimany 221410310

Table of Contents

5. Design Principles

5.1. The Open-Closed Design Principle

- The Open-Closed design principle is a principle that states that the entities should be open for extension, but closed for modification.

- Reasons for implementing this principle in The Consultant website:

1- To add new extensions to the code without changing the core implementation.
2- To organize, simplify, and modify the code easily.
3- To increase the execution time of the code output.

// Creating person class that include the shared attributes and functions for client, lawyer, admin
Class person {

Private string name
Private string ID
Private string Email
Private String phone_number

Private void Create_account ( ) { }
Private void login ( ) { }
Private void logout ( ) { }

}

// Creating a user class that extends person class attributes and functions but adding a shared function between client and lawyer which is scheduleAppointment

Class user extends pearson{

Public void scheduleAppointment( ) { }

}

// Creating a lawyer class that includes its own attributes and functions.

Class lawyer extends user{

private string Gender

@ovverride
Public Void scheduleAppointment( ) { } // To override the scheduleAppointment in specific

Way for the lawyer.

}

// Creating an admin class that includes its own attributes and functions.

Class Admin extends person {

public void updateMenus ( ) { }
public void manage_consultantSystem ( ) { }

}

// Creating a client class that includes its own attributes and functions.

Class client extends user{

private string Gender

@ovverride
Public Void scheduleAppointment( ) { } // To override the scheduleAppointment in specific Way for the client.

Public void choose_PaymentMethod (Payment payment) { }

}

// Creating a payment class that includes its own attributes.

Class payment {

Private string method_name
Private int payment_number
Private string payment_date

}

// Creating a consultation class that includes its own attributes.

```
Class consultation {

Private string consultation _Info
Private int  Consultation_ID
Private string consultation _date

}

 // End of the OCP java code
```

5.2. The Interface Segregation Principle (ISP)

- Is one of the SOLID concepts of object-oriented design that Robert C. Martin came up with. It says that a class shouldn't have to implement interfaces that it doesn't need. To put it another way, a class should only have to implement methods that are needed for it to work.

- The main idea behind ISP is to avoid making big, single-piece interfaces that make classes that implement them provide implementations for methods they don't need. Interfaces should be specific and made to fit the needs of the classes that implement them instead.

- As an example of how to use the Interface Segregation Principle on "The Consultant" website, let's say we have different types of users, and each of them needs different online meeting features. We'll make systems that are perfect for these users' wants.

Let's make tools for the different jobs that play a part in online consulting:

```
// Interface for a consultant with features related to consultancy tasks
interface Consultant {
    void scheduleMeeting(String clientName, String meetingDateTime);
    void conductMeeting(String clientName);
}

// Interface for a client with features related to client-specific tasks
interface Client {
    void joinMeeting(String consultantName);
    void communicate(String message);
}

// Interface for the website's communication tools
```

```
interface CommunicationTools {
    void initiateChat(String user1, String user2);
    void startVideoConference(String user1, String user2);
}

// Class representing the Consultant on "The Consultant" website
class ProfessionalConsultant implements Consultant, CommunicationTools {
    // Implementation of Consultant interface methods

    @Override
    public void scheduleMeeting(String clientName, String meetingDateTime) {
        // Implementation for scheduling a meeting
    }

    @Override
    public void conductMeeting(String clientName) {
        // Implementation for conducting a meeting
    }

    // Implementation of CommunicationTools interface methods

    @Override
    public void initiateChat(String user1, String user2) {
        // Implementation for initiating a chat
    }

    @Override
    public void startVideoConference(String user1, String user2) {
        // Implementation for starting a video conference
    }
}

// Class representing the Client on "The Consultant" website
class ConsultingClient implements Client, CommunicationTools {
    // Implementation of Client interface methods

    @Override
    public void joinMeeting(String consultantName) {
        // Implementation for joining a meeting
    }
```

```
    @Override
    public void communicate(String message) {
        // Implementation for client communication
    }

    // Implementation of CommunicationTools interface methods

    @Override
    public void initiateChat(String user1, String user2) {
        // Implementation for initiating a chat
    }

    @Override
    public void startVideoConference(String user1, String user2) {
        // Implementation for starting a video conference
    }
}
```

5.3. The Single Responsibility Principle (SRP)

- The Single Responsibility Principle (SRP) is a fundamental principle in object-oriented design, emphasizing that a class should have only one reason to change. In the context of "The Consultant Website," we apply SRP to enhance the maintainability and flexibility of our codebase.

- In this example, we focus on the class responsible for managing consultant profiles - the ConsultantProfileManager.
- This class adheres to SRP by having a single responsibility: the storage and retrieval of consultant information. The ConsultantProfile class represents the structure of a consultant's profile, maintaining a separation of concerns and promoting a clean and modular design.

- This approach ensures that any changes related to the management of consultant profiles can be made within the ConsultantProfileManager class without affecting the structure or behavior of the ConsultantProfile class, aligning with the principles of SRP.

```
import java.util.HashMap;
import java.util.Map;

public class ConsultantProfileManager {
    private Map<String, ConsultantProfile> consultantProfiles;

    public ConsultantProfileManager() {
        this.consultantProfiles = new HashMap<>();
    }

    public void addConsultantProfile(String consultantId, ConsultantProfile profile) {
        consultantProfiles.put(consultantId, profile);

    }

    public ConsultantProfile getConsultantProfile(String consultantId) {
        return consultantProfiles.get(consultantId);

    }
}

public class ConsultantProfile {
    private String consultantId;
    private String name;
    private String expertise;
    // Other profile-related attributes

    public ConsultantProfile(String consultantId, String name, String expertise) {
        this.consultantId = consultantId;
        this.name = name;
        this.expertise = expertise;

    }


}
```

جامعة الأمير سلطان
PRINCE SULTAN
UNIVERSITY

The Consultant! Website

Department of Computer & Information Sciences

SE322 Project deliverable 4

Dec 10th, 2023

Instructor: Dr. Reem Alsuhaibani

Section: 1163

Prepared By:

Sarah AlJurbua 220410528

Fatemah Algarni 217410226

Nour Mohammed 220410494

Sara AlMutabagani 219410047

Yara Almuslimany 221410310

6. Design Patterns

6.1. Singleton Design Pattern

- Singleton pattern is a creational design that provides access to any instance from the code base.

- **Reasons for Implementing the Singleton Design Pattern in the Consultant website:**

  1- To make sure that a class has only one instance, and has one global point to it.

  2- To prevent objects from being instantiated more than once.

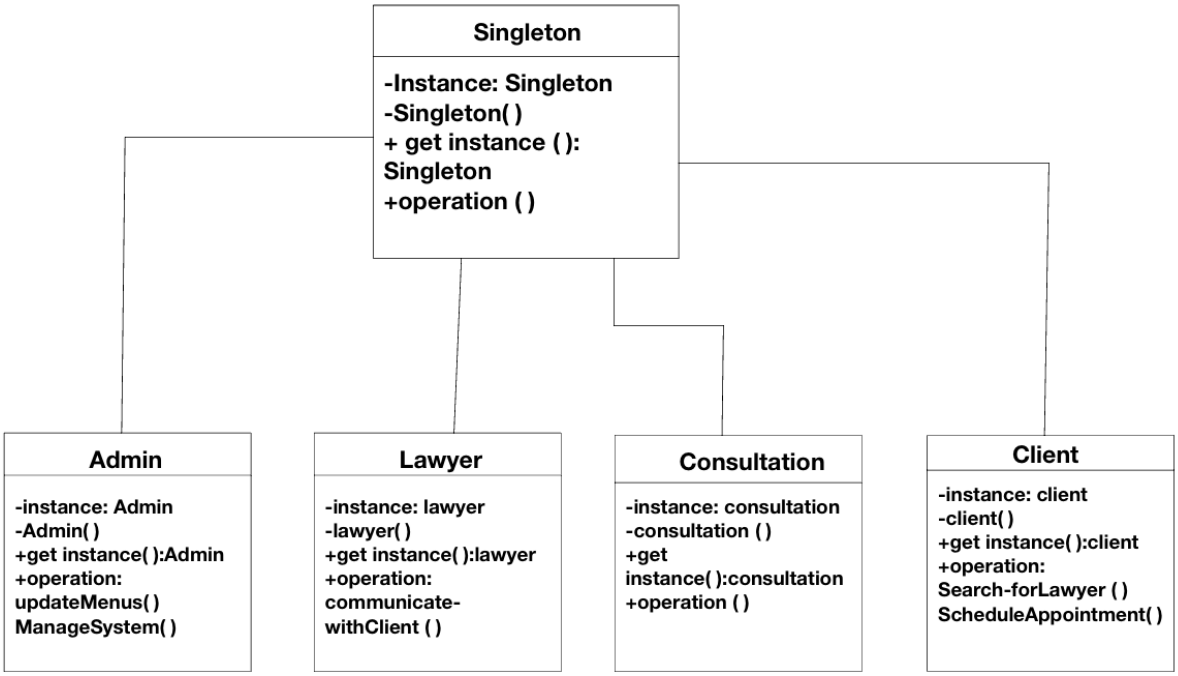
  3-To permit a variable number of instances.

- **Steps to implement the Singleton Design Pattern:**

  1- Define a private constructor.

  2- Define a static attribute that can store a reference to a Singleton instance.

  3- Create a public static getInstance( ) method.



6.2. Builder Design Pattern and Code Generation Example

- The Builder pattern is a creational design pattern that lets you make complex objects in a flexible way by keeping the building of the complex object separate from how it is represented. It has a director class that controls the building process through a builder interface. This technique is especially helpful when building an item that can have a lot of different parts or configurations.

- On the "The Consultant" website, the Builder pattern can be used to make a complex object that represents a consulting meeting. This gives you the freedom to set up meeting members, the schedule, and communication tools.

Code Generator Example:

In the context of "The Consultant" website, a code generator can be used to automate the generation of code snippets or configurations for different aspects of the consultancy platform. This could include generating code for communication tools, meeting scheduling, or user authentication.

The code generator could be a separate module or service that takes input parameters (such as meeting details) and produces the corresponding code or configuration files. This can help in reducing manual coding errors and ensuring consistency across different parts of the system.

- **Reasons for Implementing The Builder Pattern in the Consultant Website:**

1. Complex Object Construction: The Builder design is useful when building a complicated item that needs to be set up in several different ways. For the "The Consultant" website, setting up a meeting with all of its parts (attendees, schedule, contact tools) can be hard to do.
2. Flexibility and Customization: The Builder design makes it possible to create different versions of an object. This is helpful when the needs of consultants and clients mean that talks need to be set up in different ways.
3. Readability and Maintainability: The code is easier to read and keep up to date when the building process is kept separate from the real representation of the object. The steps for building are easier to understand, and the steps for building can be changed without changing the product class.
4. Avoiding Telescoping Constructors: Using telescoping builders can be hard to understand and make code more likely to make mistakes as the number of inputs for building an object grows. The Builder design is a better choice because it is cleaner.

- **Steps to Implement The Builder Design Pattern:**

1. Private Constructor: Ensure that the class has a private constructor to prevent external instantiation.
2. Static Instance Method: Provide a static method that returns the singleton instance. This method should create the instance if it doesn't exist, or return the existing instance.
3. Thread Safety (Optional): If your application is multi-threaded, consider adding synchronization to the getInstance method to ensure thread safety.
4. Accessors and Mutators (Optional): Add methods to access and modify the configuration settings within the singleton instance. For example Usage:ConfigurationManager configManager = ConfigurationManager.getInstance();

String settingValue = configManager.getSetting("someKey");

6.3. The Prototype Design Pattern

- The prototype design pattern creates objects efficiently by copying existing ones, known as prototypes, rather than building from scratch. This involves cloning an instance, serving as a blueprint and proves useful when object creation costs are high. The pattern fosters flexibility in crafting new objects with diverse configurations, minimizing the reliance on subclassing.

- **Reasons for implementing this pattern in the Consultant website:**
    - Efficient User Configuration:
        - Prototype pattern streamlines user-specific settings and preferences.
        - Users clone existing configurations for quick consultation profile setup.
    - Customized Legal Templates:
        - Prototype pattern aids in crafting tailored legal templates.
        - Users modify standard legal documents to suit individual needs.
    - Time and Cost Savings in Booking:
        - Cloning consultation booking objects reduces time and resource requirements.
        - Swift replication of past arrangements ensures an efficient booking process.
    - Consistent Payment Handling:
        - Prototype pattern enables cloning of payment configurations for standardized transactions.
        - Enhances payment system reliability and simplifies workflow updates.

- **Steps to Implement The Prototype Design Pattern:**
    1. **Create a Prototype Interface:** Define an interface or abstract class that declares a method for cloning objects.
    2. **Implement Concrete Prototypes:** Develop concrete classes that implement the prototype interface, providing the cloning functionality by copying their own state.
    3. **Create a Prototype Manager:** Establish a manager to keep track of prototype instances, facilitating easy access and retrieval.
    4. **Clone Objects Instead of Creating:** Instead of creating objects from scratch, clone existing instances when a new object is needed. This reduces the complexity and cost associated with object creation.

7. References

- GeeksforGeeks. (2023, October 31). *Singleton Design Pattern Implementation*. https://www.geeksforgeeks.org/singleton-design-pattern/
- GeeksforGeeks. (2022, December 5). *Builder design pattern*. https://www.geeksforgeeks.org/builder-design-pattern/
- Eduard Ghergu Software Architect. (2023, June 27). *Prototype Design Pattern - Definition & Examples | Pentalog*. Pentalog. https://www.pentalog.com/blog/design-patterns/prototype-design-pattern/