



# B2B MATCHMAKING SYSTEM

## Assignment 1

### Members:

1. Askari, Majid
2. Ghorbanian, Mahta
3. Jaberzadeh Ansari, Mahdi
4. Mirshafiee Khoozani, Mitra Sadat

# I.1. Project Definition

---

## **B2B Matchmaking System**

Applications on our smart phones, smart watches, tablets, and computers will become an indispensable aspect of our lives in the modern world. People may access data practically anywhere, perform any tasks on their mobile devices, and other things by expanding the internet throughout the world (e.g., by establishing 5G networks, satellite internet, or free WiFi spots) and having smart gadgets with powerful processors and vast memories.

Governments and societies will become more and more electronic as a result of the rapid change in the techniques and tools we use to solve problems on a daily basis. This means that either we need more applications or more functional and flexible portable devices for managing our job remotely (i.e. smart phones).

Although, the need for new applications on our computing devices (i.e., from a small smartwatch till a powerful professional laptop) is increasing constantly, however, the number of IT companies which provide services for application development requests hasn't increased with the same pace or distributed even in all parts of the world. Therefore, it is necessary to have some kind of matchmaking systems to connect **clients** to **providers** in this little global communication village.

Our **B2B Matchmaking System** is a bridge between **clients** and **providers**. **Providers** are those IT companies that can consult and develop different varieties of software systems, and **clients** are any type of customers that can order and pay for a development process. Our system tries to list best matches for our clients based on their interesting keywords.

This problem, is the main motivation beyond our project which we aim to build a system that answers this need using an agent-based methodology.

Our project has three main steps. Each Step will be delivered as a separate assignment:

1. **Step 1:** Project definition and High-level analysis
2. **Step 2:** Data interaction modeling and low-level design
3. **Step 3:** Implementation using JADE framework

For implementation we selected Java language, thus, we are going to use the JADE framework. Regarding the methodology we are going to use GAIA methodology for analysis and design our agent-based system.

We chose the GAIA methodology since it can be used with a variety of multi-agent systems and is broad in nature. It deals with both the macro-level (societal) and the micro-level (agent) aspects of systems.

GAIA is interested in how a community of agents works together to accomplish system-level objectives. What must be done by each individual agent in order to do this is another area of concern.

# I.2. System Specifications

---

## Table Of Contents

### 2. System Specifications

- [2-1. Business Case](#)
  - [2-2. System Description](#)
  - [2-3. Assumption](#)
  - [2-4. Requirements](#)
  - [2-5. Wish List \(Not implemented\)](#)
- 

## 2. System Specifications

In the **system specifications document**, we are going to speak about the five pieces of information. First we discuss the reasons why such a system is in demand from a business perspective in the [Business Case](#) section. Then, in the [System Description](#) section, we describe the main features of the application. Later, in the [Assumption](#) section, we discuss the pre-existing conditions that we assumed are there in place before starting the development. In the fourth part, actually [Requirements](#) we reflect the whole requirements that we have received from the customer. Finally, in the [Wish List](#) section, we determine which part of requirements are going to be planned for the future revisions and are not going to be delivered in our implementation.

### 2-1. Business Case

Due to the widespread use of the internet and smartphones, new applications are constantly emerging that are changing how we approach problems on a daily basis. The demand for new apps is increasing steadily all around the world, yet there aren't even that many IT companies in different regions. As a result, some technologies are required to link clients to suppliers around the globe. Our **B2B Research Matchmaking** serves as a link between customers and suppliers.

**Providers** can reveal their abilities by providing their resume and some topics that they are interested in and have enough power for implementing projects in that area.

In the other side our **Clients** can search in our database, review the resume of different providers, select them, and bid a price for a specific project.

If any deal happen during this process, then the selected **Provider** can enjoy 70% of the deal, the **Client** can have a nice functional application on time, and **the System** can continue its operation using the rest of the 30% of the deal.

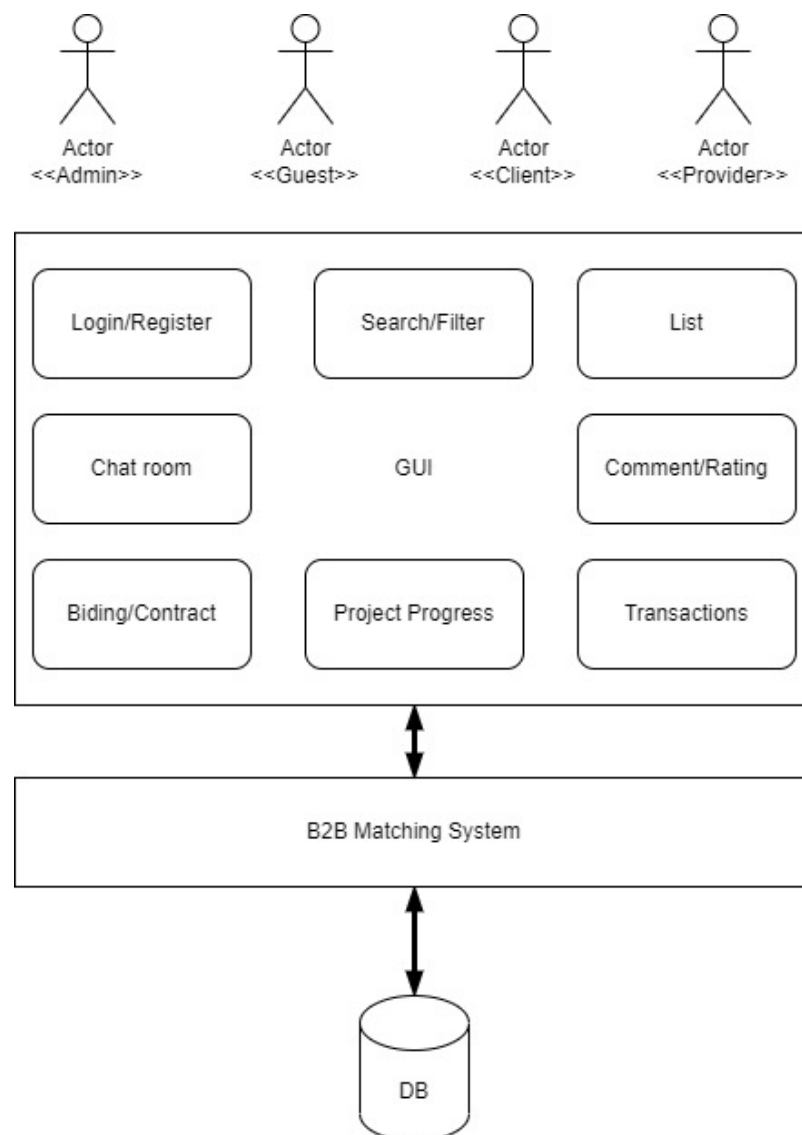
### 2-2. System Description

**The System** contains four agents. First, **the Host** or as it has been called in the requirements **the System**. This agent is responsible to *store data*, *moderate messages* and *connect* the other agents. Then **Providers**, which *provide services* by *introducing* themselves via *yellow pages*. In contrast, we have **Clients** which *search* and *consume* services. Finally, we have **Guests** that are a limited version of **Clients** which are more for *discovering* and *evaluating* the main functionality and examining the quality of data in our *database*. If a **Guest** find the information and services interesting, they *can register* and *become* a **Client**. Being a **Client** means they can start *biding Providers*.

As mentioned before, **the System** *charges* providers with *30% commission*. Therefore, besides of *registration* and *searching* modules, there must be an *accounting* module which *keeps track of financial transactions*.

All actors, can access to their *dashboard* via a *login* mechanism which is not delivered in the first phase. In fact, we mock the *login* by simulating the *sign in* mechanism.

Last but not least, we are not going to deliver the *chat*, *commenting* and *rating* features. We will discuss them in our design phase, however, in the implementation phase we exclude them for the first phase.



System High-Level Design

## 2-3. Assumption

In this section, we discuss the pre-existing conditions that we assumed are there in place before starting the development and we build our design by assuming them.

1. In our GUI, we won't provide any login mechanism for our actors. We would have a database which keeps records of our **Providers** and **Clients**. When already registered users want to login, they can just select their name from a list and press on the login key and then sign in. We assumed they don't abuse this situation and each use only login to his or her account.
2. For persisting the data, we will use one of the embedded databases in Java. A few of the dominant providers are available for choosing (i.e., [H2](#), [HyperSQL](#), [Apache Derby](#), [Berkley DB](#), [Java DB](#), [ObjectDB](#), and so forth. However, in this stage we don't know which database will be selected. We will select one due to the performance, price, and the license.
3. We have a specific policy and guideline for using any OSS component during our implementation. If we need to leverage any OSS component, we must make sure that we fulfill these steps:
  1. First of all we only accept libraries and components that have MIT license.
  2. If any OSS component does not have MIT license we must not use them.
  3. We must provide an static copy of the jar file in a separate folder in the lib directory. For example, for using a jar file as `xyz.jar`, we must make a sub folder `xyz` inside the `\lib` directory and put the jar file in it.
  4. We must provide the license file beside of the jar file.
  5. We must avoid copy/paste any snippet of code from any website or mailing list, specially [Stackoverflow.com](#).

## 2-4. Requirements

In this section, you can find the requirements. Each actor (i.e. agent) has been coded by **bold** style. Each action/verb has been declared by *italic* style. Finally, each attribute has been identified by underline style.

1. Ability to *sign up* as **Provider** and **Client**.
2. Ability to be a **Guest** and *visit* the app.
3. For **Providers**: ability to *submit* name, website, logo, resume, special keywords, hourly compensation.
4. **Providers** *can get* a verified icon if they *send* their proof of business to **the System**.  
**The System** should *make sure* that every piece of information *is correct* and then *accept* the request.
5. For **Guests**: ability to *search* keywords and *get* a list of available Providers.
6. A contract should be *sent* to a **Provider** the moment they *sign up*.
7. **Provider** should be able to *accept* or *reject* the contract.
8. Upon rejection, the **Provider** will be automatically *converted* to a **Client**, *losing* their resume, website, special keywords and hourly compensation information.
9. When a **Guest** *visits* the **App**, they can only *see* the name, website, logo, resume, and special keywords of **Providers**. They *cannot see* their hourly compensation. Also, they *cannot place* a bid for the **Provider**.
10. Signed-up Agents (**Providers** and **Clients**) *can see* every piece of information available on **the system**.

11. **Providers** can choose between Basic and Premium plans.  
Premium subscribers will appear first on the search list, regardless of their approval ratings or hourly compensation.
12. The sorting algorithm always puts Premium Providers on top, then verified Providers, and then the rest. Between each group, **Providers** should be sorted based on their approval ratings by default (can be changed).
13. A **Client** is able to change the sorting of results upon searching a keyword (e.g. **Clients'** approvals, number of projects done, the amount of hourly compensation).
14. A **Client** can request a **Provider** and place a bid. The bid can be a different value than the hourly compensation of the **Provider**.
15. **Provider** can accept or reject a bid.
16. A rejection from the **Provider** will be directly sent to the **Client**.
17. Accepting a request from the **Provider** will go through **the System** first, and not directly to the **Client**.
18. **The System**, upon receiving an accept confirmation, will pull up a contract and send it to both **Provider** and **Client**.
19. **Provider** and **Client** can accept or reject the contract.
20. Any money transfer will be handled by **the System**.  
**The System** will receive 30% of any transaction. This info should be in the contract.
21. Ability to watch the progress of the project for the both sides (**Provider** and **Client**).
22. The tracking page will show the tentative deadline, progress so far, and estimated time of completion based on the current pace.
23. A chat room page will be created for **Client** and **Provider** once a project gets accepted.
24. Any change request from the **Client** must first get accepted by the **Provider** after a project begins.  
Deadlines could change based on **Provider's** request
25. When a project is done, the **Client** can leave a comment and rating for the **Provider**,
26. A **Provider** can also leave a comment and rating for the **Client**. **Providers** can see the past ratings of a **Client** when there is a new bid.
27. The app **must** have a GUI.

## 2-5. Wish List (Not implemented)

It seems we can implement all requirements, except the number 21 till 26. Basically, it means we assume all parties are satisfied from each other and there is no need for monitoring or rating. Therefore, we hope to be able to implement items 1 to 20 and also number 27 that requests a mandatory GUI. And we put items 21 to 26 in our wish list.

## I.3. System Design Document

---

### Table Of Contents 📌

#### 3. System Design Document

- [3-1. Goal Hierarchy](#)
- [3-2. System Architecture](#)
- [3-3. Roles Identification](#)
- [3-4. Agents Description](#)
- [3-5. Agents Internal Architecture](#)
- [3-6. Technology Overview](#)

---

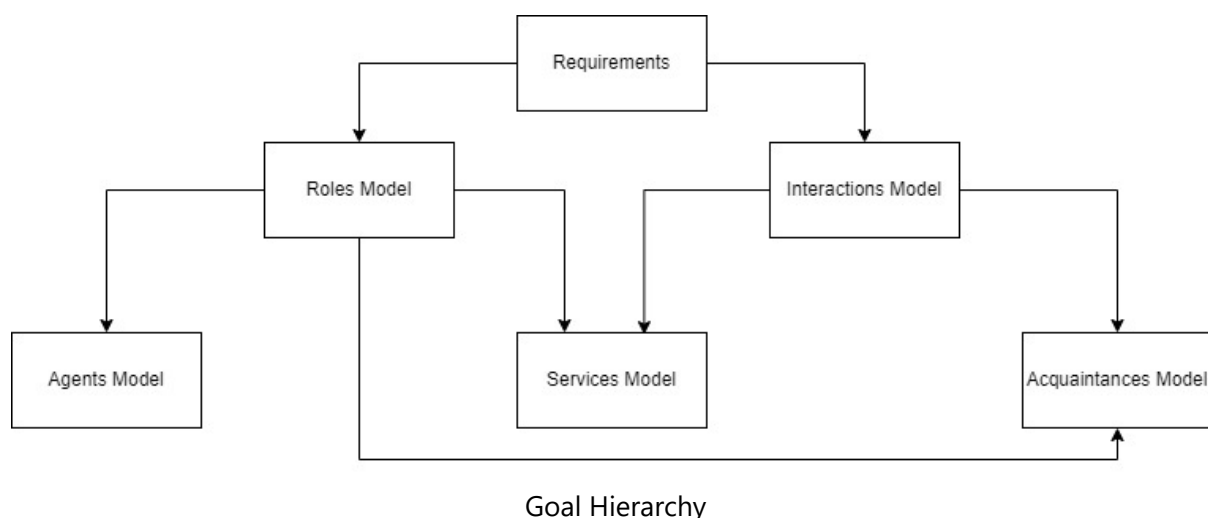
### 3. System Design Document

In the **system design document**, we are going to speak about the six pieces of information following an agent-based development methodology (i.e., GAIA), and create the analysis and design documents specified by the GAIA methodology. This section includes: [Goal Hierarchy](#), [Agent System Architecture](#), [Role Identification](#), [Agent Description](#), [Agent Internal Architecture](#), and finally [Technology Overview](#).

#### 3-1. Goal Hierarchy

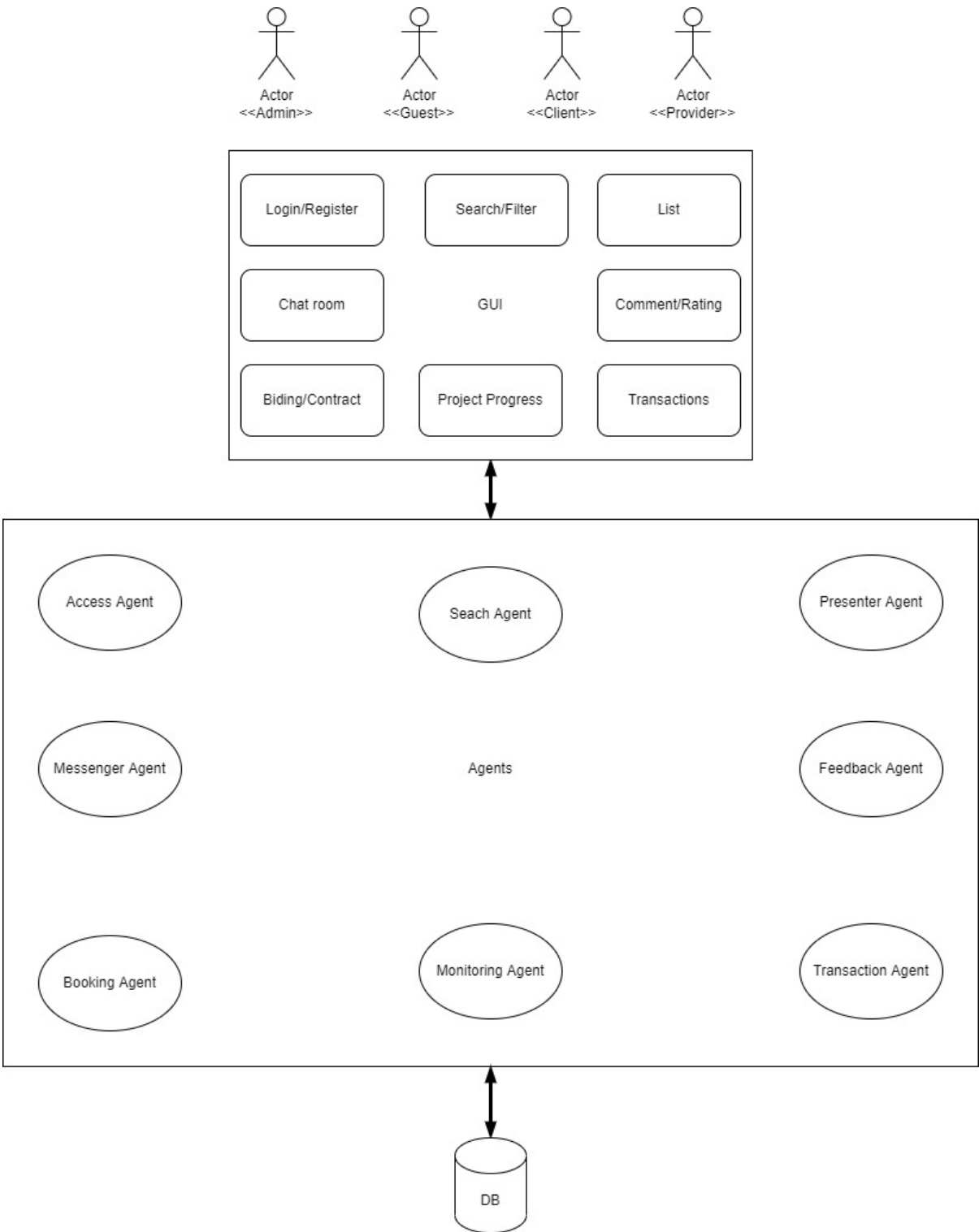
Using GAIA, we think of each agent as an entity with the resources of a computational processor. It is presumed that the objective is to create a system that maximizes a particular global quality metric. From the perspective of the system's constituent parts, nevertheless, this structure might not be ideal.

The GAIA approach encourages developers to see creating software systems as an organizational design process with software agents serving as its building blocks. Therefore, in our analysis phase, we planned to extract 5 models from the requirements.



3-2. System Architecture

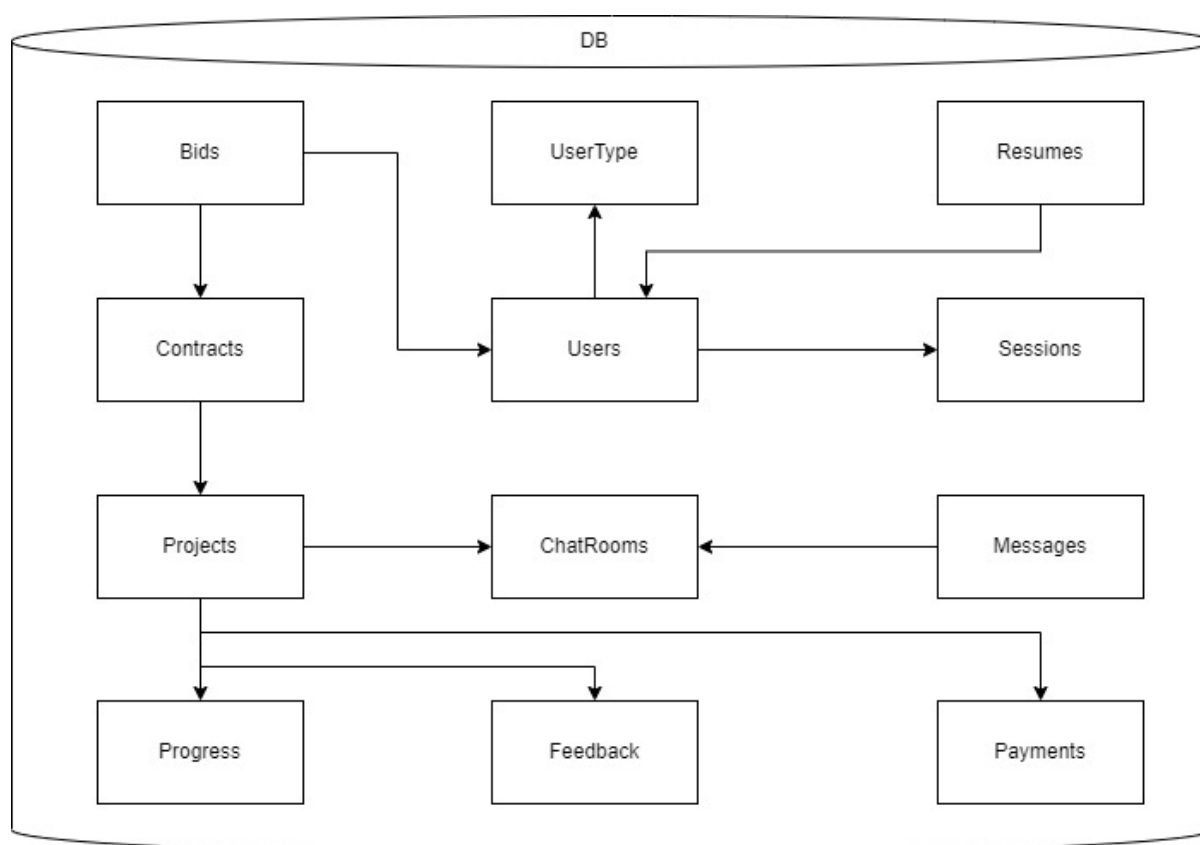
Here is a high-level design of our system.



System Architecture

If we want to go in more details, we should demonstrate our database structure first:





Database Architecture

The user starts the journey with the login page. If he or she wants, they can just login as a guest and access the system with limited search functionality.

Otherwise, if they don't have an account, they can move to register page and register as a **Provider** or **Client** by providing the required information.

After a successful login, user will be redirected to his or her dashboard.

In the dashboard

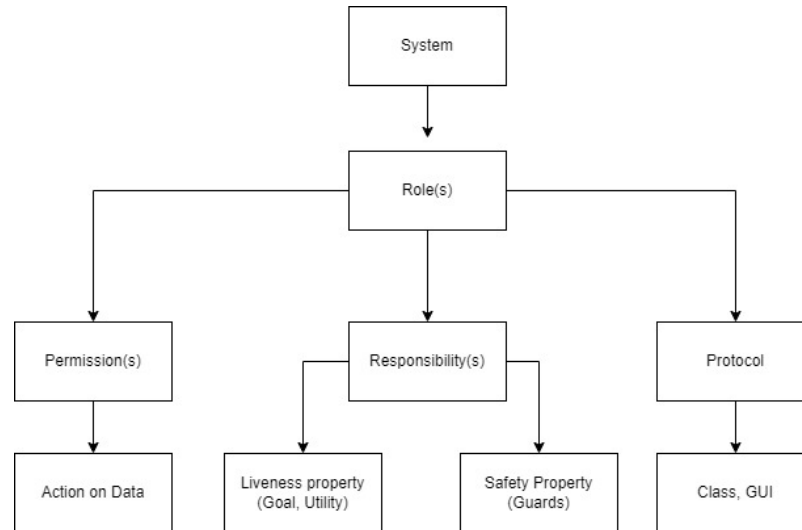
1. There is a list of current projects for monitoring, reporting, payment, and or requesting/confirming a change.
2. There is a list of past projects for commenting or rating the other party.
3. There is a pending list that shows list of projects that needs some sort of actions. For example, accepting or rejecting a bid, or confirming a contract. After any action, the list of projects will be updated.

Also, there is an option for **Clients** only to bid after selecting a **Provider** from the search list.

Finally, user can logout of the system for safety reasons.


### 3-3. Roles Identification


One of the main steps in GAIA methodology is to identifying roles based on the following chart.




Role detection process

Therefore, here is a table that demonstrates the detected roles.

	Role	By means of?	What?	What?	How?
Row#	Role Name	Permissions	Liveness Property	Safety Property	Protocols
1	Sign Up	Read and Write users data	Handles the process of sign up for Providers and Clients	Checks validity of user data.	Registration
2	Sign In	Read users data, Authenticate user, Create Session	Handles the process of authentication. If user exists then creates a session. Also, create guest session for Guests.	Checks for active users, and apply SQL injection guards	Authenticator
3	Search Engine	Read providers data	Apply a query on Keywords column of providers table	Deliver a list of providers based on the data that user allowed to access.	SearchEngine
4	Bid Handler	Read and write on bids data	Handles the process of creating, accepting, or rejecting a bid	Checks if Clients have any waiting bid or not. Only one bid per Provider is allowed.	Bid

	Role	By means of?	What?	What?	How?
5	Contract Creation	Read and Write contracts data	Handles the process of creating contracts and sends the contract to both sides after Provider accepts the bid.	Checks if there is no contracts waiting for acceptance for these 2 parties.	CreateContract
6	Project Creation	Read and Write projects data, Read contracts data	Handles the process of creating the project based on the Client request after accepting contracts by both side.	Checks both Provider and Client have been accepted the contract and there is no project in database.	CreateProject
7	Payment Handler	Reads and write payment data	Handles the process of Payments	Checks if payments has not yet been done, Checks if payment is equal to what we have in Contract.	TransferMoney
8	Project Tracker	Reads and write projects progress data	Handles the process of tracking project progress, deadline and estimations	Checks if project is still active.	TrackProject
9	Project Change Handler	Read and write projects data	Handles the process of changing a project, upon the Client request. Delivers the changed requirement/contract to Provider.	Checks if there is no change request in database.	ChangeProject
10	Message Handler	Read and write messages data	Handles the process of sending messages between Provider and Client in a specific chatroom	Checks if user belongs to a chatroom	Message

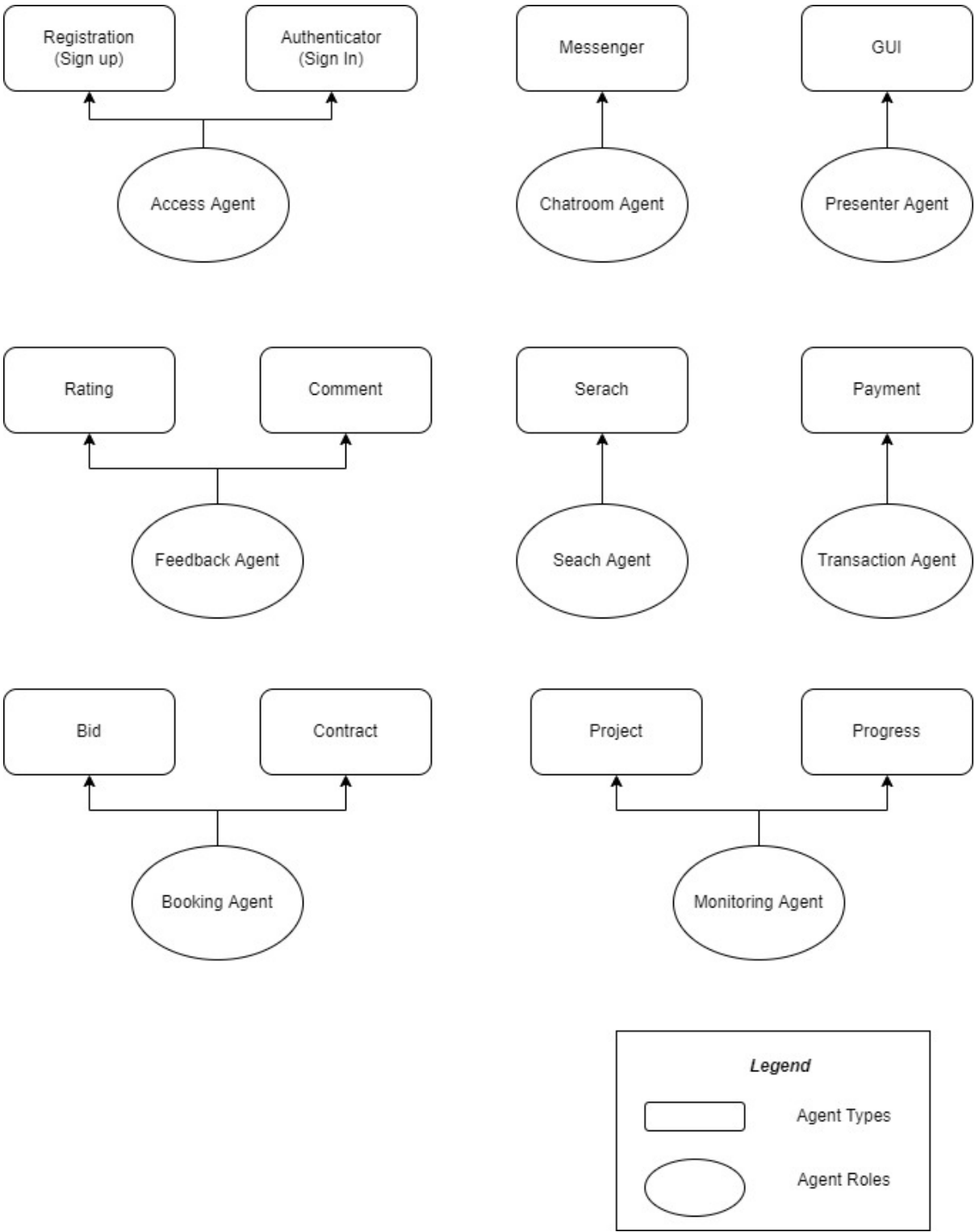
	Role	By means of?	What?	What?	How?
11	Feedback Handler	Read and write feedback data	Handles the comments and ratings of projects	Checks if user has worked with feedback receiver via a contract in the past. Checks if user has not yet deliver a feedback related to an experience.	Feedback

### 3-4. Agents Description

We detected and designed 8 agents for this system as following:

1. **Access Agent:** This agent is responsible for registration, login and logout of users. If user provides correct credentials it generates a session in the database.
2. **Search Agent:** This agents is responsible for applying suitable query on the database, list providers based on the ordering rules which mentioned in the requirements.
3. **Presenter Agent:** This agent is responsible to show search results and other data related to each user in the GUI. We can assume this agent as the UI manager.
4. **Booking Agent:** This agent manage bids and contracts. It means this agent handles transactions tills forming a project.
5. **Transaction Agent:** This agent is responsible to convert a project state from *pending* to *active* upon receiving money from the **Client**. Also, as soon as a change to the project confirmed, it changes the state to *pending* again. It also calculates and conveys the portion of the **Provider** from the earnings. A project's state cannot change after it is been flagged as *completed*.
6. **Chatroom Agent:** This agent conveys messages between two parties of a project and shows a history of old messages.
7. **Monitoring Agent:** This agent is responsible to records the amount of progress, and estimate the delivery time based on the current pace.
8. **Feedback Agent:** Finally, this agent is responsible to moderates rating and comments after a project is in *completed* state.

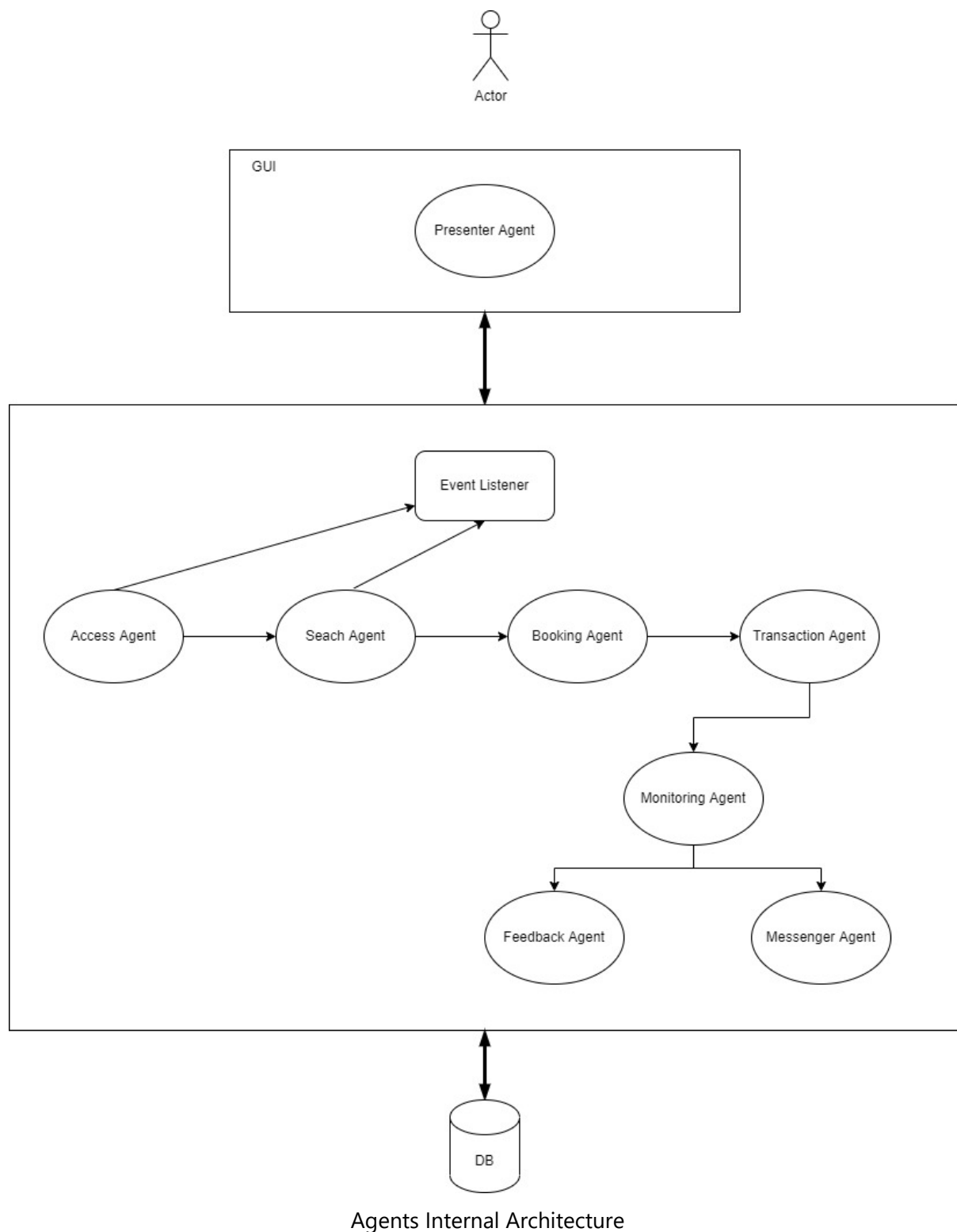
Here is a figure that illustrates our agent model.



Agent Model

3-5. Agents Internal Architecture

Here is a figure that illustrates the internal communication between the agents:



### 3-6. Technology Overview

1. Regarding the technology; we are going to use JAVA and JADE framework for implementation.
2. For persisting the information, we will use one of the JAVA internal databases, however, it is not yet clear which one at the moment. We need to clarify it later during the implementation with respect to

compatibility with JADE. What we can clearly stated now is, we are going to use one of the following databases:

- [H2](#)
- [HyperSQL](#)
- [Apache Derby](#)
- [Berkley DB](#)
- [Java DB](#)
- [ObjectDB](#)

3. Regarding IDE we are going to use Eclipse version 2022-06 and VScode version 2022-06.

4. For archiving the code, we use [this GitHub](#) repository.