Software Design Specification

Table of Contents

LIST OF TABLES

LIST OF FIGURES

DEFINITIONS, ACRONYMS, ABBREVIATIONS

Term	Description
Client	Owner of the project
Freelancer/Job seeker	User of the system who is looking for a job
Employer/Recruiter	User of the system who is looking for skilled worker
SDS	Software Design Specification
Job Portal System/ Bridge Freelancing Platform	The freelancing platform
Escrow	Legal arrangement in which third party holds the money

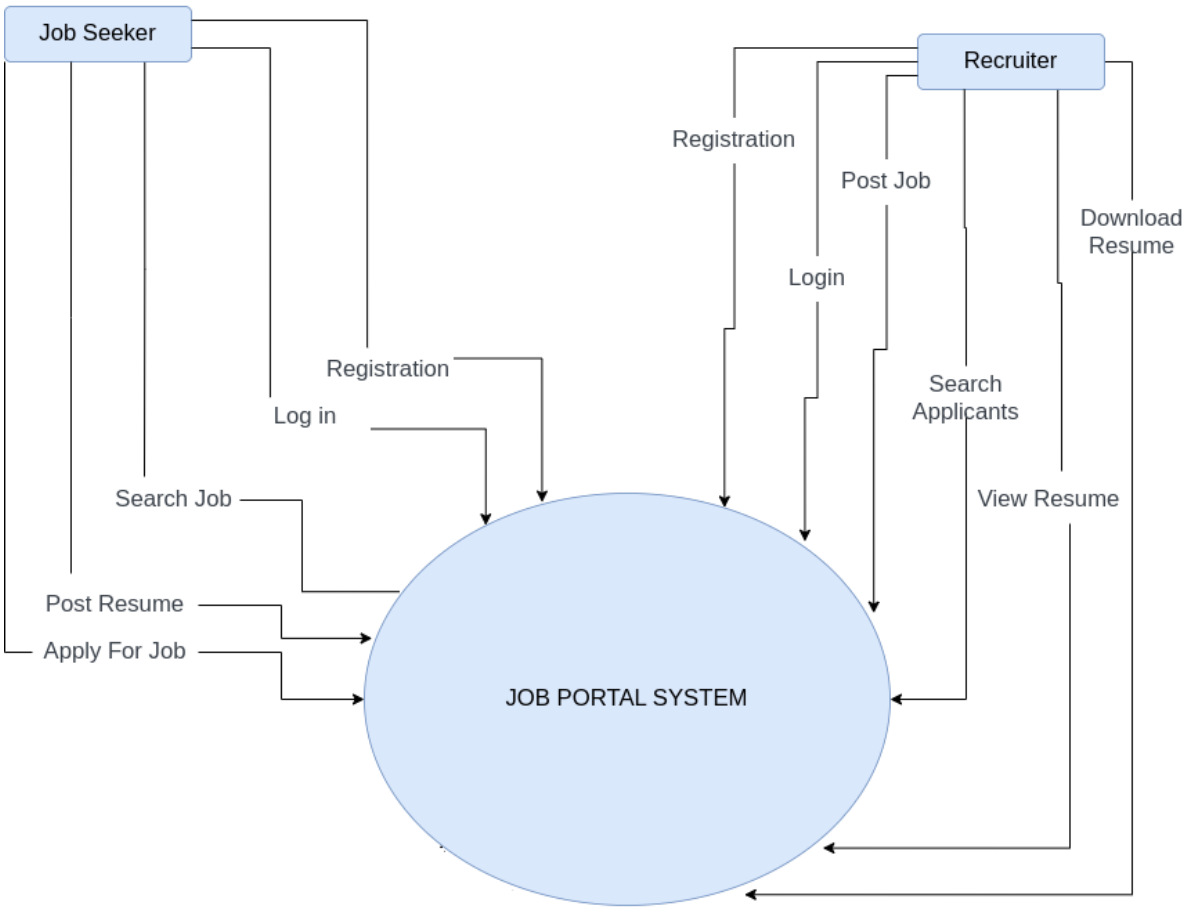
1. INTRODUCTION

1.1 Purpose

This is the Software Design Specification (SDS) for the Bridge Freelancing Platform project. The SDS will break down the project into sub-components to describe in detail what the purpose of each component is and how it will be implemented to meet the requirements listed on the Software Requirements Specification of the project. The SDS will also serve as a tool for verification and validation of the final product.

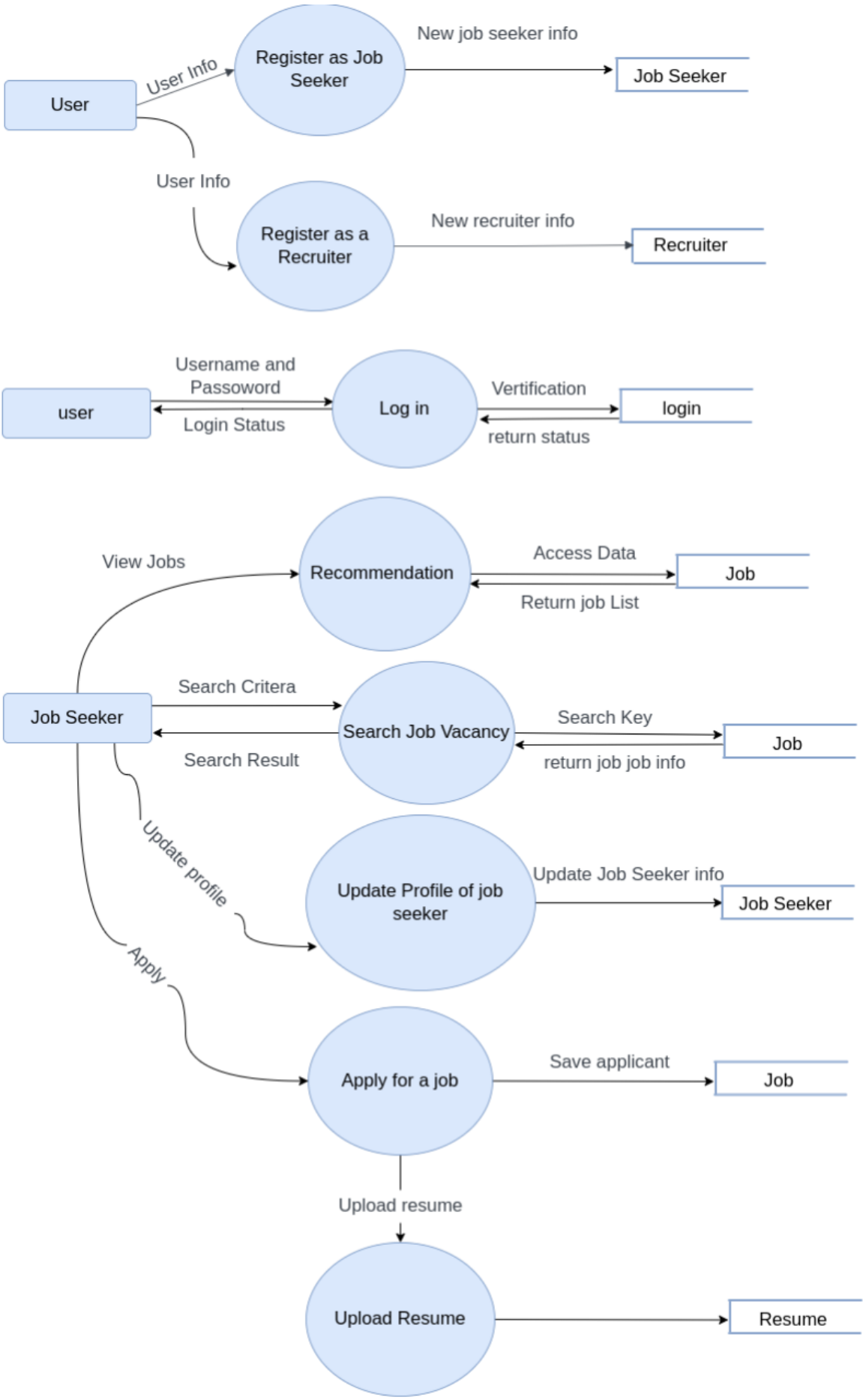
1.2 General Overview

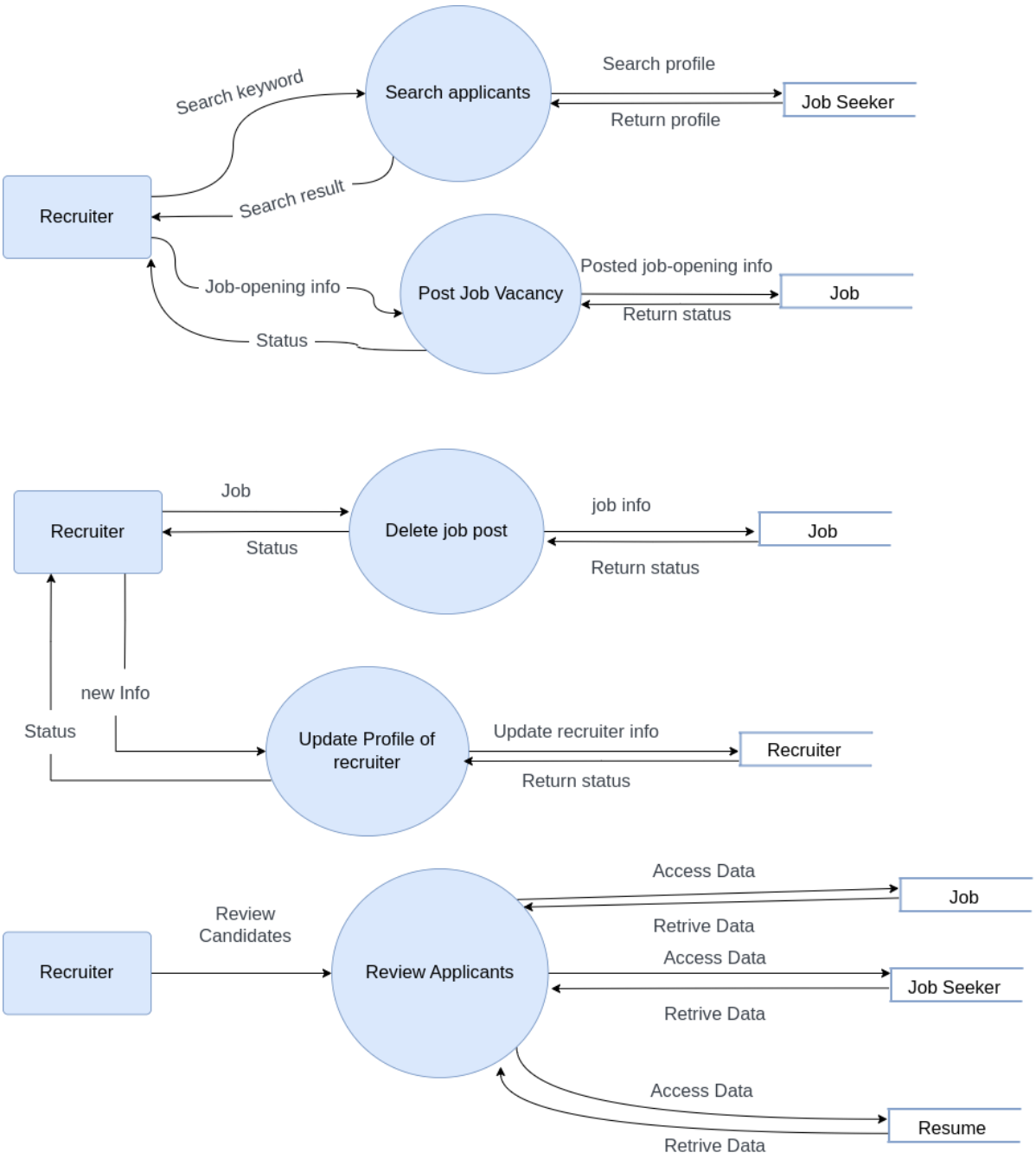
Decomposition and prototyping approaches are used to examine and fine-tune the requirements. A simplified and follow the template will be used to establish the criteria for the employment portal system in a clear and straightforward way. To make sure that all requirements are traceable and that traceability is maintained throughout the project, we will also employ tools like requirements traceability matrices. The high level context diagram of the system is presented as follows:



Since the above context diagram only gives a high level description of the system's components and their operations, it would not be sufficient to optimize, comprehend and implement new features and processes to the system. Data flow diagram was used to address this problem. Not only can it help better understand the system processes and operations but also helps discover potential problems, improve efficiency and develop better processes.

Below is a Level One Data Flow Diagram, that expands processes in the context diagram to elaborate a major high level process within the system.


User
User Info
Register as Job Seeker
New job seeker info
Job Seeker
User Info
Register as a Recruiter
New recruiter info
Recruiter
user
Username and Passoword
Login Status
Log in
Vertification
return status
login
View Jobs
Recommendation
Access Data
Return job List
Job
Job Seeker
Search Critera
Search Result
Search Job Vacancy
Search Key
return job job info
Job
Update profile
Update Profile of job seeker
Update Job Seeker info
Job Seeker
Apply
Apply for a job
Save applicant
Job
Upload resume
Upload Resume
Resume



1.3 Development Methods and Contingencies

Design Strategies

Two major software design strategies were used to conceptualize the software requirements to implementation. Function-oriented design and Object-oriented design.

Function-oriented design was used to decompose the model into a set of interacting units and modules where each module clearly defined a function. The big "*Freelancing platform*" concept was initially decomposed into two major functions:

- Employer can post Jobs, &
- Freelancer can apply for jobs

These major functions were further decomposed into:

- Employer can delete posted jobs
- Employer can review proposals/ applications
- Freelancer can submit proposals
- Freelancer can search for a job

In this manner all the operations in the system were identified. The design process heavily relied on how data flows through the system, depicting how functions change the data and the entire state of the system.

Object-oriented Design was used in the actual implementation of the system. All major actors of the system were taken as classes to make use of the various functionalities object oriented design can offer.

Important object-oriented-design concepts which were included in the system:

- ➢ Object - all entities included in the system including employer, freelancer, job were treated as objects. Every entity has some associated attributes and methods.
- ➢ Class - generalized description of objects, where all the attributes and methods which define the functionality of objects are defined. User class for employer and freelancer.
- ➢ Encapsulation - bundling of data and restriction of access to data. Information hiding was applied on Escrow payment handling process to sensitive data.
- ➢ Polymorphism - abstract classes like payment handler class were built so as to make adding new payment handler systems uniform by implementing the same abstract class.

Design was based on grouping objects to classes based on similarities and attribute characteristics and defining class hierarchies and relations.

Design Approach

The Software Design Approach used was a "**Top Down**" approach, where each component is considered as a system and is decomposed further. It starts with a generalized model of the system and keeps on defining a more specific part of it. The process kept on going until the lowest level hierarchy was achieved.

Contingencies

As mentioned above, the development method used for the freelancing platform was a combination of function-oriented design and object-oriented design. The design process relied on decomposing the system into modules and functions, as well as defining objects, classes, and encapsulation.

However, despite the approach taken, there are still several contingencies that might arise during the design and development of the software that could change the direction of the project. Some of these include:

1. **Requirements changes:** As the project progresses, the client or stakeholders might request changes to the requirements, which could impact the design and overall direction of the project.

   Workaround: To mitigate this risk, it is important to have a clear and effective communication process between the development team and the stakeholders. For that purpose, we document all requirement changes and assess their impact on the project.

2. **Technical limitations:** We might encounter technical limitations that prevent us from implementing certain features or design elements as originally intended.

   Workaround: In such cases, our development team identifies alternative solutions or workarounds that still meet the requirements while overcoming the technical limitations.

3. **Performance issues:** During the development process, we may discover that certain parts of the software are not performing as expected, which could affect the overall system performance.

   Workaround: To address this, our team uses performance testing and profiling tools to pinpoint the source of performance issues and implement optimization techniques to resolve them.

4. **Integration issues:** Integrating the different modules and components of the software can also present challenges, particularly if the components were developed by different teams.

   Workaround: To mitigate this risk, our development team performs a thorough testing process, including integration testing, to identify and address any integration issues before the final release of the project.

5. **Regulatory Compliance:** As a platform, we must adjust our policies and practices to comply with regulatory requirements, which may change over time.

   Workaround: To work around this, we stay up to date on regulatory changes and adjust our platform accordingly.

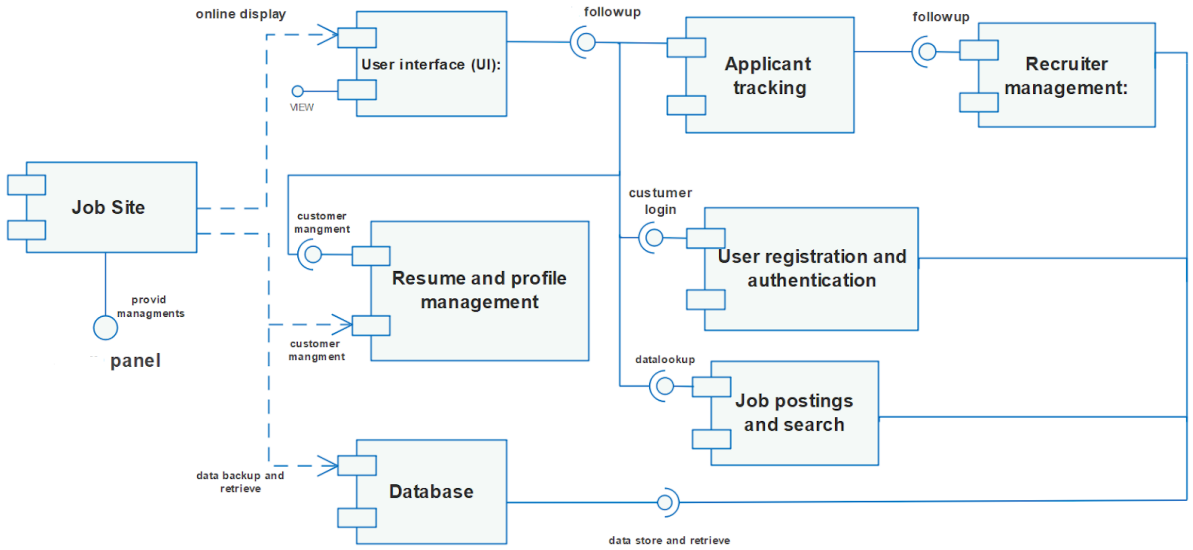
6. **Security Risks:** We recognize that, like any online platform, we face security risks. If a security breach occurs, our development team must shift their priorities to address the issue.

   Workaround: To mitigate this risk, we work to implement robust security measures and conduct regular security audits.

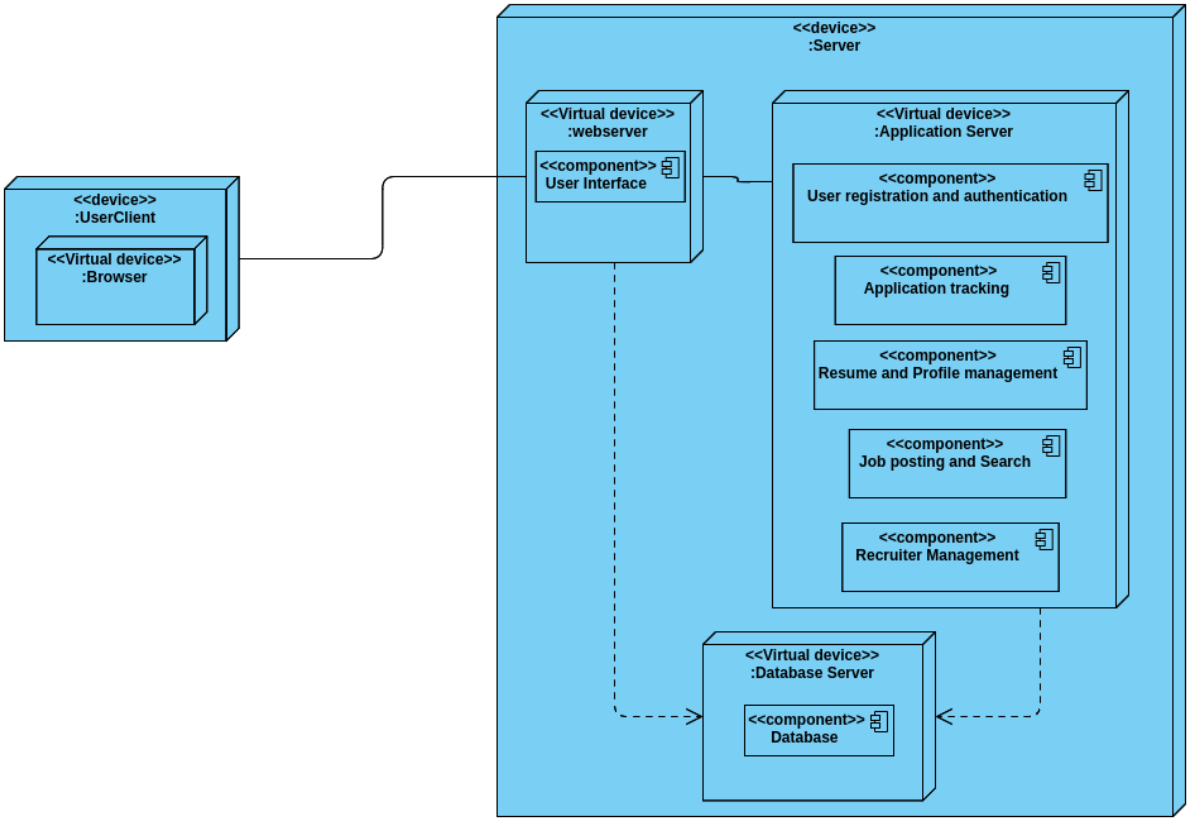
These are some of the contingencies that could arise during the design and development of our freelancing platform. By being proactive and having a contingency plan in place, our development team can minimize the impact of these risks and ensure the success of the project.

2. SYSTEM ARCHITECTURE

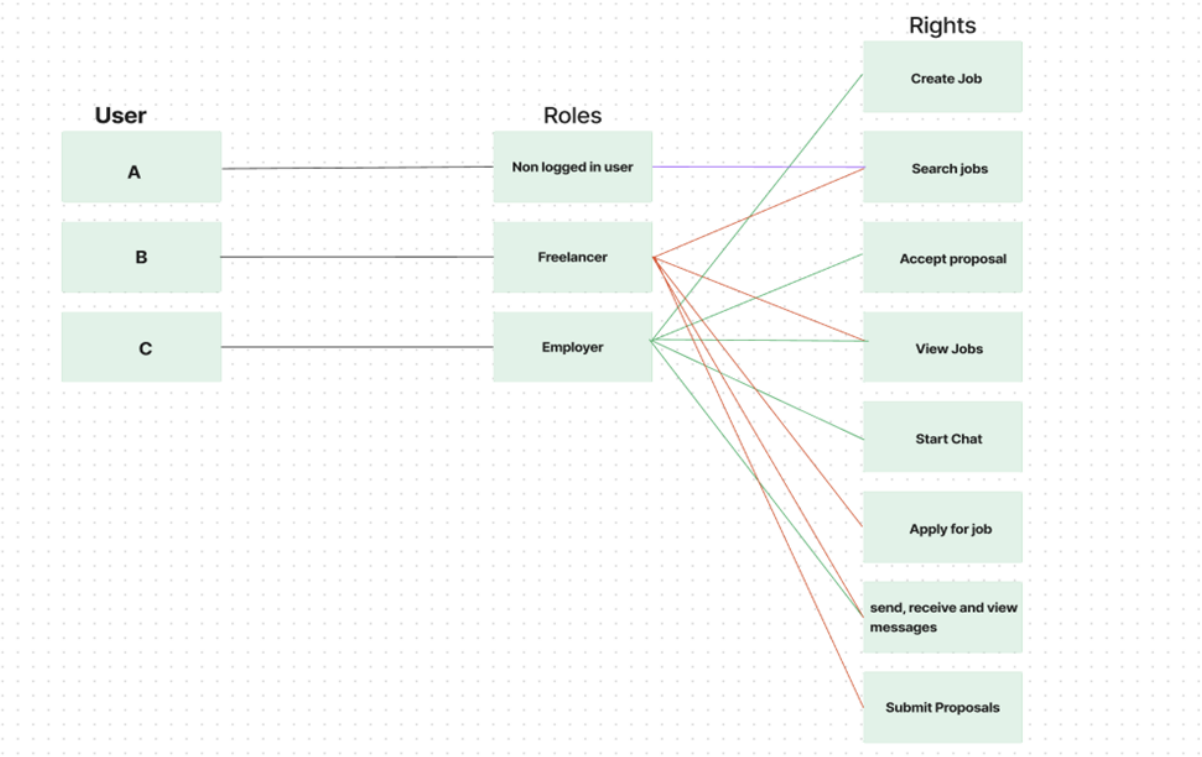
2.1 Subsystem Decomposition



2.2 Hardware/ Software Mapping

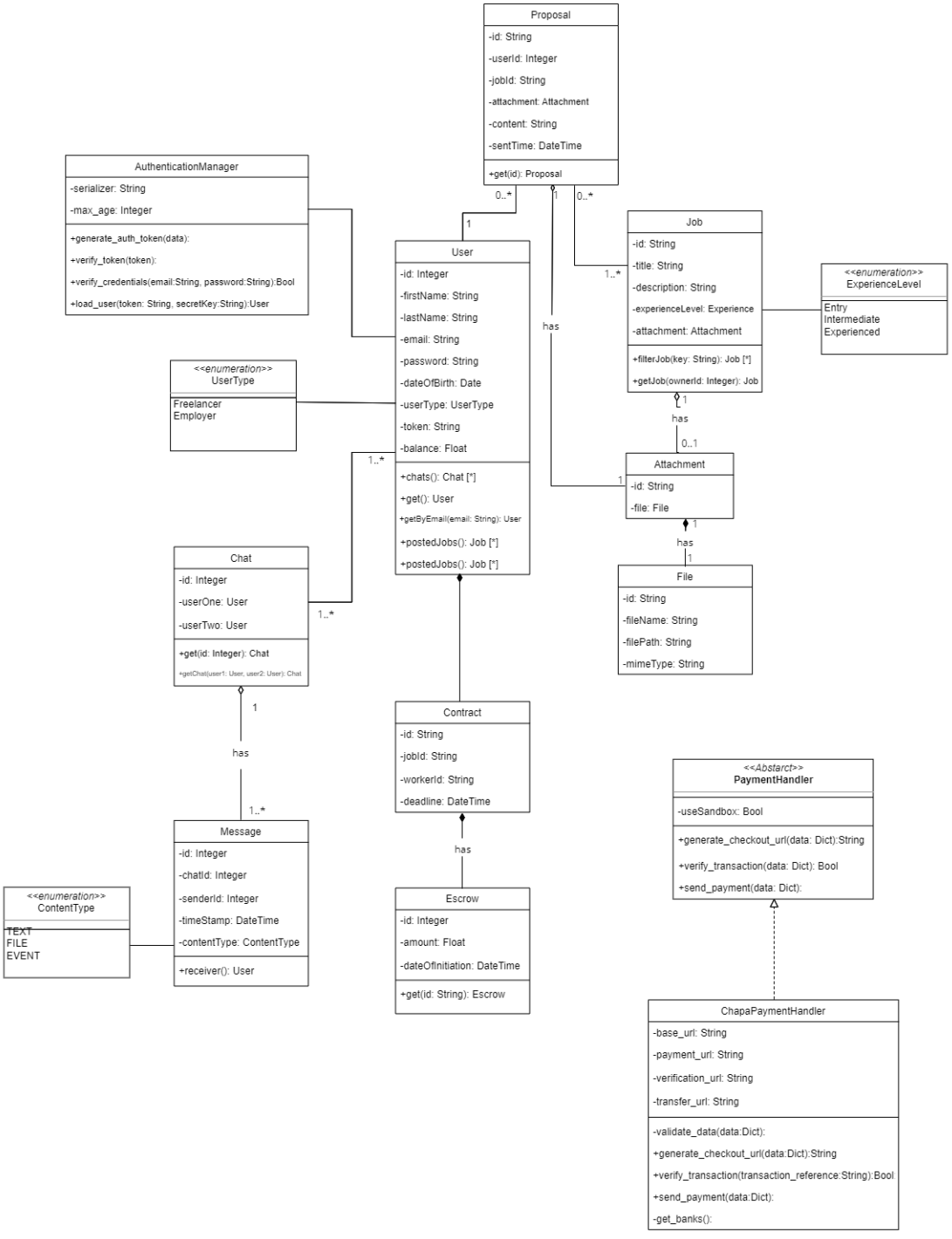


2.3 Access Control



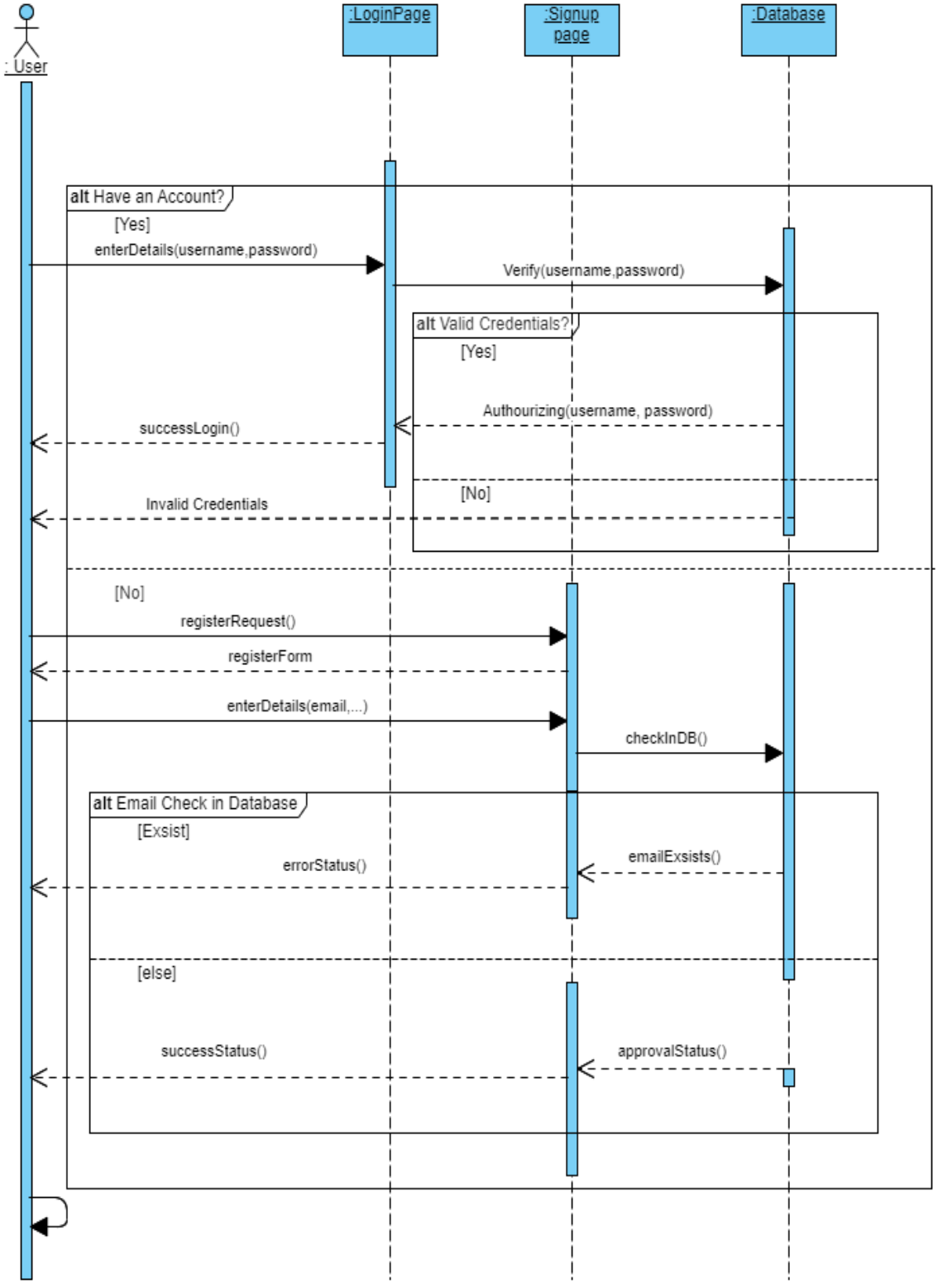
3. OBJECT MODEL

3.1 Class Diagram

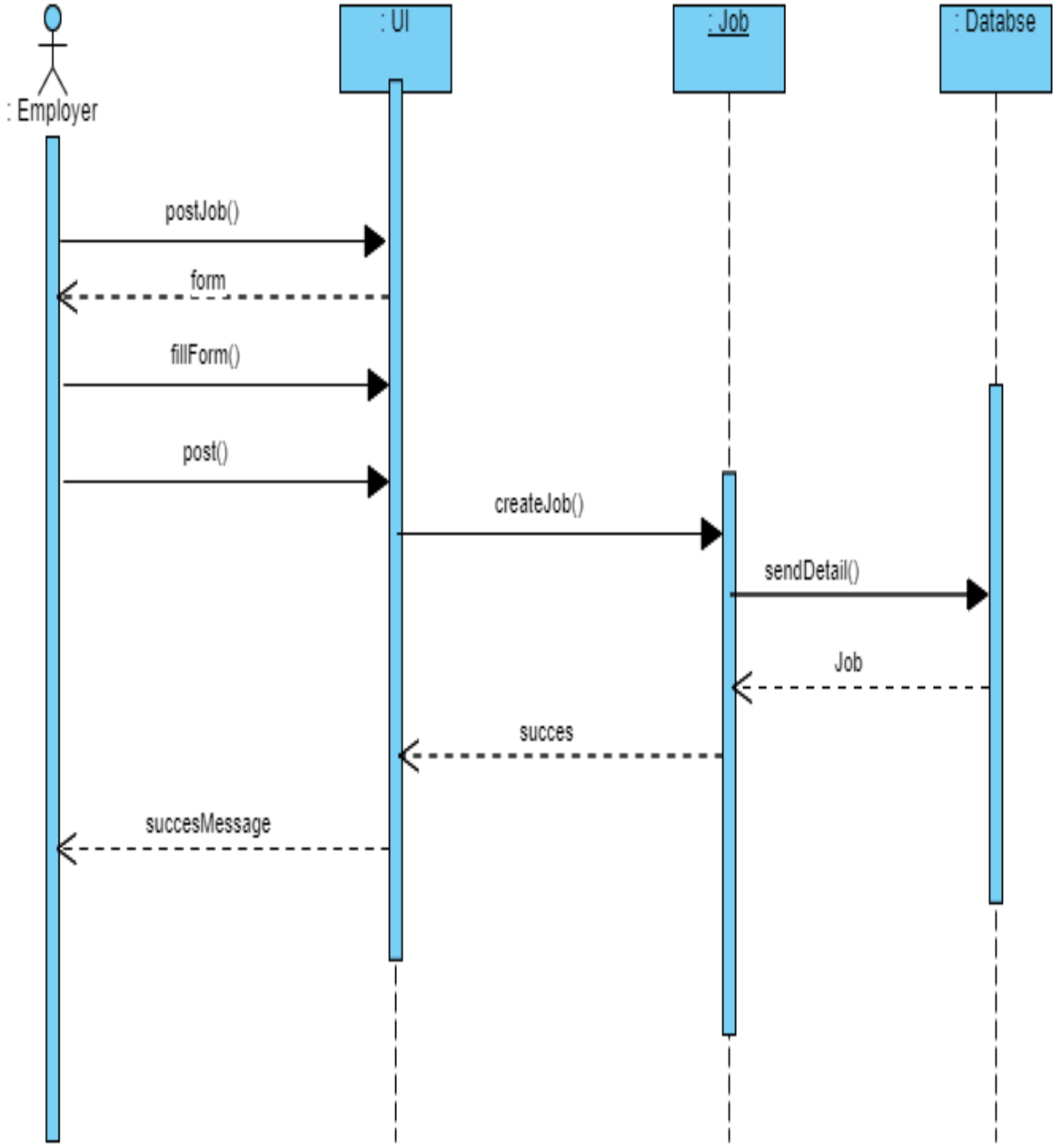


3.2 Sequence Diagrams

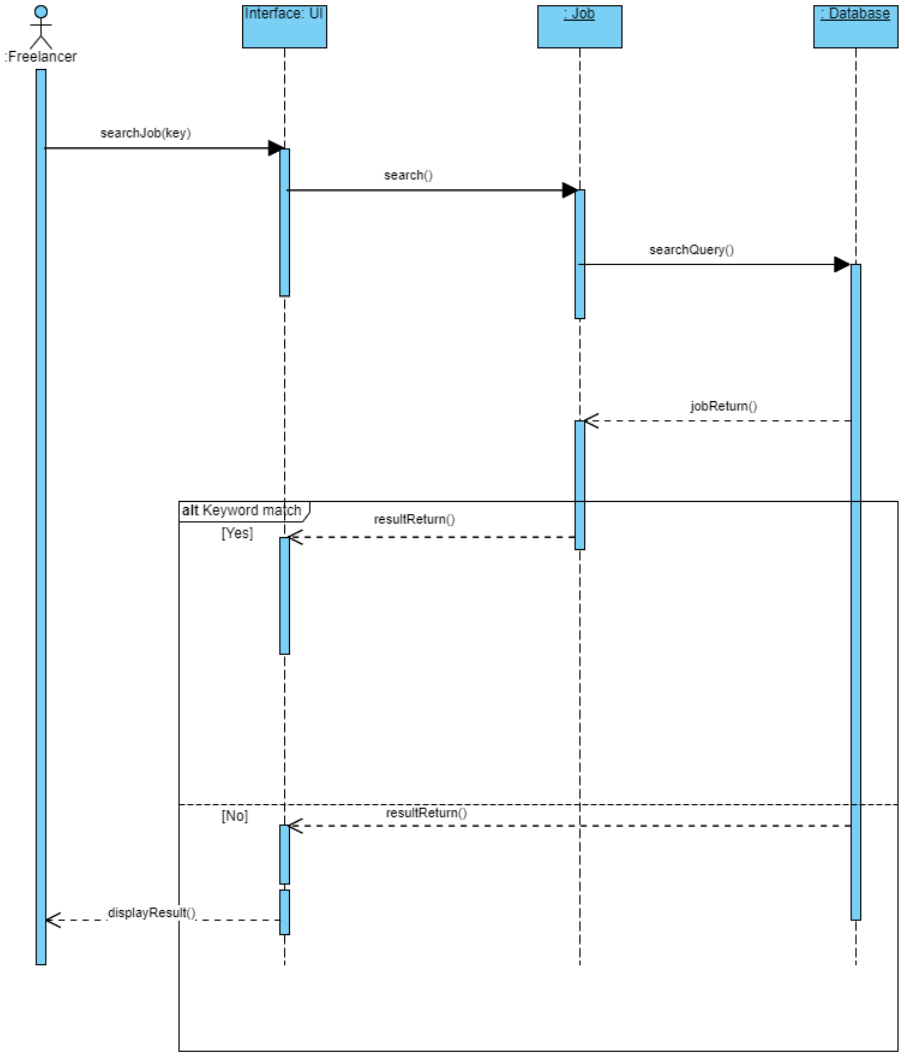
Authentication



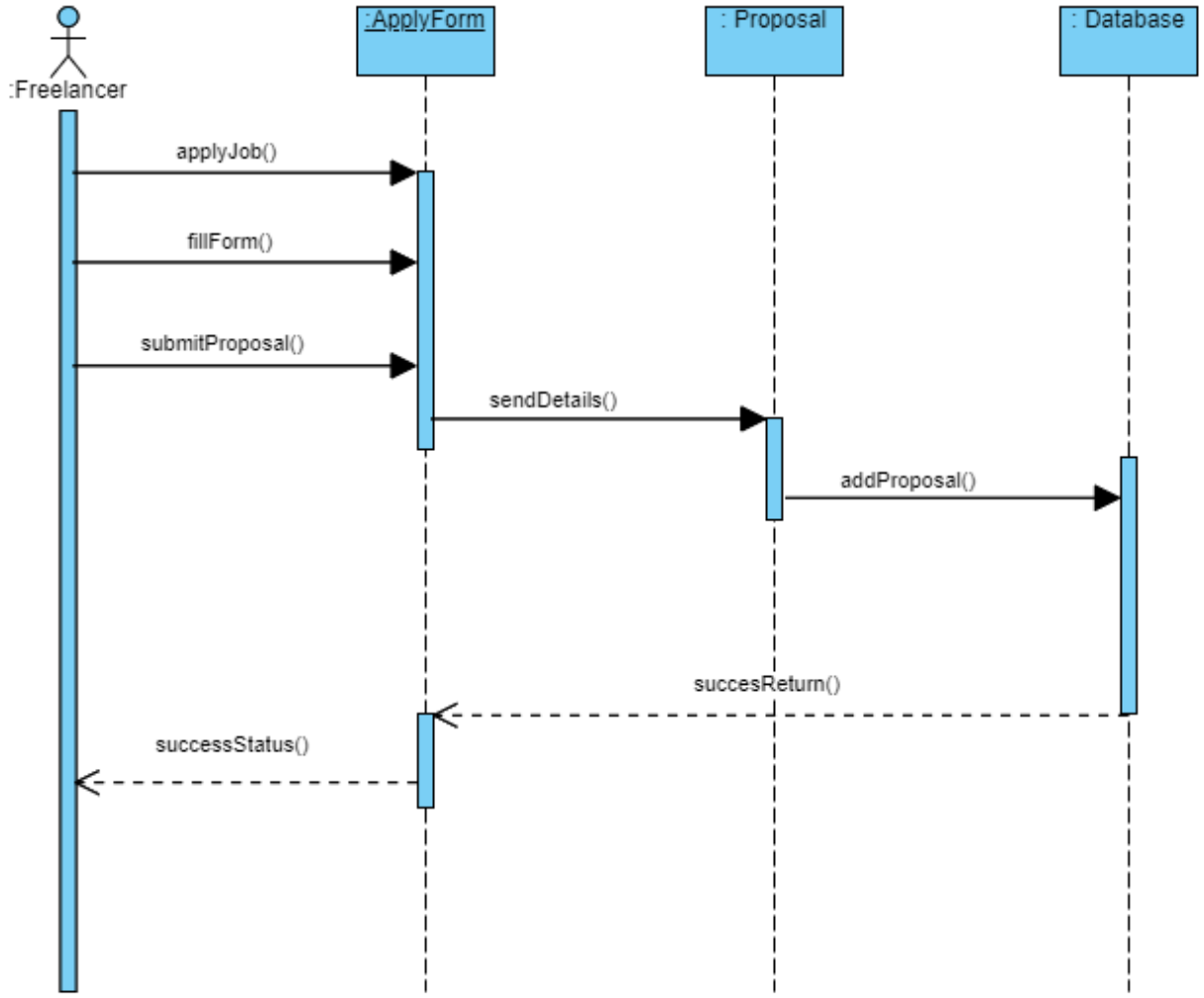
Post Job


: Employer
: UI
: Job
: Databse
postJob()
form
fillForm()
post()
createJob()
sendDetail()
Job
succes
succesMessage

Search Job

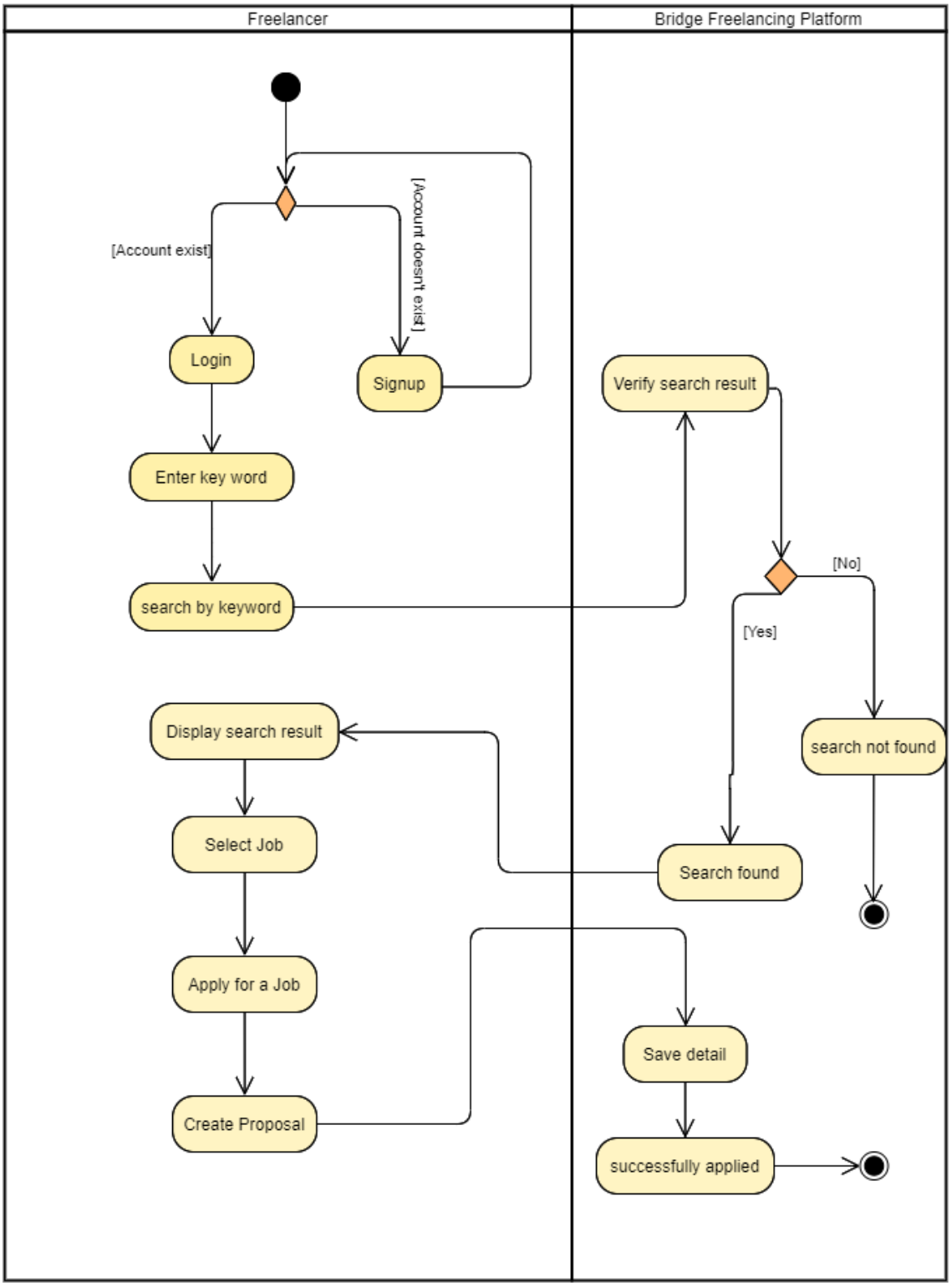

:Freelancer
Interface: UI
: Job
: Database
searchJob(key)
search()
searchQuery()
jobReturn()
alt Keyword match
[Yes]
resultReturn()
[No]
resultReturn()
displayResult()

Apply for a Job

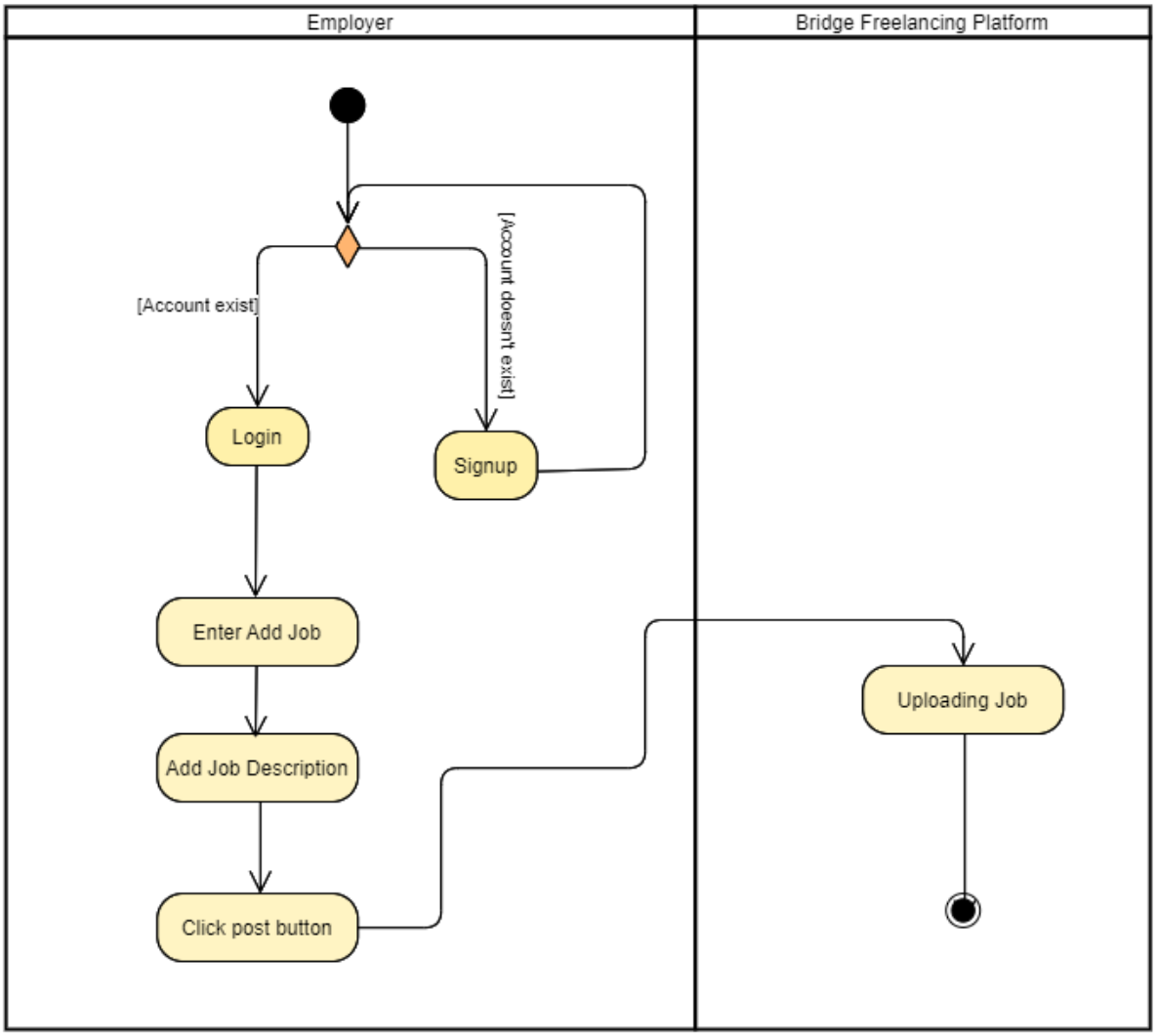


3.3 State Chart Diagram

Apply for a Job



Post Job


Employer
Bridge Freelancing Platform
[Account exist]
[Account doesn't exist]
Login
Signup
Enter Add Job
Add Job Description
Click post button
Uploading Job

4. DETAILED DESIGN

Detailed design is a crucial aspect of developing any website, especially a freelance website. A well-designed freelance website should provide a seamless user experience to both freelancers and clients by facilitating the process of finding and hiring talent. The detailed design stage involves breaking down the website's features and functionality into smaller components and determining how they should look and work together. This stage is critical as it sets the foundation for the development of the website, ensuring that it meets the desired requirements and expectations. In this article, we will delve into the key components of a detailed design for a freelance website. In this section, the attributes and operations of each individual class will be examined.

Table: 1 User Class

User
-ID : Integer **+FirstName : String** **+LastName : String** **-Email : String** **-Password : String** **-DateOfBirth : Date** **-UserType : UserType** **-Token : String** **-Balance : Float**
+chats() : Chat[*] **+get() : User** **+getByEmail(email : String) : User**

+postedJobs() : Job[*]

+viewApplicantsRating(user_id : Integer) : UserRating

-uploadResume(resume : File) : Boolean

-updateResume(resume: File) : Boolean

Table: 2 Attribute Description for User Class

Attribute	Type	Visibility	Invariant
ID	Integer	Private	Not null, unique
FirstName	String	Public	Not null, Should not contain special characters and integer
LastName	String	Public	Not null, Should not contain special characters and integer
Email	String	Private	Not null, unique, It should not be empty. Must contain • @ • . (dot) Position of • @ >1 • (dot)>position of @ + 2 • (dot)+3<= total length of email address and the total character of the Email is at least 5 characters
Password	String	Private	Not null
DateOfBirth	Date	Private	Not null
UserType	UserType	Private	Not null
Token	String	Private	Not null, unique
Balance	Float	Private	Not null, It should not be negative

Table: 3 Operation Description for User Class

Operation	Visibility	Return Type	Argument	Pre-Condition	Post-Condition
chats	Public	Chat[*]	None	None	The method returns an array of all the Chat objects that the User is involved in.
Get	Public	User	None	None	The method returns the User object of the user that is currently accessing the method.

getByEmail	Public	User	Email : String	The email address provided must match the format of a valid email address.	The method returns the User object of the user that has the specified email address.
postedJobs	Public	Job[*]	None	None	The method returns an array of all the Job objects that have been posted by the User.

Table: 4 Chat Class

Chat
-ID : Integer **-User_1 : User** **-User_2 : User**
+get(ID : Integer) : Chat **+getChat(User_1 : User, User_2 : User) : Chat**

Table: 5 Attribute Description for Chat Class

Attribute	Type	Visibility	Invariant
ID	Integer	Private	Not null, unique
User_1	User	Private	Not null, remain constant throughout the chat
User_2	User	Private	Not null, remain constant throughout the chat

Table: 6 Operation Description for Chat Class

Operation	Visibility	Return Type	Argument	Pre-Condition	Post-Condition
get	Public	Chat	ID : Integer	The "ID" argument must be a valid identifier for a chat in the system.	The function returns an object of the "Chat" class that corresponds to the chat identified by the "ID" argument.
getChat	Public	Chat	User_1 : User	Both "User_1" and "User_2" must be valid	The function returns an object of the "Chat" class

			User_2 : User	user objects in the system, and there must exist a chat between the two users.	that corresponds to the chat between the two users "User_1" and "User_2".

Table: 7 Message Class

Message
-ID : Integer **-ChatID : Integer** **-SenderID : Integer** **-TimeStamp : DateTime** **-ContentType : ContentType** **-Content : String**
+ receiver() : User

Table: 8 Attribute Description for Message Class

Attribute	Type	Visibility	Invariant
ID	Integer	Private	Not null, unique
ChatID	Integer	Private	Not null, unique
SenderID	Integer	Private	Not null, unique
TimeStamp	DateTime	Private	Not null, valid date and time
ContentType	ContentType	Private	Not null, must be one of the value listed in the ContentType class
Content	String	Private	Not null, max varchar of 4000

Table: 9 Operation Description for Message Class

Operation	Visibility	Return Type	Argument	Pre-Condition	Post-Condition
receiver	Public	User	None	The message object must have been instantiated	Returns the user object that

				and the user object associated with the receiver's ID must exist.	represents the receiver of the message.

Table: 10 File Class

File
-ID : String **-file_name : String** **-file_path : String** **-mimeType : String**

Table: 11 Attribute Description for File Class

Attribute	Type	Visibility	Invariant
ID	String	Private	Not null, unique
file_name	String	Private	Not null, does not contain invalid character as “/” or “.”
file_path	String	Private	Not null
mimeType	String	Private	Not null, must be valid MIME type that is recognized by the system

Table: 12 Job Class

Job
-ID : String **-Title : String** **-Description : String**

-Experience_Level : Experience **-Attachment_ID : String** **-Budget : Float** **-Owner_ID : Integer** **-Post_Time : DateTime**
+filterJob(key : String) : Job[*] **+getJob(ownerId : Integer) : Job** **-apply_for_a_job(jobId : Integer) : Boolean** **-postJob(job_title: String, Job_description :String, budget : float) : Boolean** **-delete_job_post(jobId : Integer) : Boolean**

Table: 13 Attribute Description for Job Class

Attribute	Type	Visibility	Invariant
ID	String	Private	Not null, unique
Title	String	Private	Not null, should remain the same
Description	String	Private	Not null
Experience_Level	Experience	Private	Not null
Attachment_ID	String	Private	Not null
Budget	String	Private	Not null
Owner_ID	Integer	Private	Not null, unique
Post_Time	DateTime	Private	Not null, valid date and time

Table: 14 Operation Description for Job Class

Operation	Visibility	Return Type	Argument	Pre-Condition	Post-Condition
filterJob	Public	Job[*]	Key : String	None	The method should return an array of all the “Job” objects that have the given “key” in their job title or job description.

getJob	Public	Job	ownerId : Integer	A “Job” object with the given “ownerId” must exist in the system.	The method should return the “Job” object that has the given “ownerId”.
apply_for_a_job	Private	Boolean	jobId : Integer	A “Job” object with the given “jobId” must exist in the system and the job must be open for applications.	The method should return “True” if the job application was successful, and “False” otherwise.
postJob	Private	Boolean	job_title : String, Job_description : String, budget : float	The user must be authenticated and have the necessary permissions to post a job.	The method should return “True” if the job posting was successful, and “False” otherwise.
delete_job_post	Private	Boolean	jobId : Integer	A “Job” object with the given “jobId” must exist in the system and the user must be authenticated and have the necessary permissions to delete the job.	The method should return “True” if the job deletion was successful, and “False” otherwise.

Table: 15 Contract Class

Contract
-ID : String **-JobID : String** **-WorkerID : Integer** **-Deadline : DateTime**

Table: 16 Attribute Description for Contract Class

Attribute	Type	Visibility	Invariant
ID	String	Private	Not null, unique
JobID	String	Private	Not null, unique
WorkerID	Integer	Private	Not null, unique
Deadline	DateTime	Private	Not null, valid date and time in the future

Table: 17 Escrow Class

Escrow
-ID : Integer **-Contract_ID : String** **-Amount : float** **-DateOfInitiation : DateTime**
+get(ID : String) : Escrow

Table: 18 Attribute Description for Escrow Class

Attribute	Type	Visibility	Invariant
ID	Integer	Private	Not null, unique
Contract_ID	String	Private	Not null, unique
Amount	Float	Private	Not null, non-negative
DateOfInitiation	DateTime	Private	Not null, valid datetime object

Table: 19 Operation Description for Escrow Class

Operation	Visibility	Return Type	Argument	Pre-Condition	Post-Condition
get	Public	Escrow	ID : String	The precondition of this method is that the “ID” argument must represent a valid escrow in the system.	The post-condition of this method is that it returns an instance of the “Escrow” class that corresponds to the escrow identified by the “ID” argument. If the “ID”

					does not correspond to a valid escrow, an error or exception may be raised.

Table: 20 Attachment Class

Attachment
-ID : String **-File_ID : String**

Table: 21 Attribute Description for Attachment Class

Attribute	Type	Visibility	Invariant
ID	String	Private	Not null, unique
File_ID	String	Private	Not null, unique

Table: 22 User_balance Class

User_Balance
-User_ID : Integer **-Amount : Float**

Table: 23 Attribute Description for User_balance Class

Attribute	Type	Visibility	Invariant
User_ID	Integer	Private	Not null, unique
Amount	Float	Private	Not null, non-negative

Table: 24 Proposal Class

Proposal
-WorkerID : Integer

-Content : String **-Attachment : String** **-JobID : String** **-Sent_Time : DateTime**
-submitProposal(JobID : Integer ,proposal : Attachment) : Boolean **+get(ID : Integer) : Proposal**

Table: 25 Attribute Description for Proposal Class

Attribute	Type	Visibility	Invariant
WorkerID	Integer	Private	Not null, unique
Content	String	Private	Not null
Attachment	String	Private	Not null
JobID	String	Private	Not null, unique
SentTime	DateTime	Private	Not null

Table: 26 Operation Description for Proposal Class

Operation	Visibility	Return Type	Argument	Pre-Condition	Post-Condition
submitProposal	Private	Boolean	JobID : Integer, proposal : Attachment	- The job with the specified JobID must exist. - The proposal attachment must be a valid file and must meet the size and format requirements of the platform. - The user submitting the proposal must have the necessary permissions to do so.	- The proposal has been added to the list of proposals for the specified job. - The method should return “True” if the proposal submission was successful, and “False” otherwise.

get	Public	Proposal	ID : Integer	- The “ID” argument must correspond to a valid proposal on the website. - The user retrieving the proposal must have the necessary permissions to do so	The specified proposal has to be returned to the caller.

Table: 27 AuthenticationManager Class

AuthenticationManager
-serializer : String **-max_age : Integer**
+generate_auth_token(data : any) : String **+verify_token(token : String) : Boolean** **+verify_credentials(email : String, password : String) : Boolean** **+load_user(token : String, secretKey : String) : User** **+logIn(username : String, password : String) : Boolean** **+signUp(username : String, password : String, email : String) : Boolean**

Table: 28 Attribute Description for AuthenticationManager Class

Attribute	Type	Visibility	Invariant
serializer	String	Private	Not null, implements the required serialization and deserialization methods, must consistently serialize and deserialize user data in a secure and accurate manner.
max_age	Integer	Private	Not null, must be a positive integer , must be consistently enforced throughout the website.

Table: 29 Operation Description for AuthenticationManager Class

Operation	Visibility	Return Type	Argument	Pre-Condition	Post-Condition
generate_auth_token	Public	String	data : any	None	A token is generated based on the data provided as an argument.
verify_token	Public	Boolean	token : String	None	The validity of the token is verified and a Boolean value indicating the result is returned.
verify_credentials	Public	Boolean	email : String, password : String	None	The provided credentials are verified against the database and a Boolean value indicating the result is returned.
load_user	Public	User	token : String, secretKey : String	The token must be verified and the secret key must be correct.	A user object is returned based on the provided token and secret key.
logIn	Public	Boolean	username : String, password : String	None	The provided credentials are verified against the database, and if successful, the user is logged in and a Boolean value indicating the result is returned.
signUp	Public	Boolean	username : String, password : String, email : String	None	A new user is created with the provided credentials and a Boolean value indicating the result is returned.

Table: 30 PaymentHandler Class

PaymentHandler

-useSandbox : Boolean
+generate_checkout_url (data : Dict) : String +verify_transaction(data : Dict) : Boolean +send_payment(data : Dict) :

Table: 31 Attribute Description for PaymentHandler Class

Attribute	Type	Visibility	Invariant
useSandbox	Boolean	Private	Not null must always have a value “True” or False

Table: 32 Operation Description for PaymentHandler Class

Operation	Visibility	Return Type	Argument	Pre-Condition	Post-Condition
generate_checkout_url	Public	String	data : Dict	The “data” argument must contain all the required information for generating the checkout URL. This information could include the amount to be charged, the currency to be used, and the item or service being purchased.	This method returns a valid URL that can be used to initiate a payment, or it raises an exception if an error occurs.
verify_transaction	Public	Boolean	data : Dict	The “data” argument must contain the necessary information to verify the transaction, such as the transaction ID, the amount charged, and the payment status.	This method returns “true” if the payment was successful, or “false” if the payment was unsuccessful, or raises an exception if an error occurs.
send_payment	Public	Void	data : Dict	The “data” argument must contain the necessary information to send the payment, such as the recipient's	This method sends the payment to the recipient, updates the payment status in the system, or

				account information, the amount to be sent, and the currency to be used.	raises an exception if an error occurs.

Table: 33 ChapaPaymentHandler Class

ChapaPaymentHandler
-base_url : String **-payment_url :String** **-verification_url : String** **-transfer_url : String**
-validate_data (data : Dict) : any **+generate_checkout_url(data : Dict) : String** **+verify_transaction(transaction_reference : String) : Boolean** **+send_payment(data : Dict)** **-get_banks() : List**

Table: 34 Attribute Description for ChapaPaymentHandler Class

Attribute	Type	Visibility	Invariant
base_url	String	Private	Not null Must be in valid url format. It includes: • protocol: such as "http" or "https." • host name • path • Optional components
payment_url	String	Private	Not null Must be in valid url format. It includes: • protocol: such as "http" or "https." • host name

			• path • Optional components The URL must be relative to the base_url, i.e., it must start with a "/".
verification_url	String	Private	Not null Must be in valid url format. It includes: • protocol: such as "http" or "https." • host name • path • Optional components The URL must be relative to the base_url, i.e., it must start with a "/".
transfer_url	String	Private	Not null Must be in valid url format. It includes: • protocol: such as "http" or "https." • host name • path • Optional components The URL must be relative to the base_url, i.e., it must start with a "/".

Table: 35 Operation Description for ChapaPaymentHandler Class

Operation	Visibility	Return Type	Argument	Pre-Condition	Post-Condition
validate_data	Private	any	data : Dict	None	The function validates the information in the data argument.
generate_checkout_url	Public	String	data : Dict	The “data” argument should contain all necessary information required to generate a checkout URL, otherwise, the method may throw an error.	The method returns a valid checkout URL for the transaction.

verify_transaction	Public	Boolean	transaction_reference : String	The "transaction_reference" argument should be a valid reference for an existing transaction, otherwise, the method may throw an error.	The function verifies the transaction and returns a Boolean value indicating whether the transaction was successful or not.
send_payment	Public	Void	data : Dict	The "data" argument should contain all necessary information required to initiate a transaction, otherwise, the method may throw an error.	The method initiates a payment transaction with the provided details.
get_banks	Private	List	None	None	The method returns a list of banks supported by the payment handler.

Enumerated Tables

Table: 36 ContentType Enumeration Class

Define the types of content of files that can be attached on different file attachment sites on the platform.

ContentType
TEXT
FILE
EVENT

Table: 37 UserType Enumeration Class

Define the types of users that can register on a freelancing website.

UserType
Freelancer
Employer

Table: 38 ExperienceLevel Enumeration Class

Define the level of experience a job posted on the platform requires.

ExperienceLevel
Entry
Intermediate
Experienced

APPENDICES

Software Design Development tools

For the design development of the software, we have used the following integrated design tools.

- Visual Paradigm
- Lucid Chart
- Figma
- Draw.io

REFERENCES

Bibliography

- Pressman, R. S., & Maxim, B. R. (2015). Software engineering: a practitioner's approach. McGraw-Hill Education.

Web Resource

- Databases that map data at Feb 11, 2023(Use this link for Detailed Design)
- Database Invariant at Feb 11, 2023(Use this link for Detailed Design)
- Detailed Design Specification at Feb 12, 2023(Use this link for Detailed Design)
- Coursera Object Oriented Design at Jan 10, 2023(Use this link for Object oriented design Principles)
- Software Development Challenges at Feb 14, 2023(Use this link for contingencies)