République Tunisienne

Ministère de l'Enseignement Supérieur
et de la Recherche Scientifique
Université de Tunis

Ecole Nationale Supérieure
d'Ingénieurs de Tunis

Code : GPE-RP-01-00
Date de création : 16-06-2023

Département Génie Informatique

Ref : PFA1-2025-07

End of Year Project Report

of

First year in Computer Engineering

Presented and publicly defended on 08/05/2025

By

Mohamed Aziz Ouerfelli
Mohamed Aziz Foudhaili
Mahjoub ben Gaied Hassine

FreeConnect : freelance platform

Composition of the jury

Mrs.	Zoulel Kouki	President
Mrs.	Emna Souissi	Supervisor

Academic Year: 2024-2025

Dedications

We dedicate this project to our parents, our brothers and sisters, as well as our closest friends, who have been a constant source of support and inspiration.

We also extend our thoughts to our professors, whose advice and encouragement have been invaluable throughout this academic journey.

Acknowledgments

We wish to express our deep gratitude to the president of the jury for attending our defense and for the attention given to our work. Her comments and suggestions will be of great value in improving our project.

We would also like to warmly thank our supervisor, Mrs. Emna Souissi, for her constant support, valuable guidance, and the trust she placed in us throughout this experience.

Finally, we extend our heartfelt thanks to the entire teaching staff of the National Higher Engineering School of Tunis for their dedication and support during this project.

Table of Contents

Table of figures

Table of tables

General introduction

With digital solutions becoming the norm, flexibility and independence have become increasingly important in the professional landscape. Traditional ways of working are evolving, giving rise to new models where autonomy and remote collaboration become the main focus. Among these, freelance platforms are playing a crucial role, bridging the gap between clients and independent professionals across the globe.

A freelance platform allows businesses to connect with skilled individuals efficiently. The idea behind our project is driven by this growing demand for flexible workspaces. Our goal is to develop a user-friendly and functional web application that supports both freelancers and clients. The platform will include key features such as project posting and proposal submission.

Within the framework of our end of year project at the National Engineering School of Tunis (ENSIT), we aim to create a dynamic and intuitive freelance platform. This platform will not only facilitate freelance work but also promote efficiency and productivity among its users.

This report is divided into four chapters:

- The first chapter presents the preliminary study, including the project context, an analysis of existing solutions and a detailed specification of functional and non-functional requirements.
- The second chapter focuses on the detailed study, covering the data dictionary, conceptual and logical data models.
- The third chapter outlines the technical study, which includes the tools used, the physical database model, and the steps involved in building and populating the database.
- The final chapter presents the formulation and execution of SQL queries and the integration of a web application. This chapter also includes the system architecture, database connection, and user interface design.

The report concludes with a general summary and a discussion of future improvements.

Chapter 1. Preliminary study

Introduction

In this chapter, we will describe the details of the context of our project by providing a detailed description of the universe to be modeled, as well as a study and critique of the existing solutions. We will also specify the work to be carried out.

1.1. Background

Nowadays, numerous freelance platforms exist in the global market. These platforms are generally designed to facilitate various aspects of freelancing, providing a seamless experience for both freelancers and clients. They offer a range of features such as project posting, freelancer search, secure payment processing, and communication tools to streamline collaboration. Some platforms cater specifically to experienced professionals, offering advanced features like portfolio management, contract handling, and skill certification, while others focus on beginners, providing guidance and simplified job-matching systems.

Additionally, many freelance platforms incorporate social networking features, allowing users to connect, share insights, and build professional relationships within their industries. Some platforms even integrate AI-driven recommendations to match freelancers with relevant opportunities based on their skills and experience.

With this in mind, we have been tasked with developing a user-friendly freelance platform called **"FreeConnect"**, designed to connect freelancers with potential clients efficiently. The platform will offer a simple and intuitive interface, helping users find freelance opportunities that align with their expertise and preferences.

1.2. Description of the universe to be modeled

We aim to develop a web application designed to facilitate seamless connections between freelancers and clients. The platform will offer an efficient job-matching experience by allowing clients to post freelance opportunities while enabling freelancers to find projects that align with their skills and expertise. To support this, we need to create a comprehensive database to store job postings, freelancer profiles, and project-related information.

Job Listings

Each job posting will include:

- A **title** and **detailed description** outlining the project requirements.
- The **required skills** and the **experience level** expected for each skill.
- An **estimated project duration** (short-term, medium-term, or long-term).
- A **budget range** (fixed price or hourly rate).
- A **category** (e.g., web development, graphic design, content writing, digital marketing, etc.).

Freelancer Profiles

Each freelancer profile will include:

- **Full name and profile description** to highlight their expertise and professional background.
- A list of **skills and competencies** categorized based on proficiency level.
- An **hourly rate** or preferred pricing model.
- A **portfolio** showcasing previous work and achievements.
- A **rating and review system**, where clients can provide feedback based on completed projects.

Client Profiles

Clients using FreeConnect will have dedicated profiles with:

- **Business or personal details**, depending on the nature of their hiring needs.
- **A rating system**, where freelancers can review their experience working with the client.

Platform Features

The application must support:

- **Job Management:** Clients should be able to post, update, and manage job listings, while freelancers should be able to apply for and track their applications.
- **Advanced Filtering & Search:** Users should be able to filter jobs based on category, skill requirements, budget, project duration, and difficulty level.
- **Proposal & Bidding System:** Freelancers should be able to submit proposals, set pricing, and negotiate terms with clients.
- **Secure Payments:** The platform should introduce a secure payment system that ensures smooth transactions between freelancers and clients.

- **Messaging & Collaboration Tools:** A built-in chat system will allow real-time communication and project discussions.

By integrating these features, **FreeConnect** aims to streamline the freelancing process, making it easier for professionals to find work and for clients to connect with skilled talent efficiently.

1.3. Study and critique of the existing

As part of our EYP, we conducted a comparative study of different freelance platforms, such as Upwork, Fiverr, and Malt, to evaluate their performance, features, and user experience. The objective of this study is to understand how these platforms facilitate the connection between freelancers and companies, as well as to evaluate the quality of the services offered, the management of payments, and the ergonomics of the user interface.

This part of our work presents and analyzes existing solutions, focusing on the strengths and weaknesses of each platform, and exploring how they meet the needs of freelancers in diverse industries. We also considered the diversity of features offered, such as project management tools, transaction security, and ease of use, to better understand their impact on the user experience.

1.3.1. Existing solutions

Upwork: Upwork is one of the largest freelancing platforms, providing businesses access to skilled professionals in fields such as programming, writing, graphic design, and marketing. It operates on a proposal-based system where freelancers can apply for job postings, and clients can either accept bids or invite freelancers directly for collaboration. The platform is known for facilitating both short-term projects and long-term work relationships.

As shown in Figure 1, Upwork's interface features a well-organized dashboard that allows freelancers to browse job postings, track ongoing projects, manage contracts, and communicate with clients. Employers benefit from tools that enable secure payment processing, time tracking, and milestone-based project management. Additionally, the built-in review and rating system helps freelancers establish credibility while allowing clients to make informed hiring decisions.

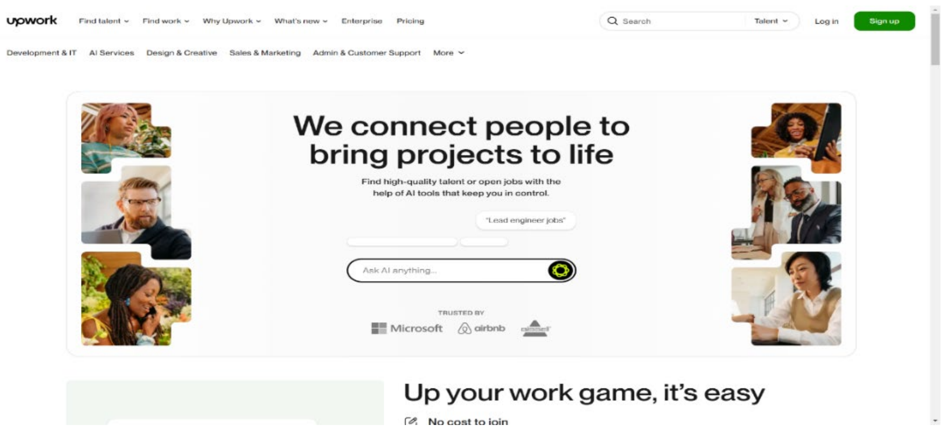


Figure 1 : Interface of Upwork[1]

With its structured workflow and global talent pool, Upwork serves as an ideal platform for professionals seeking stable freelance work and businesses looking for reliable expertise.

Fiverr: Fiverr is a popular freelancing platform designed for quick, fixed-price transactions, where freelancers, known as "sellers," create service listings called "gigs." These gigs cover a wide range of industries, including digital marketing, video editing, writing, and graphic design. Unlike traditional freelancing marketplaces, Fiverr eliminates the bidding process, allowing clients to browse and purchase services instantly.

As illustrated in Figure 2, Fiverr's interface simplifies the hiring process with its search-friendly layout, enabling clients to filter services by category, price range, and seller rating. Each gig listing includes a detailed description, turnaround time, pricing tiers, and client reviews. The platform also features additional services such as gig extras, where sellers can offer premium add-ons for customized solutions.

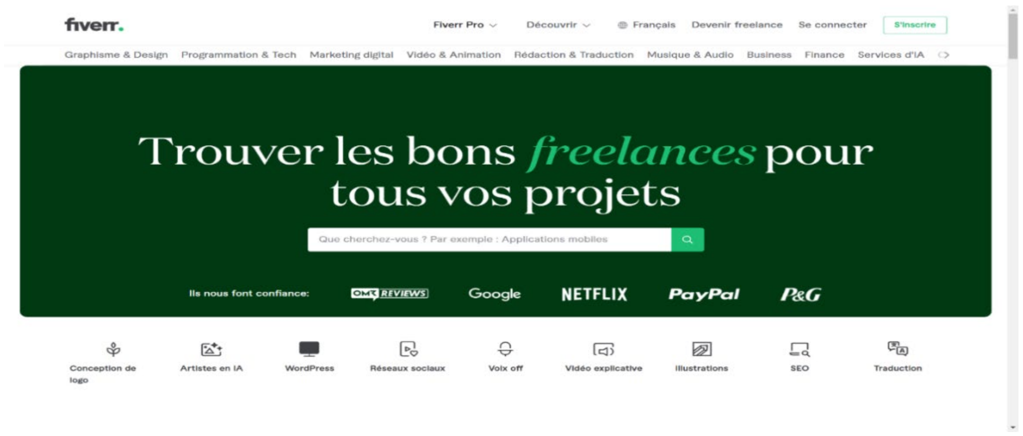


Figure 2 : Interface of Fiverr[2]

[1] https://www.upwork.com
[2] https://www.fiverr.com

With its user-friendly design, tiered pricing system, and instant hiring process, Fiverr is an excellent choice for individuals and businesses seeking fast, affordable services without long negotiations.

Freelancer: Freelancer is an Australian-based freelancing platform that connects businesses with professionals through a competitive bidding system. Employers post job listings across various fields, including software development, data entry, content writing, and engineering. Freelancers then submit proposals, competing based on experience, expertise, and price. This system allows clients to select the best candidate while keeping costs flexible.

As depicted in Figure 3, Freelancer's interface features a dynamic job marketplace where users can monitor bids, interact with clients, and manage projects efficiently. The platform supports multiple work models, including fixed-price contracts, hourly billing, and milestone-based payments. Additionally, Freelancer provides built-in tools for secure transactions, time tracking, and dispute resolution, ensuring a smooth workflow for both parties.

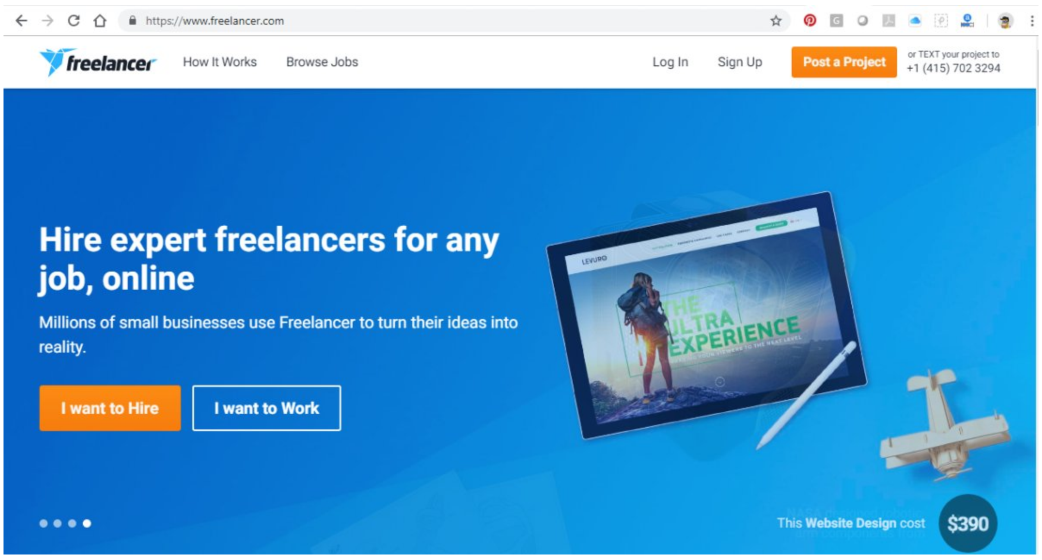


Figure 3 : Interface of Freelancer[3]

By offering a bidding system and flexible payment options, Freelancer is well-suited for professionals looking to secure projects based on merit and pricing while giving businesses access to a global network of skilled workers.

[3] https://www.freelancer.com

1.3.2. Evaluation of existing applications

In this part, we will analyze and critique the freelance platforms selected earlier. These digital platforms offer a multitude of features. Based on the following criteria: diversity of offers, advice and user opinions, we carried out a comparative study, as illustrated in Table 1. This analysis will allow us to identify missing features**.**

Table 1 : Comparison between freelance platforms

Criterion	**Upwork**	**Fiverr**	**Freelancer**
Wide choice of missions	X	X	X
Tips for freelancers	X		X
Rating system and customer reviews	X	X	X
Integrated training system			
Interface intuitive	X	X	
Presence of skills tests	X		X
Offers adapted according to experience	X		
Advanced Assignment Filtering	X		X
Powerful mobile app	X	X	X
Payment protection	X	X	X

This table shows that most of the existing platforms lack the following:

- A built-in training system to help freelancers improve their skills**.**
- Advanced filtering of missions on certain platforms, limiting customization.
- Offers adapted according to experience to help new freelancers get started.

However, they provide a vast collection of offers, an efficient rating system, as well as secure payments, which makes them attractive to self-employed workers.

1.4. Work to be done

To address the various challenges previously mentioned, we will design two main components: a **relational database** and an **interactive web application**. The following outlines the step-by-step process of how the platform works:

FreeConnect operates by connecting skilled professionals, known as freelancers, with clients who need specific tasks or projects completed. The process begins with both freelancers and clients registering on the platform and creating detailed profiles. **Freelancers** typically showcase their expertise by listing their skills, previous work experience, and a portfolio of completed projects. They may also set their hourly rates or fixed prices for specific services. On the other hand, **clients** provide an overview of the company they represent and the industry in which it operates.

Once profiles are set up, **clients post project descriptions** on the platform. These job listings include important details such as project initiation date, deadline, budget range, and any specific requirements or expectations. **Freelancers then browse these listings** and submit tailored proposals for the projects they are interested in. A proposal usually contains a personalized message explaining why the freelancer is a good fit for the job, their approach to completing the project, estimated timelines, and their proposed fees, offering competitive prices to secure the job.

After receiving proposals, **clients review these submissions** by evaluating the freelancers' profiles, portfolios, previous client reviews, and proposed terms. They may conduct interviews through the built-in messaging service to assess a freelancer's suitability further.

Once the ideal freelancer or team of freelancers is selected, the **project execution phase** follows, where the freelancer(s) work on the assigned tasks according to the agreed timeline. **Upon project completion, the freelancer submits the final deliverables** for the client's review. Once the work is approved, the **payment process** is triggered.

The final step in the process is the **feedback and rating system**, where the client posts reviews based on his experience working with the freelancer. The client provides a commentary along with a rating for both the completed project and the freelancer. This feedback contributes to the freelancer's portfolio and enhances their professional credibility, making it easier for him to secure future projects, while the client, acknowledging a good work, benefit by establishing a good reputation that may attract high-quality freelancers for subsequent jobs. This step-by-step process ensures that the website provides a structured, secure, and efficient environment for both freelancers and clients to collaborate successfully.

Conclusion

This chapter has presented the overall context of the project, evaluated existing solutions, and clarified the primary objectives of the FreeConnect platform. The next chapter will focus on an in-depth analysis and a detailed specification of the platform’s requirements.

Chapter 2. Detailed study

Introduction

The objective of this chapter is to present the simplified data dictionary, the business rules and the conceptual data model (CDM). This model is based on the description of the entities, the justification of the entities and relationships, and the standardization check. We will also develop the data logic model.

2.1. Data collection and analysis

We conducted an in-depth analysis of the available descriptions to identify a relevant dataset. This analysis resulted in the creation of an initial data dictionary by carefully examining the records provided. Then, we refined this dictionary by eliminating synonyms, polysemy and calculated data, keeping only the fundamental data.

The clean data dictionary of our case study is given in Table 2.

Table 2 : Data dictionary

Code	**Designation**	**Data type & length**	**Constraints & Rules**
User_ID	User identifier	Numeric (auto-increment)	Unique, not null
User_First_Name	User's first name	Variable character (50)	Not null
User_Last_Name	User's last name	Variable character (50)	Not null
User_Email	User’s email	Variable character(100)	Unique, valid format
User_Location	User’s location	Variable character(100)	
User_Birthday	User’s birthday	Date	
User_Password	User’s password	Variable character(100)	

User_Num_Tel	User’s phone number	Numeric	Unique, valid format
User_Photo	User’s photo	Image	
Free_Portfolio_Link	URL that directs to an online showcase of a freelancer’s work	Variable character(100)	Valid format
Free_Hourly_Rate	The amount a freelancer charges per hour for their services.	Decimal(10,2)	
Free_Experience	Years of experience of a freelancer	Digital	≥ 0
Free_Status	The freelancer can pause or resume their working status on the platform	Variable character (10)	Values: "active", "inactive"
Free_Level	Categorization of a freelancer’s expertise	Variable character (50)	Values: "beginner", "intermediate", "expert"
Company_ID	Identifier of the company the customer works for	Numeric (auto-increment)	Unique, not null
Company_Name	Company’s name	Variable character (100)	Not null
Company_Industry	Company’s industry/sector	Variable character (100)	Optional
Project_ID	Identifier of a project	Numeric (auto-increment)	Unique, not null
Project_Title	Project Title	Variable character (100)	Not null
Project_Description	Detailed description of the project	Text	
Project_Budget	Budget of a project	Numeric (Decimal)	≥ 0

Project_StartDate	Project start date	Date	≥ current date
Project_Deadline	Submission deadline of a project	Date	≥ project start date
Project_Status	Status of a project (Open/Ongoing/Completed)	Variable (20)	Values: "open", "in progress", "completed"
Proposal_ID	Identifier of a freelancer formal offer to a client with a project, outlining their approach, timeline and cost	Numeric (auto-increment)	Unique, not null
Proposal_Amount	Proposed amount to do a project	Numeric (Decimal)	≥ 0
Proposal_ Deadline	Time estimated by the freelancer to complete a project	Date	≥ current date
Proposal _Status	Status of a proposal (Pending/ Accepted / Rejected)	Variable character (20)	Values: "pending", "accepted", "rejected"
Proposal_Description	Proposal's content	Text	
Payment_ID	Payment identifier	Numeric (auto-increment)	Unique, not null
Payment_Amount	Amount transferred	Numeric (Decimal)	≥ 0
Payment_Date	Payment date	Date	Non-zero
Payment_Method	Method used (CB, PayPal, etc.)	Variable (30)	Non-zero
Evaluation_ID	Identifier of a review provided by the client to a project or a freelancer	Numeric (auto-increment)	Unique, not null
Evaluation_Score	Score awarded (1 to 5)	Digital	1 ≤ score ≤ 5

Evaluation_Comment	Related commentary of a review	Text	Optional
Evaluation_Date	Date and time of the evaluation	Date	
Notification_ID	Identifier of a notification to a user of the platform	Numeric (auto-increment)	Unique, not null
Notification_Type	Type (Proposal, Payment, Message)	Variable character (30)	Predefined values
Notification_Content	Notification content	Text	Non-zero
Notification_Status	Status (read/ not read)	Variable character (10)	Values: "read", "not read"
Skill_ID	Identifier of a skill the freelancer has	Numeric (auto-increment)	Unique, not null
Skill_Name	Skill name	Variable character (50)	Unique, non-zero
Message_ID	Identifier of a chat message between users working on a same project	Numeric (auto-increment)	Unique, not null
Message_Content	Message content	Text	Non-zero
Sent_Date	Date and time the message was sent	Date	Non-zero
Received_Date	Date and time the message is received	Date	Non-zero
Message_Status	Message status (read, unread)	Variable character (10)	Values: "read", "unread"

2.2. Management Rules

In this section, we will describe the rules according to their types.

2.2.1. Definition Rules

In this section, we define the classification rules:

- A project is identified by an ID, title, description, type, budget, and deadline.
- A user is identified by an ID, name, email, phone number, and role (client or freelancer).
- A freelancer is identified by an ID, name, hourly rate, experience, and level.
- A client is identified by an ID, name, and company.
- A proposal is identified by an ID, amount, deadline, and status.
- A payment is identified by an ID, amount, date, and method.
- An evaluation is identified by an ID, rating, comment, and date.
- A notification is identified by an ID, type, content, and status.

2.2.2. Fact Rules

This section describes the fact-based rules:

- A user can be a client or a freelancer.
- A client can post multiple projects.
- A freelancer can submit multiple proposals.
- A project can receive multiple proposals.
- An accepted proposal generates a single payment.
- A completed project receives an evaluation from the client.
- A notification is sent for every new proposal, payment, or message.

2.2.3. Necessity Rules

This section describes necessity-based rules:

- A project must have a defined budget.
- A proposal must have a specified amount.
- A payment must have an associated method.
- An evaluation must include a rating.

2.2.4. Validation Rules

This section describes validation rules:

- The project budget must be greater than or equal to 0.
- The proposal amount must be greater than or equal to 0.
- The evaluation rating must be between 1 and 5.
- The email address of a user must be unique.

2.2.5. Constraint Rules

This section describes constraint-based rules:

- The project start date must be later than the current date.
- The project deadline must be later than the project start date.
- The status of a freelancer can only be "active" or "inactive".
- The status of a project can only be "open", "in progress", or "completed".
- The proposal status can only be "pending", "accepted", or "rejected".
- The notification status can only be "read" or "not read".

2.3. Conceptual Data Model

The CDM of our project is shown in Figure 4.

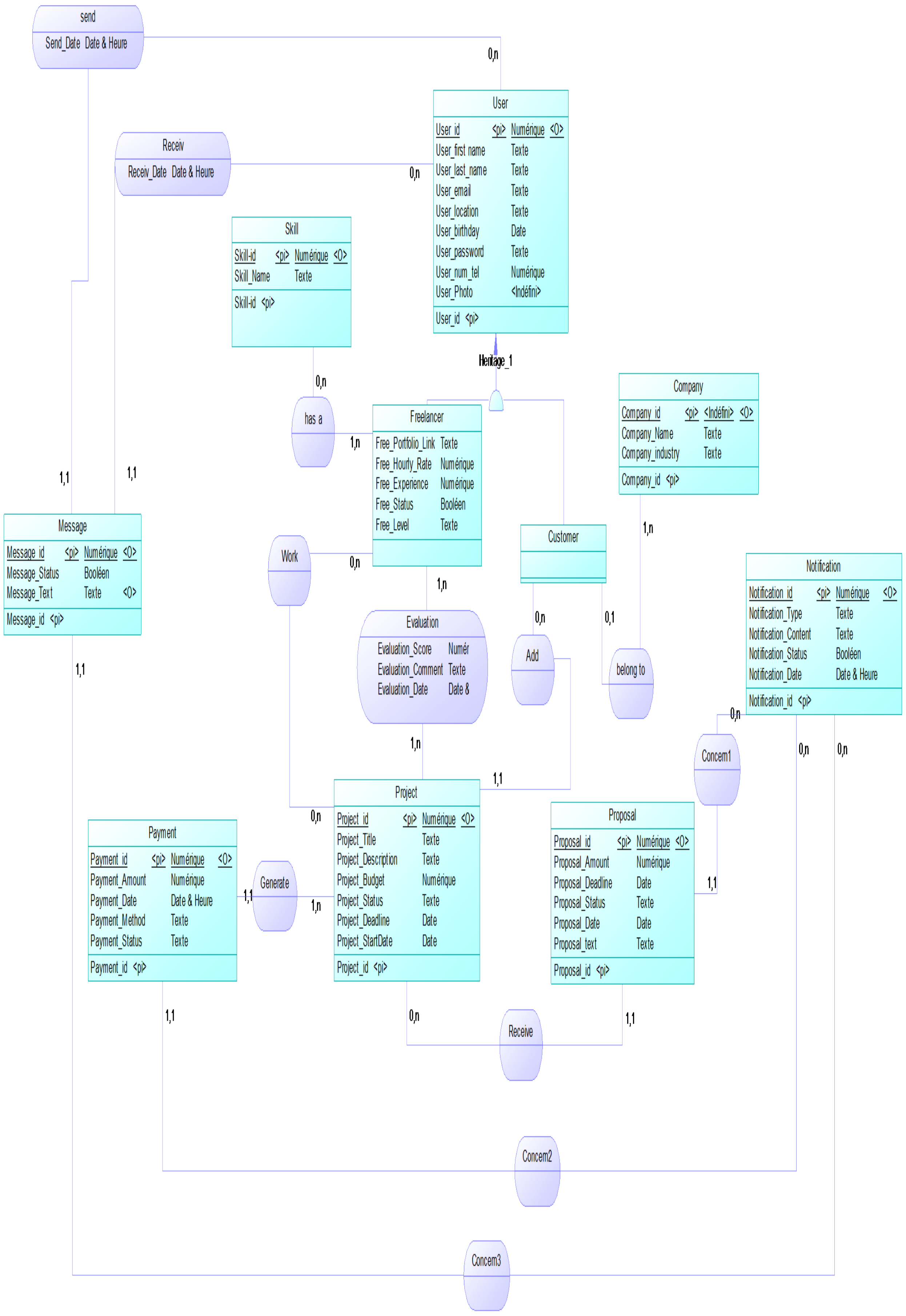
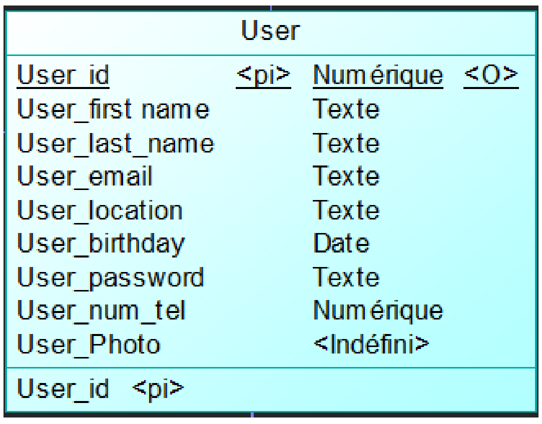
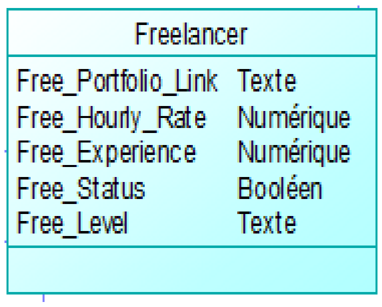
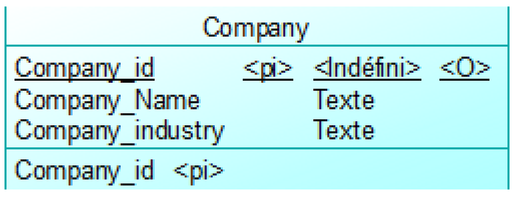


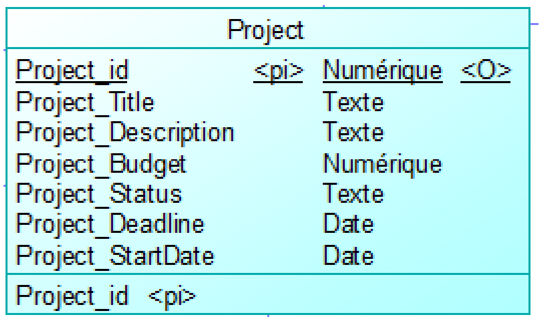
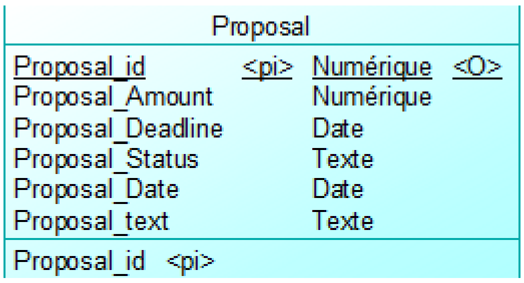
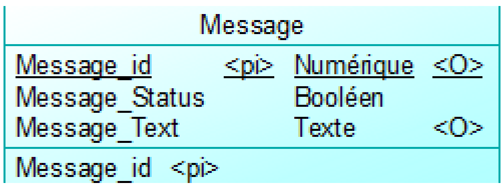
Figure 4 : Conceptual Data Model

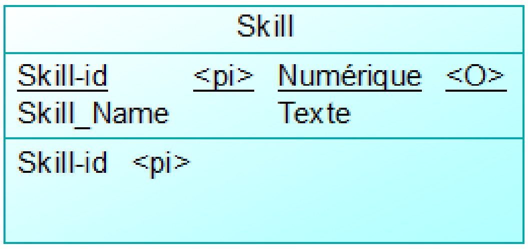
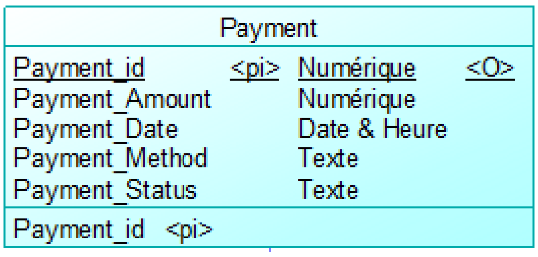
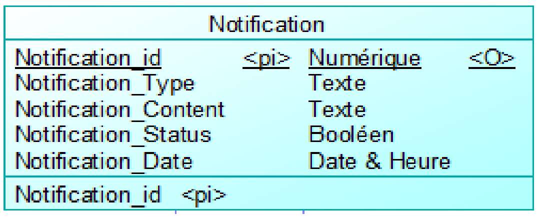
2.3.1. Description of Entities

A detailed description of the entities is elaborated in Table 3.

Table 3 : Description of attributes for each entity

Entity	Description
User  User User_id <pi> Numérique <O> User_first name Texte User_last_name Texte User_email Texte User_location Texte User_birthday Date User_password Texte User_num_tel Numérique User_Photo <Indéfini> User_id <pi> **Functional Dependencies:** User_id → User_first_name, User_last_name, User_email, User_location, User_birthday, User_password, User_num_tel, User_Photo	The 'User' entity describes a candidate through the following attributes: • **User_id**: The user identifier • **User_first_name**: The user's first name • **User_last_name**: The user's last name • **User_email:** The user's email address • **User_location**: The user's location • **User_birthday:** The user's date of birth • **User_password:** The user's password • **User_num_tel**: The user's phone number • **User_Photo**: The user's photo « It is identified by the user identifier. »
Freelancer  Freelancer Free_Portfolio_Link Texte Free_Hourly_Rate Numérique Free_Experience Numérique Free_Status Booléen Free_Level Texte **Functional Dependencies**: User_id → Free_portfolio_url, Free_hourly_rate, Free_experience, Free_Status, Free_Level	The 'Freelancer' entity inherits all attributes from the User entity and has the following additional attributes: **Free_portfolio_url**: Portfolio URL of the freelancer **Free_hourly_rate**: Hourly rate of the freelancer **Free_experience**: Years of experience **Free_Status**: Freelancer's availability status **Free_Level**: Freelancer's expertise level « It is identified by the user identifier. »
The company  Company Company_id <pi> <Indéfini> <O> Company_Name Texte Company_industry Texte Company_id <pi> **Functional Dependencies:**	The 'Company' entity describes a candidate through the following attributes: **Company_id**: The company identifier **Company_Name**: The company name **Company_industry**: The company's industry sector « It is identified by the company identifier. »

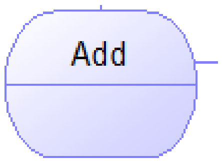
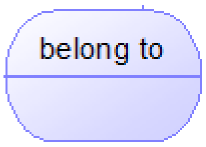
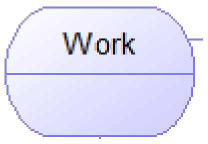
Company_id → Company_Name, Company_industry	
Project  Project Project_id <pi> Numérique <O> Project_Title Texte Project_Description Texte Project_Budget Numérique Project_Status Texte Project_Deadline Date Project_StartDate Date Project_id <pi> **Functional Dependencies**: Project_id → Project_Title, Project_Description, Project_Budget, Project_Status, Project_Deadline, Project_StartDate	The 'Project' entity describes a candidate through the following attributes: **Project_id**: The project identifier **Project_Title**: The project title **Project_Description**: The project description **Project_Budget**: The project budget **Project_Status:** The current status of the project **Project_Deadline**: The project deadline **Project_StartDate**: The project start date « It is identified by the project identifier. »
Proposal  Proposal Proposal_id <pi> Numérique <O> Proposal_Amount Numérique Proposal_Deadline Date Proposal_Status Texte Proposal_Date Date Proposal_text Texte Proposal_id <pi> **Functional Dependencies:** Proposal_id → Proposal_Description, Proposal_Deadline, Proposal_Status, Proposal_Date, Proposal_Text	The 'Proposal' entity describes a candidate through the following attributes: **Proposal_id**: The proposal identifier **Proposal_Description**: The proposal description **Proposal_Deadline**: The proposal deadline **Proposal_Status**: The current status of the proposal **Proposal_Date**: The date the proposal was submitted **Proposal_Text**: The proposal content « It is identified by the proposal identifier. »
Message  Message Message_id <pi> Numérique <O> Message_Status Booléen Message_Text Texte <O> Message_id <pi> **Functional Dependencies:** Message_id → Message_Status, Message_Text	The 'Message' entity describes a candidate through the following attributes: **Message_id**: The message identifier **Message_Status**: The message status **Message_Text**: The message content « It is identified by the message identifier. »
Skill	The 'Skill' entity describes a candidate through the following attributes: **Skill_id**: The skill identifier **Skill_Name**: The name of the skill « It is identified by the skill identifier. »

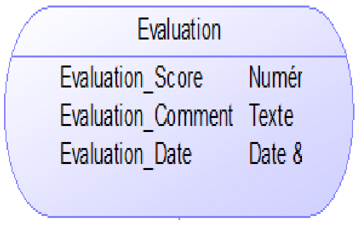
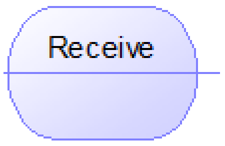
 Skill Skill-id <pi> Numérique <O> Skill_Name Texte Skill-id <pi> **Functional Dependencies:** Skill_id → Skill_Name	
Payment  Payment Payment_id <pi> Numérique <O> Payment_Amount Numérique Payment_Date Date & Heure Payment_Method Texte Payment_Status Texte Payment_id <pi> **Functional Dependencies:** Payment_id → Payment_Amount, Payment_Date, Payment_Method, Payment_Status	The 'Payment' entity describes a candidate through the following attributes: **Payment_id**: The payment identifier **Payment_Amount**: The payment amount **Payment_Date**: The date of payment **Payment_Method**: The payment method **Payment_Status**: The current status of the payment « It is identified by the payment identifier. »
Notification  Notification Notification_id <pi> Numérique <O> Notification_Type Texte Notification_Content Texte Notification_Status Booléen Notification_Date Date & Heure Notification_id <pi> **Functional Dependencies:** Notification_id → Notification_Type, Notification_Content, Notification_Status, Notification_Date	The 'Notification' entity describes a candidate through the following attributes: **Notification_id**: The notification identifier **Notification_Type**: The type of notification **Notification_Content**: The notification content **Notification_Status**: The status of the notification **Notification_Date**: The date of the notification « It is identified by the notification identifier. »

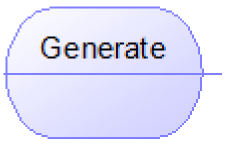
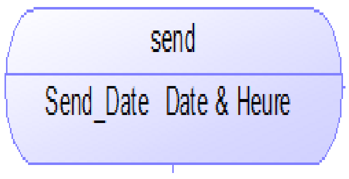
2.3.2. Description of relations and cardinalities

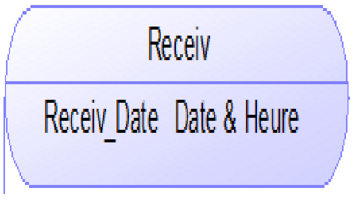
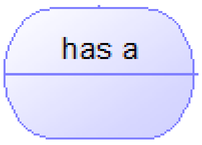
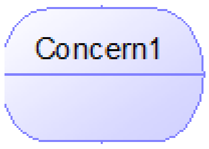
Relations represent the links between entities and are defined by cardinalities. Cardinalities indicate the minimum and maximum number of times an occurrence of an entity can participate in an association. They are properties of entity classes and help characterize the links between an entity and the relation to which it is connected. Table 4 provides a detailed description of relations and cardinalities.

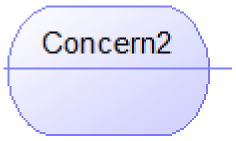
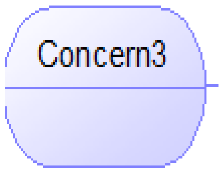
Table 4 : Description of relations

Relation	Entities	Cardinalities	Description
Add 	Customer-Project	0,n – 1,1	The relation **"Add"** has no attributes of its own and has a dimension of 2. It connects the **customer** and the **project**. It is characterized by the cardinality pair **(0,n) - (1,1)**, since a customer can add **zero or multiple projects**. However, a project can only be added by **one customer**.
Belong to 	Customer-Company	0,1 – 1,n	The relation **"Belong to"** has no attributes of its own and has a dimension of 2. It connects the **customer** and the **company**. It is characterized by the cardinality pair **(0,1) - (1,n)**, since a customer belongs to **zero or one company**. However, a company can employ **one or multiple customers**.
Work 	Freelancer-Project	0,n – 0,n	The relation **"Work"** has no attributes of its own and has a dimension of 2. It connects the **freelancer** and the **project**.

			It is characterized by the cardinality pair **(0,n) - (0,n)**, since a freelancer can work on **zero or multiple projects**. However, a project can be done by **zero or multiple freelancers**.
Evaluation 	Freelancer-Project	1,n – 1,n	The relation **"Evaluation"** has a dimension of 2 and contains three attributes : **Evaluation_Score**: The evaluation score **Evaluation_Comment**: The evaluation comment **Evaluation_Date**: The time of the evaluation. It connects the **freelancer** and the **project**. It is characterized by the cardinality pair **(1,n) - (1,n)**, since a freelancer can be evaluated on **one or multiple projects**. However, a project can evaluate **one or multiple freelancers**.
Receive 	Project-Proposal	0,n – 1,1	The relation **"Receive"** has no attributes of its own and has a dimension of 2. It connects the **project** and the **proposal**.

			It is characterized by the cardinality pair **(0,n) - (1,1)**, since a project can receive **zero or multiple proposals**. However, a proposal can only be sent to **one project**.
Generate 	Project-Payment	1,n – 1,1	The relation **"Generate"** has no attributes of its own and has a dimension of 2. It connects the **project** and the **payment**. It is characterized by the cardinality pair **(1,n) - (1,1)**, since a project can generate **one or multiple payments**. However, a payment can only be generated by **one project**.
Send 	User-Message	0,n – 1,1	The relation **"Send"** has a dimension of 2 and contains the attribute **"Send_Date"**, which represents the time the message was sent. It connects the **user** and the **message**. It is characterized by the cardinality pair **(0,n) - (1,1)**, since a user can send **zero or multiple messages**. However, a message can only be sent by **one user**.

Receiv 	User-Message	0,n – 1,1	The relation **"Receiv"** has a dimension of 2 and contains the attribute **"Receiv_Date"** which represents the time the message is received. It connects the **user** and the **message**. It is characterized by the cardinality pair **(0,n) - (1,1)**, since a user can receive **zero or multiple messages**. However, a message can only be received by **one user**.
Has a 	Freelancer-Skill	1,n – 0,n	The relation **"Has a"** has no attributes of its own and has a dimension of 2. It connects the **freelancer** and the **skill**. It is characterized by the cardinality pair **(1,n) - (0,n)**, since a freelancer can have **one or multiple skills**. However, a skill can be possessed by **zero or multiple freelancers**.
Concern1 	Notification-Proposal	0,n – 1,1	The relation **"Concern1"** has no attributes of its own and has a dimension of 2. It connects the **notification** and the **proposal**.

			It is characterized by the cardinality pair **(0,n) - (1,1)**, since a notification can concern **zero or multiple proposals**. However, a proposal can only be concerned by **one notification**.
Concern2  	Notification-Payment	0,n – 1,1	The relation **"Concern2"** has no attributes of its own and has a dimension of 2. It connects the **notification** and the **payment**. It is characterized by the cardinality pair **(0,n) - (1,1)**, since a notification can concern **zero or multiple payments**. However, a payment can only be concerned by **one notification**.
Concern3  	Notification-Message	0,n – 1,1	The relation **"Concern3"** has no attributes of its own and has a dimension of 2. It connects the **notification** and the **message**. It is characterized by the cardinality pair **(0,n) - (1,1)**, since a notification can concern **zero or multiple messages**. However, a message can only be concerned by **one notification**.

2.4. Verification of normalization

Our goal is to develop a verified and normalized model in the third normal form. To do this, we will define each of the normal forms and apply them to our model.

First Normal Form (1NF): All attributes of the entity must contain atomic values, meaning they cannot be multiple or decomposable.

Second Normal Form (2NF): It satisfies 1NF, and all attributes of each entity fully depend on the primary key.

Third Normal Form (3NF): It is verified if the 2NF is met and there are no non-trivial functional dependencies between attributes.

Verification on the User Entity:

- ✓ **First Normal Form:**

  From the *User* entity, we verify that all attributes contain atomic values, meaning they are not multiple or decomposable. The entity is therefore in 1NF.

- ✓ **Second Normal Form:**

  We analyze the following functional dependencies:

  User_ID → *User_Name*

  User_ID → *User_FirstName*

  User_ID → *User_Email*

  User_ID → *User_Password*

  User_ID → *User_Photo*

  User_ID → *User_BirthDate*

  All attributes fully depend on the primary key User_ID, so the entity is in 2NF.

- ✓ **Third Normal Form:**

  Since the *User* entity satisfies 2NF and there are no functional dependencies between non-key attributes, it is in 3NF.

We apply the same verification to the other entities as shown in Table 5:

Table 5 : Verification of normalization

Entity	**1NF**	**2NF**	**3NF**
User	✓	✓	✓
Freelancer	✓	✓	✓
Project	✓	✓	✓
Company	✓	✓	✓
Notification	✓	✓	✓
Payment	✓	✓	✓
Message	✓	✓	✓
Proposal	✓	✓	✓

Thus, all entities are normalized up to the third normal form (3NF).

2.5. Logical Data Model

The Logical Data Model (LDM) is an intermediate step used to transition from the CDM, which is a semantic model, to a physical representation of the data. Using PowerAMC, we generated the LDM presented in Figure 5.

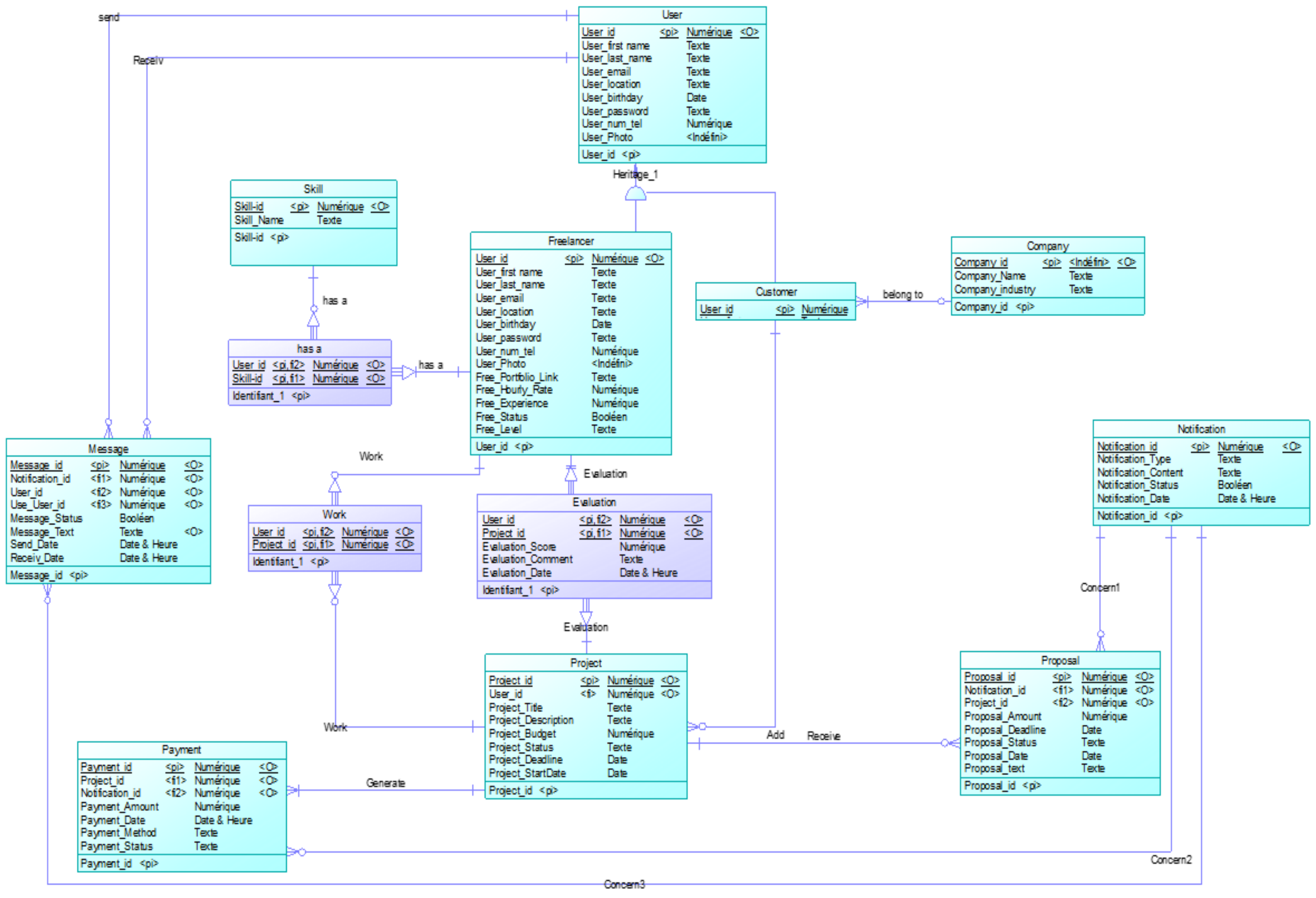


Figure 5 : Logical Data Model

Conclusion

This chapter was dedicated to a detailed study of the database design for our project. We addressed several key aspects, including the presentation of the cleaned and filtered data dictionary, the identification of business rules by type, the conceptual data modeling (CDM), and the logical data modeling (LDM).

The next chapter is devoted to the technical study of the requirements.

Chapter 3. Technical study

Introduction

In this chapter, we will describe the hardware and software environment, derive the Physical Data Model (PDM), and create the database based on the LDM.

3.1. Hardware environment

This project was carried out using a computer with the specifications detailed in Table 6.

Table 6 : Hardware environment

Specification	PC
Processor	Intel CORE i5
RAM	16 GB
Operating system	Windows 11 Professional

3.2. Software environment

The software tools and technologies used for the development of this application are listed in Table 7:

Table 7 : Software environment

Software	Logo	Description
Visual Studio Code		Visual Studio Code is a streamlined code editor with support for development operations like debugging, task running, and version control. It aims to provide just the tools a developer needs for a quick code-build-debug cycle [1].
Angular		Angular is a development platform, built on TypeScript. It includes a

		component-based framework for building scalable web applications [2].
Node.js		Node.js is an open-source and cross-platform JavaScript runtime environment [3].
Express.js		Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications [4].
MongoDB		MongoDB is a document database with the scalability and flexibility that you want with the querying and indexing that you need [5].
Mongoose		Mongoose provides a straight-forward, schema-based solution to model your application data. It includes built-in type casting, validation and query building [6].
PowerAMC (Version 16.6)		Power AMC is an all-in-one business modeling and metadata management tool designed to document enterprise architecture, published by SAP [7].

3.3. Physical Data Model

A PDM makes it easier to analyze the tables, views and other building blocks of a database including the multidimensional objects needed to power a data warehouse. Compared to the CDM, a PDM is more concrete and gives you a substantial representation of the database' structure as illustrated in Figure 6.

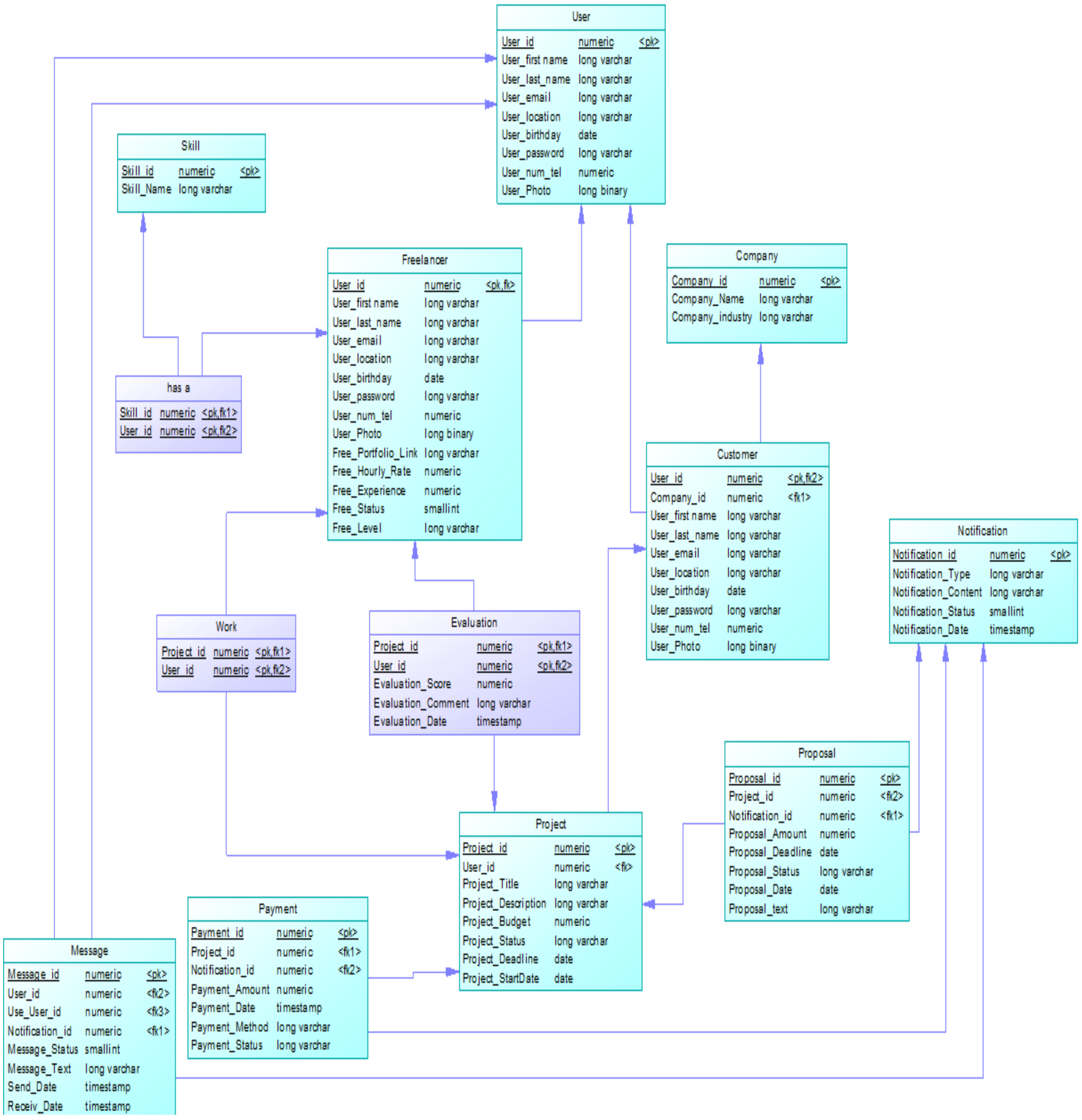
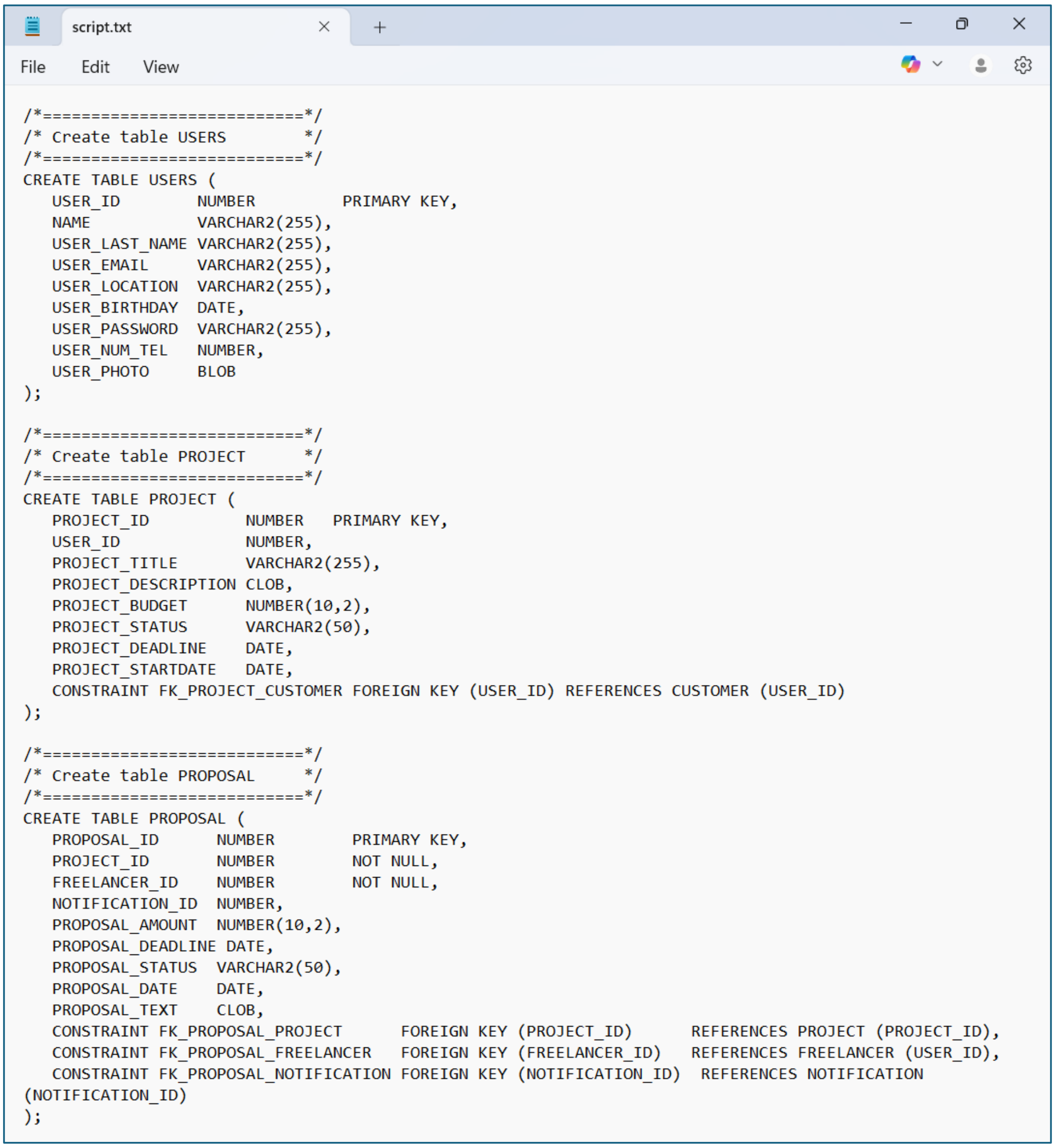


Figure 6 : Physical Data Model

3.4. Database Creation

PowerAMC can generate a database creation script that we can then execute in the Database Management System (DBMS) environment, or directly generate a database structure using a database connection.

The database creation script is successfully generated. It is described in Appendix A. An excerpt is shown in Figure 7.

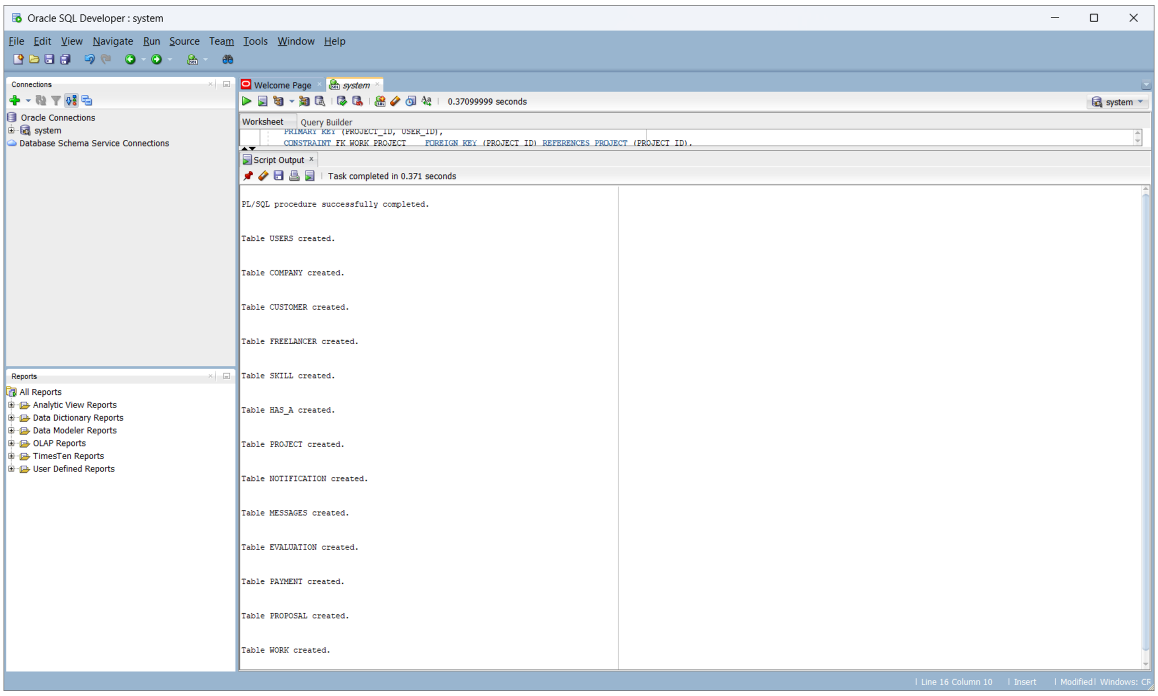


```
/*===========================*/
/* Create table USERS        */
/*===========================*/
CREATE TABLE USERS (
   USER_ID        NUMBER          PRIMARY KEY,
   NAME           VARCHAR2(255),
   USER_LAST_NAME VARCHAR2(255),
   USER_EMAIL     VARCHAR2(255),
   USER_LOCATION  VARCHAR2(255),
   USER_BIRTHDAY  DATE,
   USER_PASSWORD  VARCHAR2(255),
   USER_NUM_TEL   NUMBER,
   USER_PHOTO     BLOB
);

/*===========================*/
/* Create table PROJECT      */
/*===========================*/
CREATE TABLE PROJECT (
   PROJECT_ID          NUMBER    PRIMARY KEY,
   USER_ID             NUMBER,
   PROJECT_TITLE       VARCHAR2(255),
   PROJECT_DESCRIPTION CLOB,
   PROJECT_BUDGET      NUMBER(10,2),
   PROJECT_STATUS      VARCHAR2(50),
   PROJECT_DEADLINE    DATE,
   PROJECT_STARTDATE   DATE,
   CONSTRAINT FK_PROJECT_CUSTOMER FOREIGN KEY (USER_ID) REFERENCES CUSTOMER (USER_ID)
);

/*===========================*/
/* Create table PROPOSAL     */
/*===========================*/
CREATE TABLE PROPOSAL (
   PROPOSAL_ID       NUMBER        PRIMARY KEY,
   PROJECT_ID        NUMBER        NOT NULL,
   FREELANCER_ID     NUMBER        NOT NULL,
   NOTIFICATION_ID   NUMBER,
   PROPOSAL_AMOUNT   NUMBER(10,2),
   PROPOSAL_DEADLINE DATE,
   PROPOSAL_STATUS   VARCHAR2(50),
   PROPOSAL_DATE     DATE,
   PROPOSAL_TEXT     CLOB,
   CONSTRAINT FK_PROPOSAL_PROJECT      FOREIGN KEY (PROJECT_ID)      REFERENCES PROJECT (PROJECT_ID),
   CONSTRAINT FK_PROPOSAL_FREELANCER   FOREIGN KEY (FREELANCER_ID)   REFERENCES FREELANCER (USER_ID),
   CONSTRAINT FK_PROPOSAL_NOTIFICATION FOREIGN KEY (NOTIFICATION_ID)  REFERENCES NOTIFICATION
(NOTIFICATION_ID)
);
```

Figure 7 : Excerpt of the database creation script

ORACLE SQL Developer provides the ability to establish a connection to the database to visualize the existing relationships in the database and to write and execute SQL scripts.

The database creation was successfully completed, as shown in Figure 8.



```
    PRIMARY KEY (PROJECT_ID, USER_ID),
    CONSTRAINT FK_WORK_PROJECT    FOREIGN KEY (PROJECT_ID) REFERENCES PROJECT (PROJECT_ID),
```

```
PL/SQL procedure successfully completed.

Table USERS created.

Table COMPANY created.

Table CUSTOMER created.

Table FREELANCER created.

Table SKILL created.

Table HAS_A created.

Table PROJECT created.

Table NOTIFICATION created.

Table MESSAGES created.

Table EVALUATION created.

Table PAYMENT created.

Table PROPOSAL created.

Table WORK created.
```

Figure 8 : Result of executing the database creation script

3.5. Populating the Database

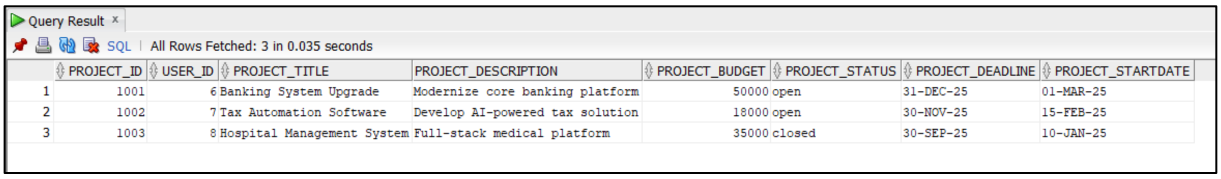
We added tuples to each of the tables in the database. An excerpt is presented in Figure 9.

The complete dataset is provided in Appendix B.

```
INSERT INTO PROJECT VALUES (
    1001, 6, 'Banking System Upgrade',
    'Modernize core banking platform', 50000.00, 'open',
    TO_DATE('2025-12-31', 'YYYY-MM-DD'), TO_DATE('2025-03-01', 'YYYY-MM-DD')
);
INSERT INTO PROJECT VALUES (
    1002, 7, 'Tax Automation Software',
    'Develop AI-powered tax solution', 18000.00, 'open',
    TO_DATE('2025-11-30', 'YYYY-MM-DD'), TO_DATE('2025-02-15', 'YYYY-MM-DD')
);
INSERT INTO PROJECT VALUES (
    1003, 8, 'Hospital Management System',
    'Full-stack medical platform', 35000.00, 'closed',
    TO_DATE('2025-09-30', 'YYYY-MM-DD'), TO_DATE('2025-01-10', 'YYYY-MM-DD')
);
```

Figure 9 : Data in the PROJECT table

The results are presented below in Figure 10.



All Rows Fetched: 3 in 0.035 seconds

	PROJECT_ID	USER_ID	PROJECT_TITLE	PROJECT_DESCRIPTION	PROJECT_BUDGET	PROJECT_STATUS	PROJECT_DEADLINE	PROJECT_STARTDATE
1	1001	6	Banking System Upgrade	Modernize core banking platform	50000	open	31-DEC-25	01-MAR-25
2	1002	7	Tax Automation Software	Develop AI-powered tax solution	18000	open	30-NOV-25	15-FEB-25
3	1003	8	Hospital Management System	Full-stack medical platform	35000	closed	30-SEP-25	10-JAN-25

Figure 10 : Data in the PROJECT table after population

Conclusion

In this chapter, we explored the working environment and the physical data model. Furthermore, we examined in detail the creation and population of the database. In the next chapter, we will address querying the database through SQL queries and via a website.

Chapter 4. Implementation and Deployment

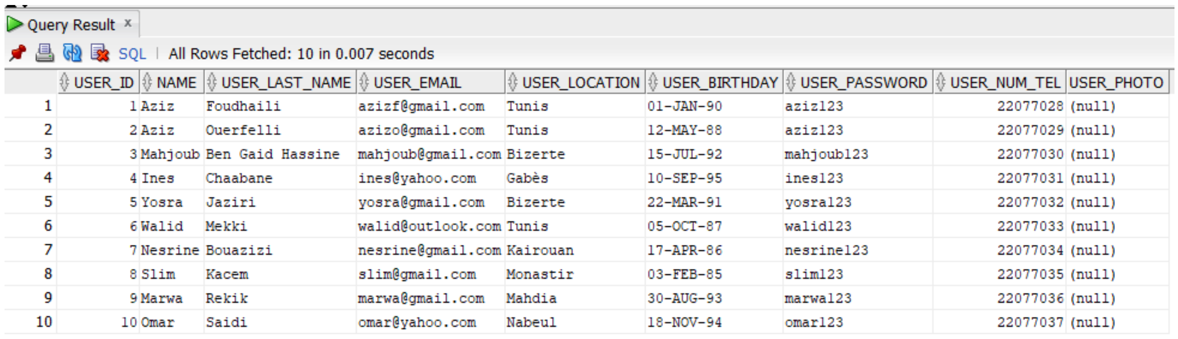
Introduction

This final chapter presents the implementation and deployment of the project. We present the queries, followed by their SQL formulation and the results obtained.

4.1. SQL Query Formulation and Execution

Table 8 presents the queries, their corresponding SQL formulations, and the results obtained from their execution.

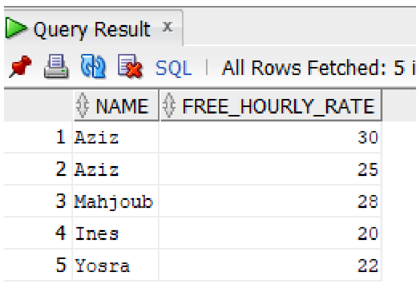
Table 8 : SQL Queries and Results of Their Execution

Q1	Display all users.
SQL	**SELECT * FROM** USERS;
	

Query Result ×

SQL | All Rows Fetched: 10 in 0.007 seconds

	USER_ID	NAME	USER_LAST_NAME	USER_EMAIL	USER_LOCATION	USER_BIRTHDAY	USER_PASSWORD	USER_NUM_TEL	USER_PHOTO
1	1	Aziz	Foudhaili	azizf@gmail.com	Tunis	01-JAN-90	aziz123	22077028	(null)
2	2	Aziz	Ouerfelli	azizo@gmail.com	Tunis	12-MAY-88	aziz123	22077029	(null)
3	3	Mahjoub	Ben Gaid Hassine	mahjoub@gmail.com	Bizerte	15-JUL-92	mahjoub123	22077030	(null)
4	4	Ines	Chaabane	ines@yahoo.com	Gabès	10-SEP-95	ines123	22077031	(null)
5	5	Yosra	Jaziri	yosra@gmail.com	Bizerte	22-MAR-91	yosra123	22077032	(null)
6	6	Walid	Mekki	walid@outlook.com	Tunis	05-OCT-87	walid123	22077033	(null)
7	7	Nesrine	Bouazizi	nesrine@gmail.com	Kairouan	17-APR-86	nesrine123	22077034	(null)
8	8	Slim	Kacem	slim@gmail.com	Monastir	03-FEB-85	slim123	22077035	(null)
9	9	Marwa	Rekik	marwa@gmail.com	Mahdia	30-AUG-93	marwa123	22077036	(null)
10	10	Omar	Saidi	omar@yahoo.com	Nabeul	18-NOV-94	omar123	22077037	(null)

Q2	List freelancers with hourly rate.
SQL	**SELECT** NAME, FREE_HOURLY_RATE **FROM** FREELANCER;
	

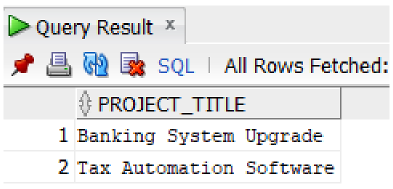
Query Result ×

SQL | All Rows Fetched: 5 i

	NAME	FREE_HOURLY_RATE
1	Aziz	30
2	Aziz	25
3	Mahjoub	28
4	Ines	20
5	Yosra	22

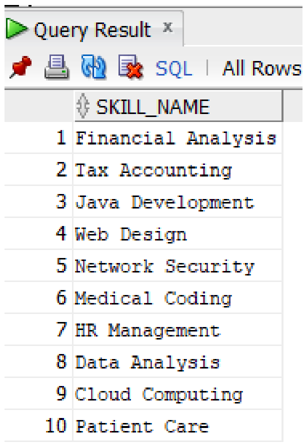
Q3	Show open projects.

SQL	**SELECT** PROJECT_TITLE **FROM** PROJECT
	WHERE PROJECT_STATUS = 'open';



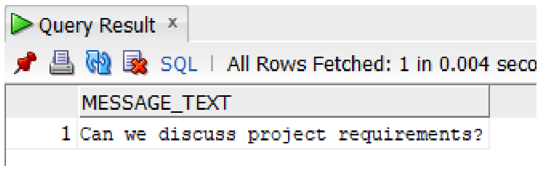
```
Query Result ×
SQL | All Rows Fetched:
   PROJECT_TITLE
 1 Banking System Upgrade
 2 Tax Automation Software
```

Q4	Display all skill names.
SQL	**SELECT** SKILL_NAME **FROM** SKILL;



```
Query Result ×
SQL | All Rows
    SKILL_NAME
  1 Financial Analysis
  2 Tax Accounting
  3 Java Development
  4 Web Design
  5 Network Security
  6 Medical Coding
  7 HR Management
  8 Data Analysis
  9 Cloud Computing
 10 Patient Care
```

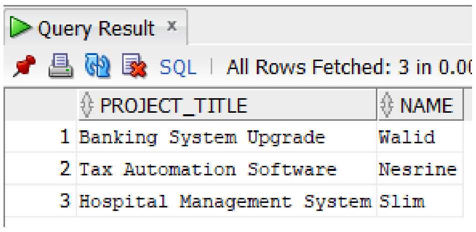
Q5	Show message content.
SQL	**SELECT** MESSAGE_TEXT **FROM** MESSAGES;



```
Query Result ×
SQL | All Rows Fetched: 1 in 0.004 seco
   MESSAGE_TEXT
 1 Can we discuss project requirements?
```

Q6	List projects with their clients.

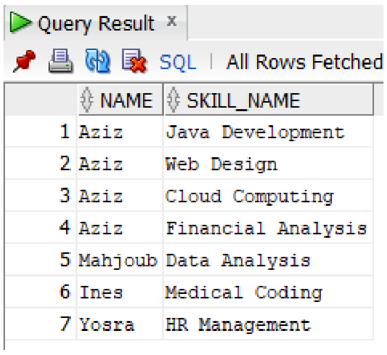
SQL	**SELECT** P.PROJECT_TITLE, C.NAME
	FROM PROJECT P
	JOIN CUSTOMER C **ON** P.USER_ID = C.USER_ID;



```
Query Result ×
SQL | All Rows Fetched: 3 in 0.0
     PROJECT_TITLE              NAME
   1 Banking System Upgrade     Walid
   2 Tax Automation Software    Nesrine
   3 Hospital Management System Slim
```

Q7	Show freelancers with their skills.

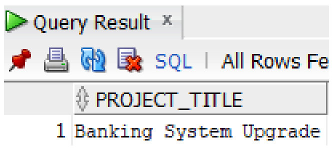
SQL	**SELECT** F.NAME, S.SKILL_NAME
	FROM FREELANCER F
	JOIN HAS_A H **ON** F.USER_ID = H.USER_ID
	JOIN SKILL S **ON** S.SKILL_ID = H.SKILL_ID;



```
Query Result ×
SQL | All Rows Fetched
     NAME    SKILL_NAME
   1 Aziz    Java Development
   2 Aziz    Web Design
   3 Aziz    Cloud Computing
   4 Aziz    Financial Analysis
   5 Mahjoub Data Analysis
   6 Ines    Medical Coding
   7 Yosra   HR Management
```

Q8	Display projects with proposals.

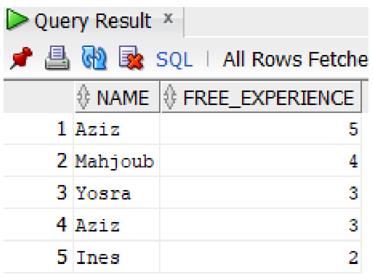
SQL	**SELECT DISTINCT** P.PROJECT_TITLE
	FROM PROJECT P
	JOIN PROPOSAL PR **ON** P.PROJECT_ID = PR.PROJECT_ID;



```
Query Result ×
SQL | All Rows Fe
     PROJECT_TITLE
   1 Banking System Upgrade
```

Q9	List all freelancers from the most to the least experienced.

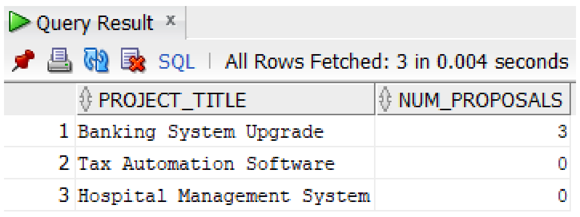
SQL	**SELECT** NAME, FREE_EXPERIENCE
	FROM FREELANCER
	ORDER BY FREE_EXPERIENCE **DESC**;



```
Query Result ×
SQL | All Rows Fetche
   NAME     FREE_EXPERIENCE
1  Aziz                   5
2  Mahjoub                4
3  Yosra                  3
4  Aziz                   3
5  Ines                   2
```

Q10	Show for each project the number of received proposals.

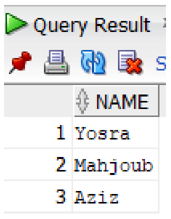
SQL	**SELECT** P.PROJECT_TITLE,
	(**SELECT COUNT**(PR.PROPOSAL_ID)
	FROM PROPOSAL PR
	WHERE PR.PROJECT_ID = P.PROJECT_ID) **AS**
	NUM_PROPOSALS
	FROM PROJECT P;



```
Query Result ×
SQL | All Rows Fetched: 3 in 0.004 seconds
   PROJECT_TITLE                NUM_PROPOSALS
1  Banking System Upgrade                   3
2  Tax Automation Software                  0
3  Hospital Management System               0
```

Q11	List the unevaluated freelancers.

SQL	**SELECT** F.NAME
	FROM FREELANCER F
	WHERE NOT EXISTS (
	SELECT E.USER_ID
	FROM EVALUATION E
	WHERE E.USER_ID = F.USER_ID);

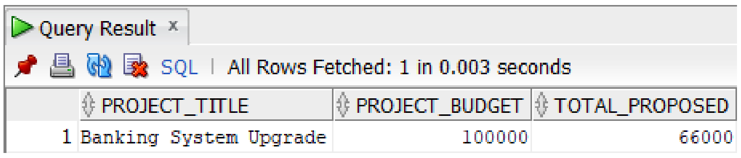


```
Query Result
S
   NAME
1  Yosra
2  Mahjoub
3  Aziz
```

Q12	List all projects with budget grater than the sum of its proposal.

SQL	**SELECT** P.PROJECT_TITLE, P.PROJECT_BUDGET,
	SUM(PR.PROPOSAL_AMOUNT) **AS** TOTAL_PROPOSED
	FROM PROJECT P
	JOIN PROPOSAL PR **ON** P.PROJECT_ID = PR.PROJECT_ID
	GROUP BY P.PROJECT_TITLE, P.PROJECT_BUDGET

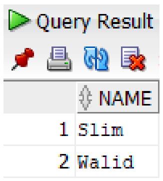
	HAVING P.PROJECT_BUDGET > **SUM**(PR.PROPOSAL_AMOUNT);



```
Query Result ×
SQL | All Rows Fetched: 1 in 0.003 seconds
     PROJECT_TITLE            PROJECT_BUDGET  TOTAL_PROPOSED
   1 Banking System Upgrade           100000           66000
```

Q13	List the clients with payments greater than 1000$.

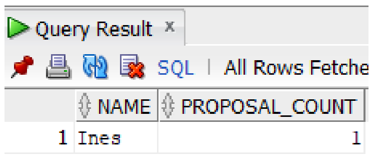
SQL	**SELECT DISTINCT** C.NAME
	FROM CUSTOMER C
	JOIN PROJECT P **ON** C.USER_ID = P.USER_ID
	JOIN PAYMENT PA **ON** PA.PROJECT_ID = P.PROJECT_ID
	WHERE PA.PAYMENT_AMOUNT > 1000;



```
Query Result
     NAME
   1 Slim
   2 Walid
```

Q14	Which freelancers have submitted proposals and have also worked on at least one project? Display their names along with the number of proposals they submitted, ordered by the highest number of proposals.

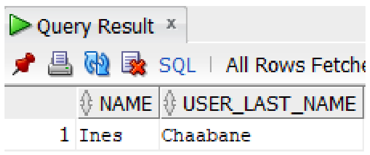
SQL	**SELECT** U.NAME, **COUNT**(PR.PROPOSAL_ID) **AS** PROPOSAL_COUNT
	FROM USERS U
	JOIN FREELANCER F **ON** U.USER_ID = F.USER_ID
	JOIN PROPOSAL PR **ON** F.USER_ID = PR.FREELANCER_ID
	WHERE F.USER_ID **IN** (
	    SELECT DISTINCT USER_ID
	    FROM WORK)
	GROUP BY U.NAME
	ORDER BY PROPOSAL_COUNT **DESC;**



```
Query Result ×
SQL | All Rows Fetche
     NAME  PROPOSAL_COUNT
   1 Ines               1
```

Q15	Find all freelancers who have submitted only proposals to project with ID = 1001, and all those proposals have the status 'rejected'.
SQL	**SELECT** U.NAME, U.USER_LAST_NAME **FROM** USERS U **JOIN** FREELANCER F **ON** U.USER_ID = F.USER_ID

```
WHERE EXISTS (
   SELECT P.PROPOSAL_ID
   FROM PROPOSAL P
   WHERE P.FREELANCER_ID = F.USER_ID
     AND P.PROJECT_ID = 1001
     AND P.PROPOSAL_STATUS = 'rejected' )
AND NOT EXISTS (
   SELECT P2.PROPOSAL_ID
   FROM PROPOSAL P2
   WHERE P2.FREELANCER_ID = F.USER_ID
     AND (P2.PROJECT_ID <> 1001 OR P2.PROPOSAL_STATUS <>
'rejected')
);
```



```
  NAME  USER_LAST_NAME
1 Ines  Chaabane
```

4.2. Web Application Implementation

The final stage of our project involves the implementation of a web application based on the selected technologies. This constitutes the last part of this report and aims to present the overall architecture of the freelance platform, as well as the work accomplished.

4.2.1. Overall Architecture of the Solution

The architecture of our solution is designed to effectively integrate the various components necessary for a fully functional freelance platform, as illustrated in Figure 11. The backend is developed using Node.js and Express, which interact seamlessly with a MongoDB database. We chose MongoDB because it is a document-oriented database, particularly well-suited for storing flexible and unstructured data such as freelancers' CVs. The backend communicates with the frontend, which is built using Angular. This architecture follows the MEAN stack (MongoDB, Express, Angular, Node.js), a full JavaScript-based technology stack that promotes an efficient workflow, a smooth user experience, and effective data management.

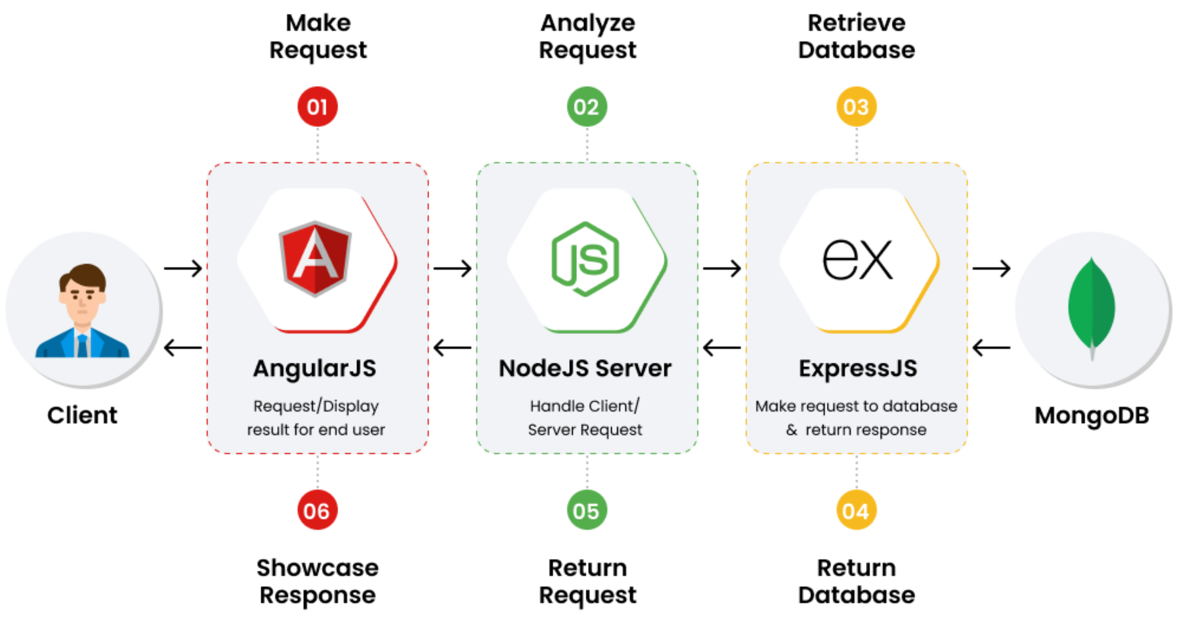
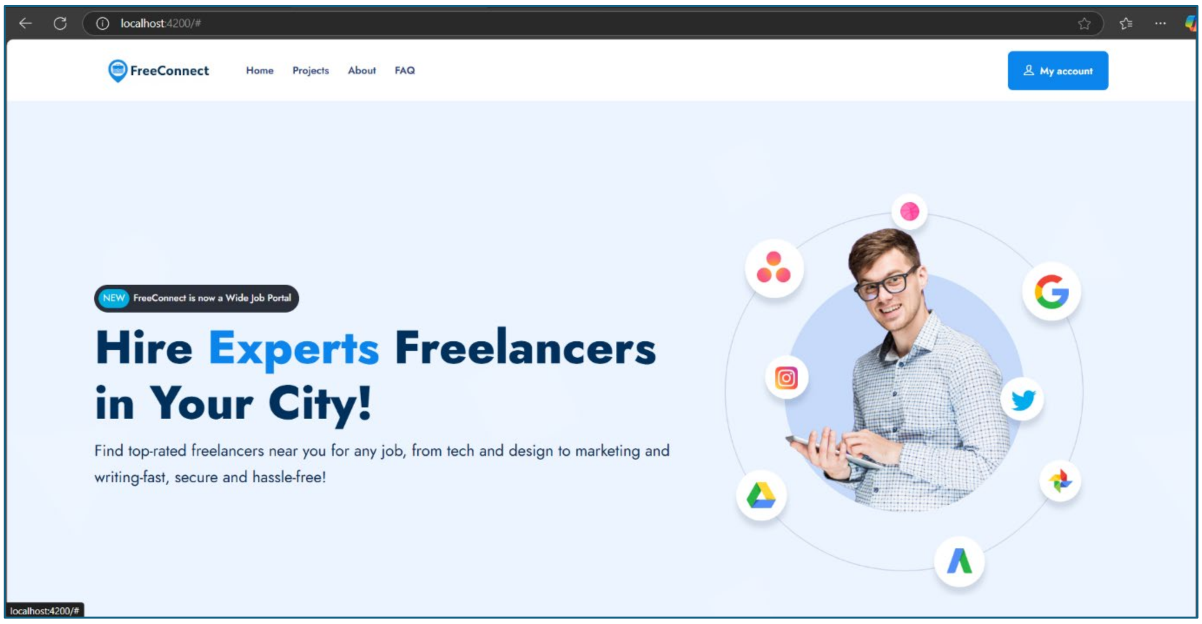


Figure 11 : Overall Architecture of the Application

4.2.2. Main Features

The **homepage** of our application provides a user-friendly and an intuitive experience from the very first visit. Accessible to all, this homepage allows visitors to quickly access all the main features as shown in Figure 12.



Hire Experts Freelancers in Your City!

Find top-rated freelancers near you for any job, from tech and design to marketing and writing-fast, secure and hassle-free!

Figure 12 : Homepage Interface

The **Projects** section presents available services in a clean card-based format that showcases essential details as shown in Figure 13.

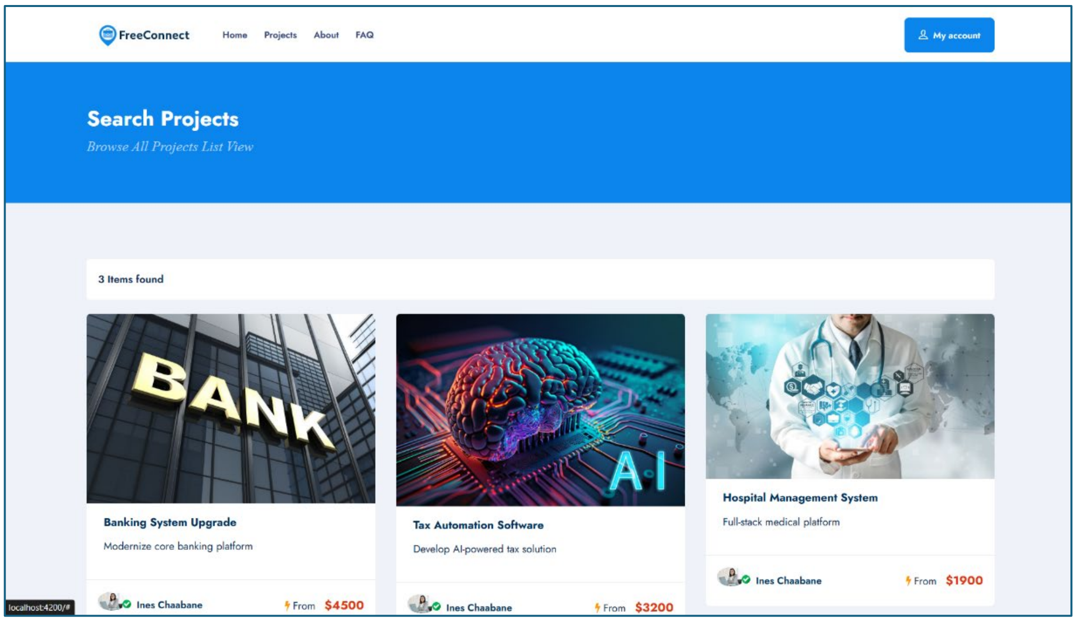


Figure 13 : Projects Interface

The **Dashboard** provides users with a visual overview of their account activity as shown in Figure 14.

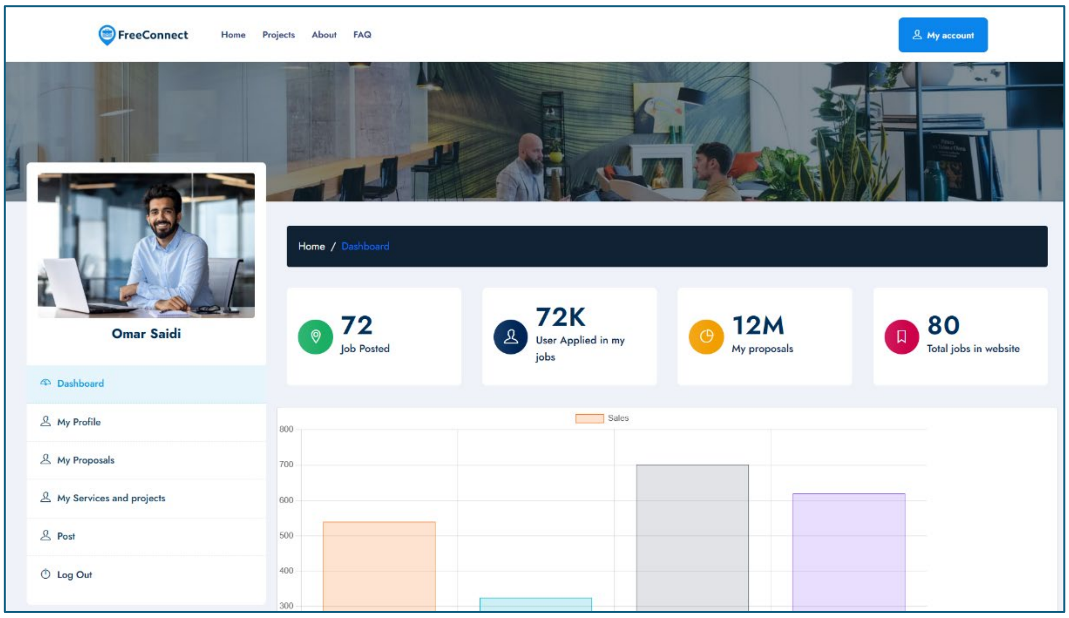


Figure 14 : Dashboard Interface

The **My Services and Projects** section displays the projects posted by a customer. It is populated through the **Post** feature as shown in Figure 15.

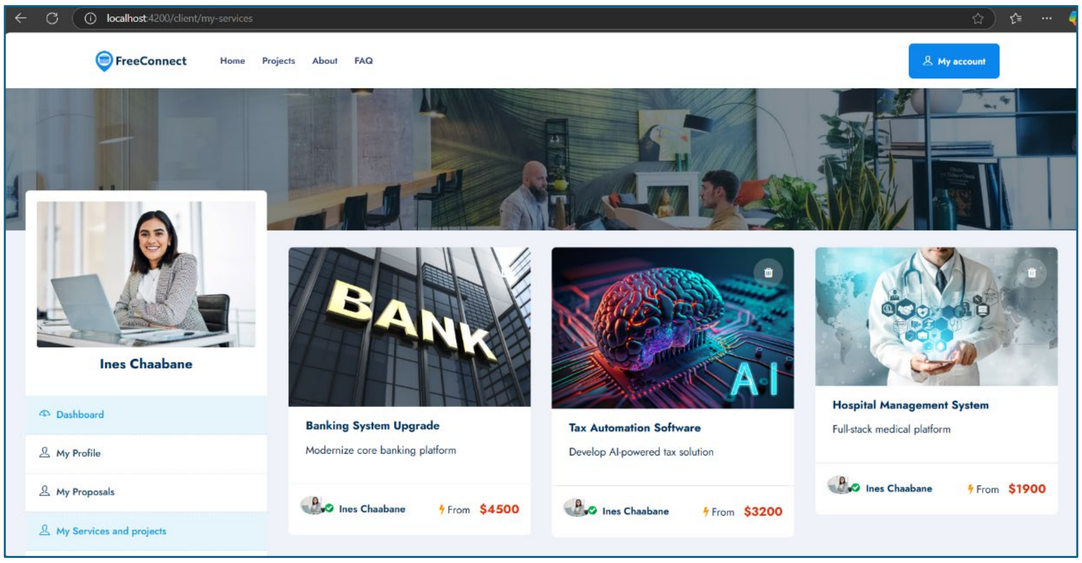


Figure 15 : My Services and Projects Interface

The **My Proposals** section shows all the job proposals submitted by a freelancer. It allows users to track the status of each proposal as shown in Figure 16.

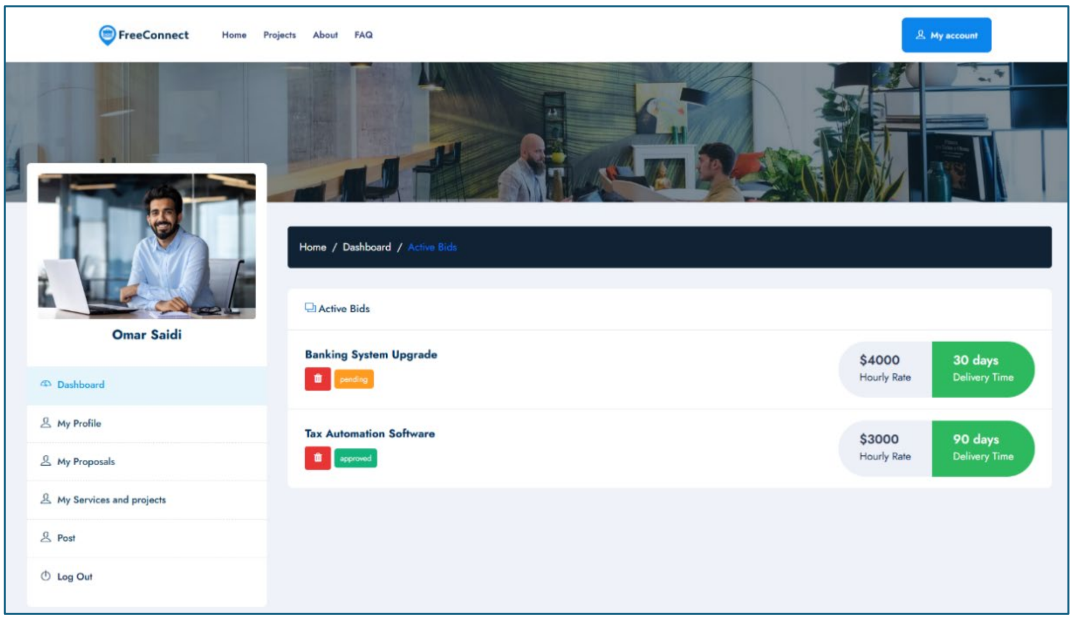


Figure 16 : My proposals Interface

Conclusion

In this chapter, we explored how to query the database using SQL language. For each query, we provided its SQL formulation along with the result of its execution. Finally, we showcased the developed website, highlighting its main features.

General conclusion

This report has presented the development of our freelance platform project, which focus on the design, creation, population and querying of a database, connected to a dynamic web interface.

We began by establishing the general structure of the project, followed by a thorough analysis and conceptual modeling of the data using PowerAMC. We then moved on to the logical and physical modeling phases.

Next, we detailed the database creation and population process using Oracle SQL Developer. This included inserting initial data to simulate real-world scenarios.

Following the database setup, we presented a variety of SQL queries, ranging from basic to more complex levels. For each query, we explained the purpose, provided the SQL formulation, and illustrated the expected results to validate the correctness of the operations.

In addition to working directly with SQL, we developed a web application to offer a dynamic interaction with a MongoDB database. This interface enables users to post and manage projects, submit proposals, and interact through the platform efficiently. The web application was designed to ensure intuitive navigation, responsive performance, and seamless integration with the database, providing a practical demonstration of how front-end and back-end components work together in a real-world system.

This project has been highly valuable to us. It has allowed us to reinforce our theoretical knowledge in database systems and web development and apply it to a real-world scenario.

Looking ahead, several enhancements could be implemented. These include user account management, payment integration, and advanced search and filtering options. Furthermore, we envision developing a mobile version of the platform to enhance accessibility and provide users with more flexibility to manage their freelance activities anytime, anywhere.

Bibliography

[1] Microsoft, “Visual Studio Code FAQ” Microsoft, [online]. Available: https://code.visualstudio.com/docs/supporting/FAQ. [Accessed April 2025].

[2] Angular, “What is Angular?” Angular, [online]. Available: https://v17.angular.io/guide/what-is-angular. [Accessed April 2025].

[3] Nodejs, “Introduction to Node.js” Nodejs, [online]. Available: https://nodejs.org/en/learn/getting-started/introduction-to-nodejs. [Accessed April 2025].

[4] Expressjs, “Web Applications” Express, [online]. Available: https://expressjs.com/. [Accessed April 2025].

[5] MongoDB, “What is MongoDB?” MongoDB, [online]. Available: https://www.mongodb.com/company/what-is-mongodb. [Accessed April 2025].

[6] Mongoosejs, “mongoose” Mongoose, [online]. Available: https://mongoosejs.com/. [Accessed April 2025].

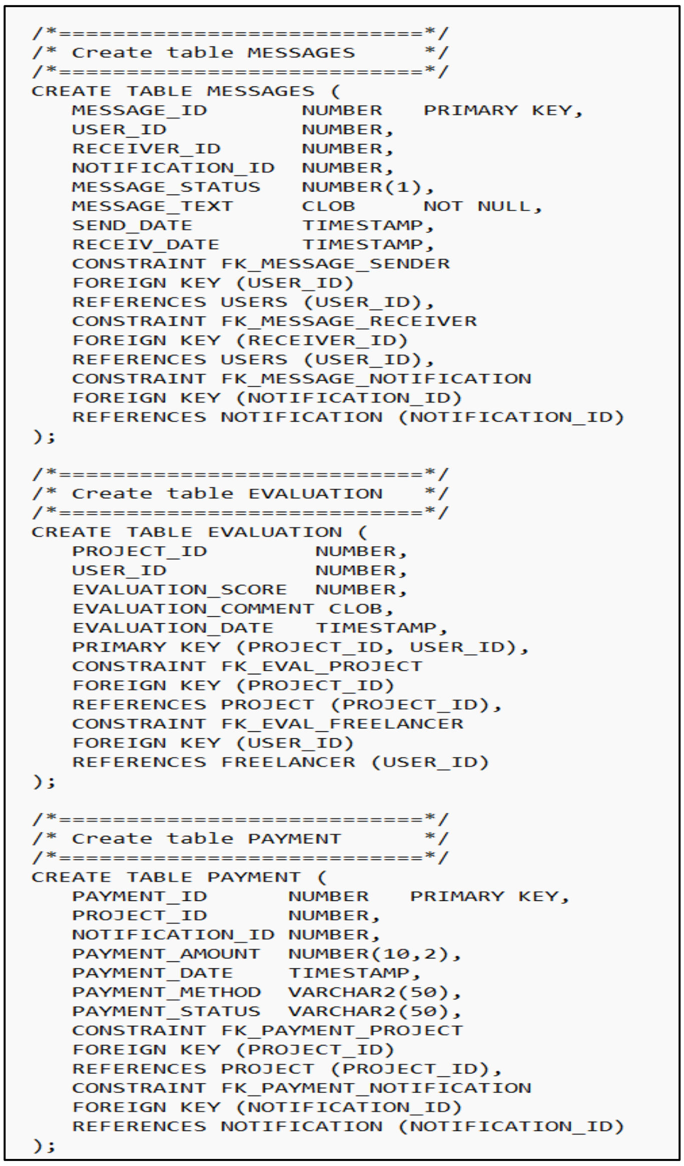
[7] PowerDesigner, “What is SAP PowerDesigner” PowerDesigner, [online]. Available: https://www.powerdesigner.biz/. [Accessed April 2025]

Appendices

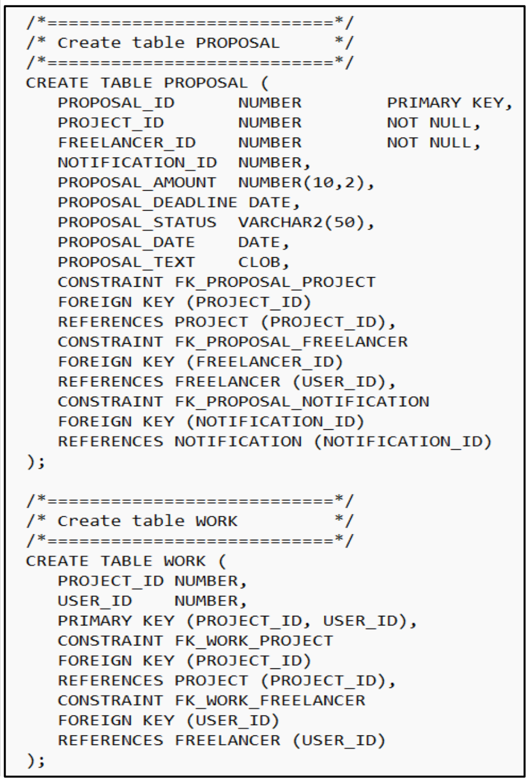
Appendix A : The generated database creation script

```
/* Dropping tables (Oracle syntax) */
BEGIN
   EXECUTE IMMEDIATE 'DROP TABLE WORK';
   EXECUTE IMMEDIATE 'DROP TABLE PROPOSAL';
   EXECUTE IMMEDIATE 'DROP TABLE PAYMENT';
   EXECUTE IMMEDIATE 'DROP TABLE EVALUATION';
   EXECUTE IMMEDIATE 'DROP TABLE PROJECT';
   EXECUTE IMMEDIATE 'DROP TABLE HAS_A';
   EXECUTE IMMEDIATE 'DROP TABLE MESSAGES';
   EXECUTE IMMEDIATE 'DROP TABLE NOTIFICATION';
   EXECUTE IMMEDIATE 'DROP TABLE FREELANCER';
   EXECUTE IMMEDIATE 'DROP TABLE CUSTOMER';
   EXECUTE IMMEDIATE 'DROP TABLE COMPANY';
   EXECUTE IMMEDIATE 'DROP TABLE USERS';
   EXECUTE IMMEDIATE 'DROP TABLE SKILL';
EXCEPTION
   WHEN OTHERS THEN
      NULL;
END;
/

/*============================*/
/* Create table USERS         */
/*============================*/
CREATE TABLE USERS (
   USER_ID          NUMBER            PRIMARY KEY,
   NAME             VARCHAR2(255),
   USER_LAST_NAME   VARCHAR2(255),
   USER_EMAIL       VARCHAR2(255),
   USER_LOCATION    VARCHAR2(255),
   USER_BIRTHDAY    DATE,
   USER_PASSWORD    VARCHAR2(255),
   USER_NUM_TEL     NUMBER,
   USER_PHOTO       BLOB
);

/*============================*/
/* Create table COMPANY       */
/*============================*/
CREATE TABLE COMPANY (
   COMPANY_ID         NUMBER    PRIMARY KEY,
   COMPANY_NAME       VARCHAR2(255),
   COMPANY_INDUSTRY VARCHAR2(255)
);

/*============================*/
/* Create table CUSTOMER      */
/*============================*/
CREATE TABLE CUSTOMER (
   USER_ID          NUMBER    PRIMARY KEY,
   COMPANY_ID       NUMBER,
   NAME             VARCHAR2(255),
   USER_LAST_NAME VARCHAR2(255),
   USER_EMAIL       VARCHAR2(255),
   USER_LOCATION    VARCHAR2(255),
   USER_BIRTHDAY    DATE,
   USER_PASSWORD    VARCHAR2(255),
   USER_NUM_TEL     NUMBER,
   USER_PHOTO       BLOB,
   CONSTRAINT FK_CUSTOMER_USER
   FOREIGN KEY (USER_ID)
   REFERENCES USERS (USER_ID),
   CONSTRAINT FK_CUSTOMER_COMPANY
   FOREIGN KEY (COMPANY_ID)
   REFERENCES COMPANY (COMPANY_ID)
);
```

```
/*==========================*/
/* Create table FREELANCER   */
/*==========================*/
CREATE TABLE FREELANCER (
   USER_ID              NUMBER    PRIMARY KEY,
   NAME                 VARCHAR2(255),
   USER_LAST_NAME       VARCHAR2(255),
   USER_EMAIL           VARCHAR2(255),
   USER_LOCATION        VARCHAR2(255),
   USER_BIRTHDAY        DATE,
   USER_PASSWORD        VARCHAR2(255),
   USER_NUM_TEL         NUMBER,
   USER_PHOTO           BLOB,
   FREE_PORTFOLIO_LINK VARCHAR2(255),
   FREE_HOURLY_RATE     NUMBER(10,2),
   FREE_EXPERIENCE      NUMBER,
   FREE_STATUS          NUMBER(1),
   FREE_LEVEL           VARCHAR2(50),
   CONSTRAINT FK_FREELANCER_USER
   FOREIGN KEY (USER_ID) REFERENCES USERS (USER_ID)
);

/*==========================*/
/* Create table SKILL        */
/*==========================*/
CREATE TABLE SKILL (
   SKILL_ID    NUMBER    PRIMARY KEY,
   SKILL_NAME VARCHAR2(255)
);

/*==========================*/
/* Create table HAS_A        */
/*==========================*/
CREATE TABLE HAS_A (
   SKILL_ID NUMBER,
   USER_ID  NUMBER,
   PRIMARY KEY (SKILL_ID, USER_ID),
   CONSTRAINT FK_HAS_A_SKILL
   FOREIGN KEY (SKILL_ID)
   REFERENCES SKILL (SKILL_ID),
   CONSTRAINT FK_HAS_A_FREELANCER
   FOREIGN KEY (USER_ID)
   REFERENCES FREELANCER (USER_ID)
);

/*==========================*/
/* Create table PROJECT      */
/*==========================*/
CREATE TABLE PROJECT (
   PROJECT_ID           NUMBER    PRIMARY KEY,
   USER_ID              NUMBER,
   PROJECT_TITLE        VARCHAR2(255),
   PROJECT_DESCRIPTION CLOB,
   PROJECT_BUDGET       NUMBER(10,2),
   PROJECT_STATUS       VARCHAR2(50),
   PROJECT_DEADLINE     DATE,
   PROJECT_STARTDATE    DATE,
   CONSTRAINT FK_PROJECT_CUSTOMER
   FOREIGN KEY (USER_ID)
   REFERENCES CUSTOMER (USER_ID)
);

/*==========================*/
/* Create table NOTIFICATION */
/*==========================*/
CREATE TABLE NOTIFICATION (
   NOTIFICATION_ID       NUMBER    PRIMARY KEY,
   NOTIFICATION_TYPE     VARCHAR2(50),
   NOTIFICATION_CONTENT CLOB,
   NOTIFICATION_STATUS   NUMBER(1),
   NOTIFICATION_DATE     TIMESTAMP
);
```



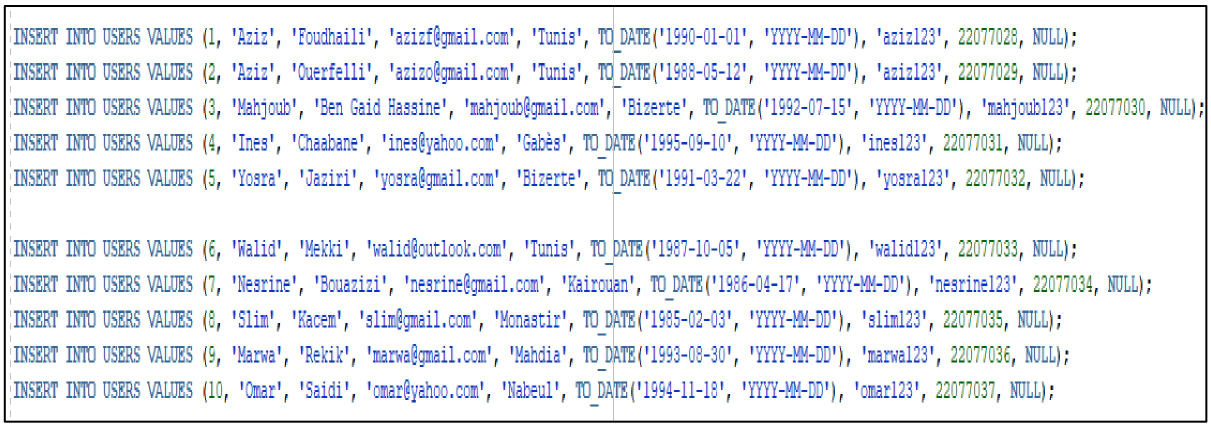
```
/*===========================*/
/* Create table MESSAGES     */
/*===========================*/
CREATE TABLE MESSAGES (
   MESSAGE_ID        NUMBER    PRIMARY KEY,
   USER_ID           NUMBER,
   RECEIVER_ID       NUMBER,
   NOTIFICATION_ID   NUMBER,
   MESSAGE_STATUS    NUMBER(1),
   MESSAGE_TEXT      CLOB      NOT NULL,
   SEND_DATE         TIMESTAMP,
   RECEIV_DATE       TIMESTAMP,
   CONSTRAINT FK_MESSAGE_SENDER
   FOREIGN KEY (USER_ID)
   REFERENCES USERS (USER_ID),
   CONSTRAINT FK_MESSAGE_RECEIVER
   FOREIGN KEY (RECEIVER_ID)
   REFERENCES USERS (USER_ID),
   CONSTRAINT FK_MESSAGE_NOTIFICATION
   FOREIGN KEY (NOTIFICATION_ID)
   REFERENCES NOTIFICATION (NOTIFICATION_ID)
);

/*===========================*/
/* Create table EVALUATION   */
/*===========================*/
CREATE TABLE EVALUATION (
   PROJECT_ID          NUMBER,
   USER_ID             NUMBER,
   EVALUATION_SCORE    NUMBER,
   EVALUATION_COMMENT  CLOB,
   EVALUATION_DATE     TIMESTAMP,
   PRIMARY KEY (PROJECT_ID, USER_ID),
   CONSTRAINT FK_EVAL_PROJECT
   FOREIGN KEY (PROJECT_ID)
   REFERENCES PROJECT (PROJECT_ID),
   CONSTRAINT FK_EVAL_FREELANCER
   FOREIGN KEY (USER_ID)
   REFERENCES FREELANCER (USER_ID)
);

/*===========================*/
/* Create table PAYMENT      */
/*===========================*/
CREATE TABLE PAYMENT (
   PAYMENT_ID        NUMBER    PRIMARY KEY,
   PROJECT_ID        NUMBER,
   NOTIFICATION_ID   NUMBER,
   PAYMENT_AMOUNT    NUMBER(10,2),
   PAYMENT_DATE      TIMESTAMP,
   PAYMENT_METHOD    VARCHAR2(50),
   PAYMENT_STATUS    VARCHAR2(50),
   CONSTRAINT FK_PAYMENT_PROJECT
   FOREIGN KEY (PROJECT_ID)
   REFERENCES PROJECT (PROJECT_ID),
   CONSTRAINT FK_PAYMENT_NOTIFICATION
   FOREIGN KEY (NOTIFICATION_ID)
   REFERENCES NOTIFICATION (NOTIFICATION_ID)
);
```



```
/*==========================*/
/* Create table PROPOSAL    */
/*==========================*/
CREATE TABLE PROPOSAL (
   PROPOSAL_ID       NUMBER        PRIMARY KEY,
   PROJECT_ID        NUMBER        NOT NULL,
   FREELANCER_ID     NUMBER        NOT NULL,
   NOTIFICATION_ID   NUMBER,
   PROPOSAL_AMOUNT   NUMBER(10,2),
   PROPOSAL_DEADLINE DATE,
   PROPOSAL_STATUS   VARCHAR2(50),
   PROPOSAL_DATE     DATE,
   PROPOSAL_TEXT     CLOB,
   CONSTRAINT FK_PROPOSAL_PROJECT
   FOREIGN KEY (PROJECT_ID)
   REFERENCES PROJECT (PROJECT_ID),
   CONSTRAINT FK_PROPOSAL_FREELANCER
   FOREIGN KEY (FREELANCER_ID)
   REFERENCES FREELANCER (USER_ID),
   CONSTRAINT FK_PROPOSAL_NOTIFICATION
   FOREIGN KEY (NOTIFICATION_ID)
   REFERENCES NOTIFICATION (NOTIFICATION_ID)
);

/*==========================*/
/* Create table WORK        */
/*==========================*/
CREATE TABLE WORK (
   PROJECT_ID NUMBER,
   USER_ID    NUMBER,
   PRIMARY KEY (PROJECT_ID, USER_ID),
   CONSTRAINT FK_WORK_PROJECT
   FOREIGN KEY (PROJECT_ID)
   REFERENCES PROJECT (PROJECT_ID),
   CONSTRAINT FK_WORK_FREELANCER
   FOREIGN KEY (USER_ID)
   REFERENCES FREELANCER (USER_ID)
);
```

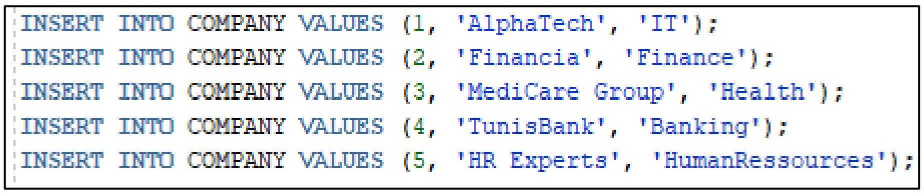
Appendix B : Populating the database

Populating the USERS table



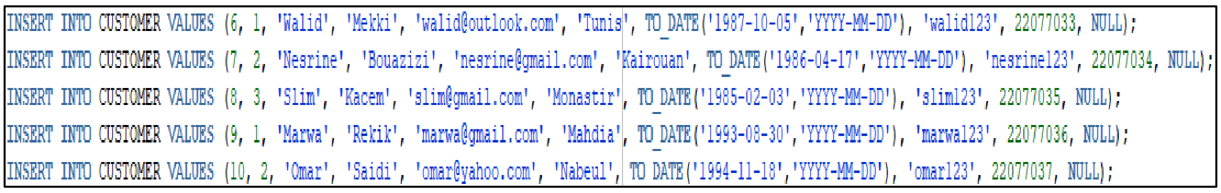
```sql
INSERT INTO USERS VALUES (1, 'Aziz', 'Foudhaili', 'azizf@gmail.com', 'Tunis', TO_DATE('1990-01-01', 'YYYY-MM-DD'), 'aziz123', 22077028, NULL);
INSERT INTO USERS VALUES (2, 'Aziz', 'Ouerfelli', 'azizo@gmail.com', 'Tunis', TO_DATE('1988-05-12', 'YYYY-MM-DD'), 'aziz123', 22077029, NULL);
INSERT INTO USERS VALUES (3, 'Mahjoub', 'Ben Gaid Hassine', 'mahjoub@gmail.com', 'Bizerte', TO_DATE('1992-07-15', 'YYYY-MM-DD'), 'mahjoub123', 22077030, NULL);
INSERT INTO USERS VALUES (4, 'Ines', 'Chaabane', 'ines@yahoo.com', 'Gabès', TO_DATE('1995-09-10', 'YYYY-MM-DD'), 'ines123', 22077031, NULL);
INSERT INTO USERS VALUES (5, 'Yosra', 'Jaziri', 'yosra@gmail.com', 'Bizerte', TO_DATE('1991-03-22', 'YYYY-MM-DD'), 'yosra123', 22077032, NULL);

INSERT INTO USERS VALUES (6, 'Walid', 'Mekki', 'walid@outlook.com', 'Tunis', TO_DATE('1987-10-05', 'YYYY-MM-DD'), 'walid123', 22077033, NULL);
INSERT INTO USERS VALUES (7, 'Nesrine', 'Bouazizi', 'nesrine@gmail.com', 'Kairouan', TO_DATE('1986-04-17', 'YYYY-MM-DD'), 'nesrine123', 22077034, NULL);
INSERT INTO USERS VALUES (8, 'Slim', 'Kacem', 'slim@gmail.com', 'Monastir', TO_DATE('1985-02-03', 'YYYY-MM-DD'), 'slim123', 22077035, NULL);
INSERT INTO USERS VALUES (9, 'Marwa', 'Rekik', 'marwa@gmail.com', 'Mahdia', TO_DATE('1993-08-30', 'YYYY-MM-DD'), 'marwa123', 22077036, NULL);
INSERT INTO USERS VALUES (10, 'Omar', 'Saidi', 'omar@yahoo.com', 'Nabeul', TO_DATE('1994-11-18', 'YYYY-MM-DD'), 'omar123', 22077037, NULL);
```

Populating the COMPANY table



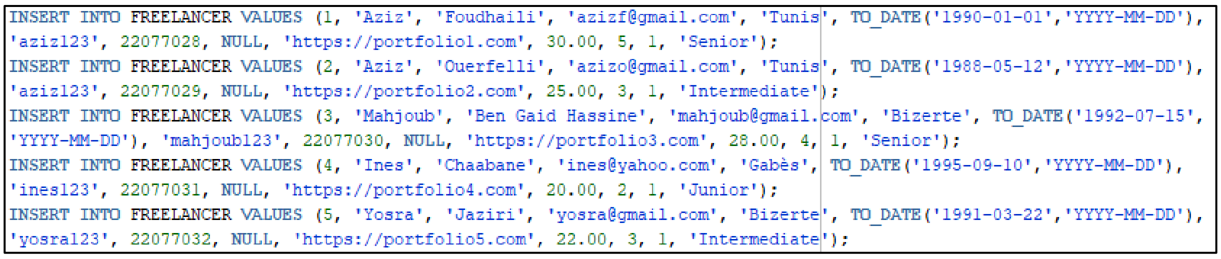
```sql
INSERT INTO COMPANY VALUES (1, 'AlphaTech', 'IT');
INSERT INTO COMPANY VALUES (2, 'Financia', 'Finance');
INSERT INTO COMPANY VALUES (3, 'MediCare Group', 'Health');
INSERT INTO COMPANY VALUES (4, 'TunisBank', 'Banking');
INSERT INTO COMPANY VALUES (5, 'HR Experts', 'HumanRessources');
```

Populating the CUSTOMER table



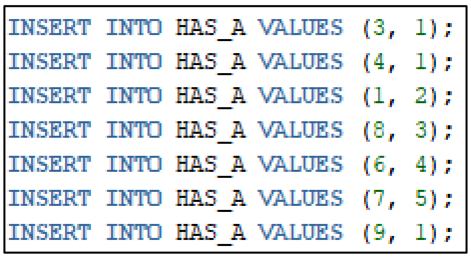
```sql
INSERT INTO CUSTOMER VALUES (6, 1, 'Walid', 'Mekki', 'walid@outlook.com', 'Tunis', TO_DATE('1987-10-05','YYYY-MM-DD'), 'walid123', 22077033, NULL);
INSERT INTO CUSTOMER VALUES (7, 2, 'Nesrine', 'Bouazizi', 'nesrine@gmail.com', 'Kairouan', TO_DATE('1986-04-17','YYYY-MM-DD'), 'nesrine123', 22077034, NULL);
INSERT INTO CUSTOMER VALUES (8, 3, 'Slim', 'Kacem', 'slim@gmail.com', 'Monastir', TO_DATE('1985-02-03','YYYY-MM-DD'), 'slim123', 22077035, NULL);
INSERT INTO CUSTOMER VALUES (9, 1, 'Marwa', 'Rekik', 'marwa@gmail.com', 'Mahdia', TO_DATE('1993-08-30','YYYY-MM-DD'), 'marwa123', 22077036, NULL);
INSERT INTO CUSTOMER VALUES (10, 2, 'Omar', 'Saidi', 'omar@yahoo.com', 'Nabeul', TO_DATE('1994-11-18','YYYY-MM-DD'), 'omar123', 22077037, NULL);
```

Populating the FREELANCER table



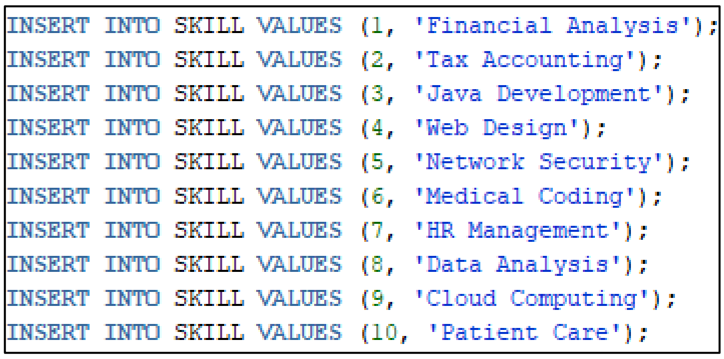
```sql
INSERT INTO FREELANCER VALUES (1, 'Aziz', 'Foudhaili', 'azizf@gmail.com', 'Tunis', TO_DATE('1990-01-01','YYYY-MM-DD'),
'aziz123', 22077028, NULL, 'https://portfolio1.com', 30.00, 5, 1, 'Senior');
INSERT INTO FREELANCER VALUES (2, 'Aziz', 'Ouerfelli', 'azizo@gmail.com', 'Tunis', TO_DATE('1988-05-12','YYYY-MM-DD'),
'aziz123', 22077029, NULL, 'https://portfolio2.com', 25.00, 3, 1, 'Intermediate');
INSERT INTO FREELANCER VALUES (3, 'Mahjoub', 'Ben Gaid Hassine', 'mahjoub@gmail.com', 'Bizerte', TO_DATE('1992-07-15',
'YYYY-MM-DD'), 'mahjoub123', 22077030, NULL, 'https://portfolio3.com', 28.00, 4, 1, 'Senior');
INSERT INTO FREELANCER VALUES (4, 'Ines', 'Chaabane', 'ines@yahoo.com', 'Gabès', TO_DATE('1995-09-10','YYYY-MM-DD'),
'ines123', 22077031, NULL, 'https://portfolio4.com', 20.00, 2, 1, 'Junior');
INSERT INTO FREELANCER VALUES (5, 'Yosra', 'Jaziri', 'yosra@gmail.com', 'Bizerte', TO_DATE('1991-03-22','YYYY-MM-DD'),
'yosra123', 22077032, NULL, 'https://portfolio5.com', 22.00, 3, 1, 'Intermediate');
```

Populating the HAS_A table



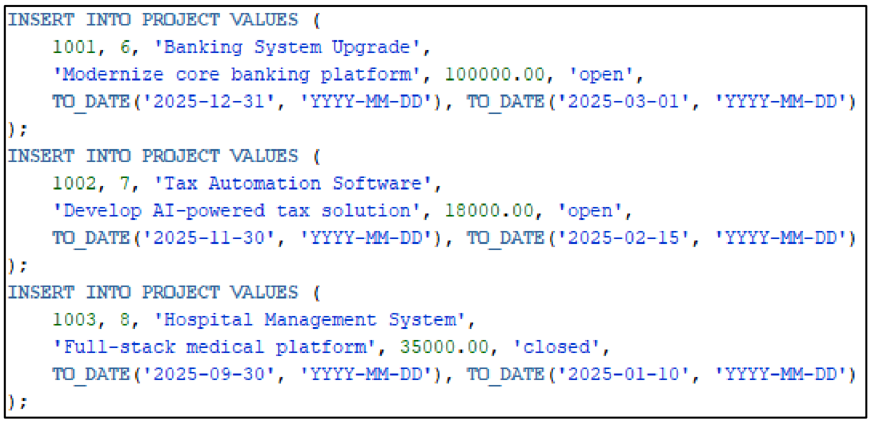
```sql
INSERT INTO HAS_A VALUES (3, 1);
INSERT INTO HAS_A VALUES (4, 1);
INSERT INTO HAS_A VALUES (1, 2);
INSERT INTO HAS_A VALUES (8, 3);
INSERT INTO HAS_A VALUES (6, 4);
INSERT INTO HAS_A VALUES (7, 5);
INSERT INTO HAS_A VALUES (9, 1);
```

Populating the SKILL table



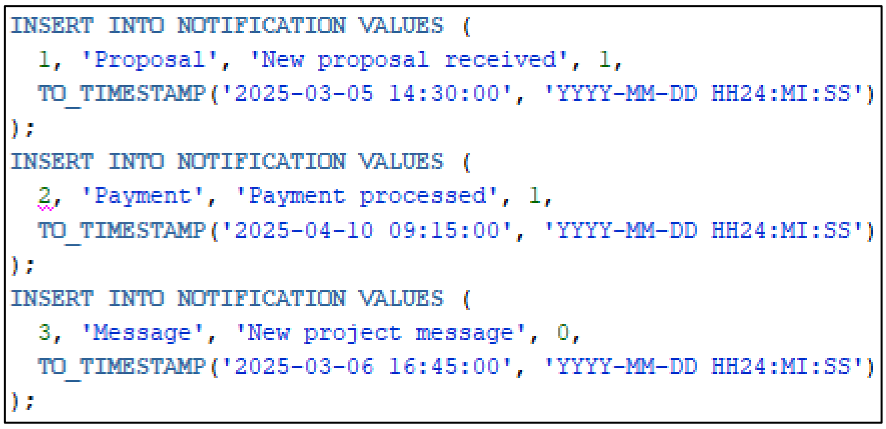
```sql
INSERT INTO SKILL VALUES (1, 'Financial Analysis');
INSERT INTO SKILL VALUES (2, 'Tax Accounting');
INSERT INTO SKILL VALUES (3, 'Java Development');
INSERT INTO SKILL VALUES (4, 'Web Design');
INSERT INTO SKILL VALUES (5, 'Network Security');
INSERT INTO SKILL VALUES (6, 'Medical Coding');
INSERT INTO SKILL VALUES (7, 'HR Management');
INSERT INTO SKILL VALUES (8, 'Data Analysis');
INSERT INTO SKILL VALUES (9, 'Cloud Computing');
INSERT INTO SKILL VALUES (10, 'Patient Care');
```

Populating the PROJECT table



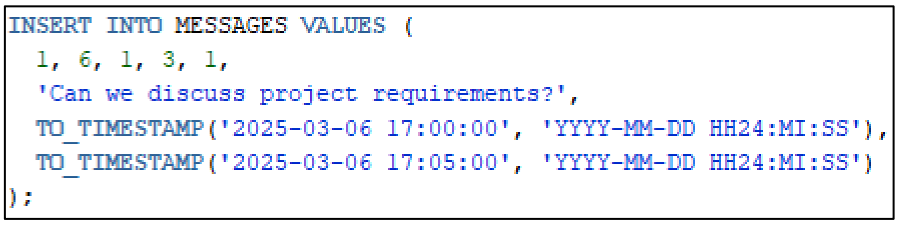
```sql
INSERT INTO PROJECT VALUES (
    1001, 6, 'Banking System Upgrade',
    'Modernize core banking platform', 100000.00, 'open',
    TO_DATE('2025-12-31', 'YYYY-MM-DD'), TO_DATE('2025-03-01', 'YYYY-MM-DD')
);
INSERT INTO PROJECT VALUES (
    1002, 7, 'Tax Automation Software',
    'Develop AI-powered tax solution', 18000.00, 'open',
    TO_DATE('2025-11-30', 'YYYY-MM-DD'), TO_DATE('2025-02-15', 'YYYY-MM-DD')
);
INSERT INTO PROJECT VALUES (
    1003, 8, 'Hospital Management System',
    'Full-stack medical platform', 35000.00, 'closed',
    TO_DATE('2025-09-30', 'YYYY-MM-DD'), TO_DATE('2025-01-10', 'YYYY-MM-DD')
);
```

Populating the NOTIFICATION table



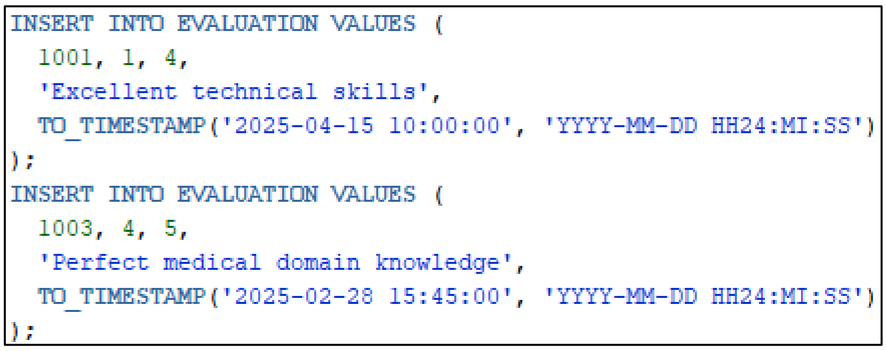
```sql
INSERT INTO NOTIFICATION VALUES (
  1, 'Proposal', 'New proposal received', 1,
  TO_TIMESTAMP('2025-03-05 14:30:00', 'YYYY-MM-DD HH24:MI:SS')
);
INSERT INTO NOTIFICATION VALUES (
  2, 'Payment', 'Payment processed', 1,
  TO_TIMESTAMP('2025-04-10 09:15:00', 'YYYY-MM-DD HH24:MI:SS')
);
INSERT INTO NOTIFICATION VALUES (
  3, 'Message', 'New project message', 0,
  TO_TIMESTAMP('2025-03-06 16:45:00', 'YYYY-MM-DD HH24:MI:SS')
);
```

Populating the MESSAGES table



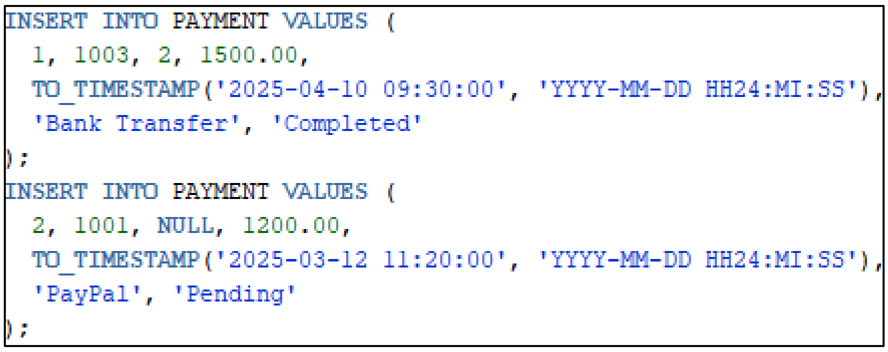
```sql
INSERT INTO MESSAGES VALUES (
  1, 6, 1, 3, 1,
  'Can we discuss project requirements?',
  TO_TIMESTAMP('2025-03-06 17:00:00', 'YYYY-MM-DD HH24:MI:SS'),
  TO_TIMESTAMP('2025-03-06 17:05:00', 'YYYY-MM-DD HH24:MI:SS')
);
```

Populating the EVALUATION table



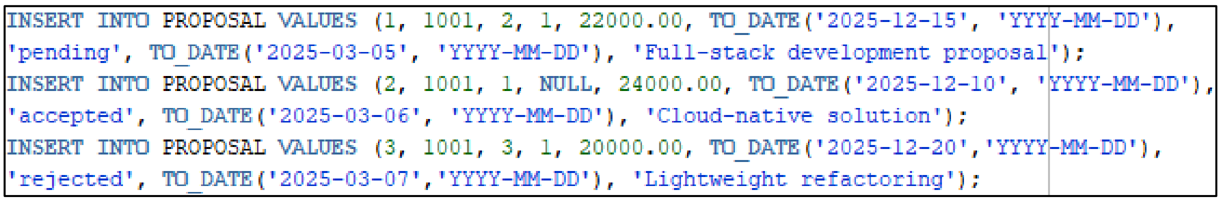
```sql
INSERT INTO EVALUATION VALUES (
  1001, 1, 4,
  'Excellent technical skills',
  TO_TIMESTAMP('2025-04-15 10:00:00', 'YYYY-MM-DD HH24:MI:SS')
);
INSERT INTO EVALUATION VALUES (
  1003, 4, 5,
  'Perfect medical domain knowledge',
  TO_TIMESTAMP('2025-02-28 15:45:00', 'YYYY-MM-DD HH24:MI:SS')
);
```

Populating the PAYMENT table



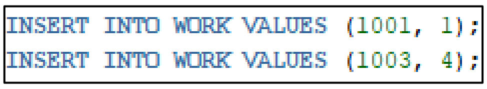
```sql
INSERT INTO PAYMENT VALUES (
  1, 1003, 2, 1500.00,
  TO_TIMESTAMP('2025-04-10 09:30:00', 'YYYY-MM-DD HH24:MI:SS'),
  'Bank Transfer', 'Completed'
);
INSERT INTO PAYMENT VALUES (
  2, 1001, NULL, 1200.00,
  TO_TIMESTAMP('2025-03-12 11:20:00', 'YYYY-MM-DD HH24:MI:SS'),
  'PayPal', 'Pending'
);
```

Populating the PROPOSAL table



```sql
INSERT INTO PROPOSAL VALUES (1, 1001, 2, 1, 22000.00, TO_DATE('2025-12-15', 'YYYY-MM-DD'),
'pending', TO_DATE('2025-03-05', 'YYYY-MM-DD'), 'Full-stack development proposal');
INSERT INTO PROPOSAL VALUES (2, 1001, 1, NULL, 24000.00, TO_DATE('2025-12-10', 'YYYY-MM-DD'),
'accepted', TO_DATE('2025-03-06', 'YYYY-MM-DD'), 'Cloud-native solution');
INSERT INTO PROPOSAL VALUES (3, 1001, 3, 1, 20000.00, TO_DATE('2025-12-20','YYYY-MM-DD'),
'rejected', TO_DATE('2025-03-07','YYYY-MM-DD'), 'Lightweight refactoring');
```

Populating the WORK table



```sql
INSERT INTO WORK VALUES (1001, 1);
INSERT INTO WORK VALUES (1003, 4);
```

Abstract

This project aims to create a database for the FreeConnect freelance platform, with a dynamic web interface. For the database design, we used PowerAMC, while for database management, we opted for MongoDB. Regarding the development of the web application, we utilized HTML, CSS, JS and Angular for the frontend, and Node.js for the backend.

Keywords: FreeConnect, Freelancer, PowerAMC, database, dynamic website.

Résumé

Ce projet vise à créer une base de données pour la plateforme freelance FreeConnect, avec une interface web dynamique. Pour la conception de la base de données, nous avons utilisé PowerAMC, tandis que pour la gestion de la base, nous avons opté pour MongoDB. En ce qui concerne le développement de l'application web, nous avons utilisé HTML, CSS, JS et Angular pour le frontend, et Node.js pour le backend.

Mots clés: FreeConnect, Freelancer, PowerAMC, base de données, site web dynamique.

الملخص

يهدف هذا المشروع إلى إنشاء قاعدة بيانات لمنصة العمل الحر FreeConnect، مع واجهة ويب ديناميكية. لتصميم قاعدة البيانات، استخدمنا PowerAMC، وبالنسبة لإدارة قواعد البيانات، اخترنا MongoDB. أما بالنسبة لتطوير تطبيق الويب، فقد استخدمنا HTML و CSS و JS و Angular للواجهة الأمامية و Node.js للواجهة الخلفية.

الكلمات المفاتيح: FreeConnect ، عامل حر، PowerAMC، قاعدة بيانات، موقع ديناميكي.