National Technical University of Athens
School of Electrical and Computer Engineering
Relational Databases
Spring Semester 2021-2022

Project Team 109
Ioannis Dressos - 03119608
Ioannis Boskovic - 03119640
Georgios Kitsios - 03119801

Semester Project Report

Introduction

The project concerns the development of a graphical user interface application that connects to a relational database. The user must be able to view and edit the database entries without any technical knowledge being required on their part.

We (the team) chose to develop a single Java application instead of a web application stack in order for the installation and configuration processes to be simple and easy even for someone who has little to no experience with such proceedings.

Technical Specifications

GitHub Repository: https://github.com/idressos/hfri-erp

We chose to develop the GUI application in the Java programming language specifically due to the following advantages it has to offer to this project:

- Simple, fast compilation from source code - accelerates testing phase significantly.
- The single binary produced by the building process is compatible with every operating system that supports the JVM, as long as it has a desktop environment installed.
- The source code is organized and easy to read due to the object-oriented nature of Java.
- The Java syntax is easy to understand and get a hold of, especially for someone who already has experience in other programming languages.
- Java provides an exceptional built-in API for communication with relational databases (java.sql package).

The application depends on libraries provided by the implementation manufacturer or by third parties under open-source licenses which this academic project complies with:

- https://github.com/mariadb-corporation/mariadb-connector-j
  - Used for connecting to and communicating with the database server.
- https://github.com/stleary/JSON-java
  - Used for reading and storing the configuration file of the application.
- https://github.com/mwiede/jsch
  - Used (optionally) for tunneling the connection between the application and the database server through an SSH server.

These libraries are automatically included in the application binary during compilation and the user does not need to intervene with the process in any way to obtain them.

For the development of this application we used Java version 17. Specifically we used the Java SE Development Kit 17 which is the official implementation of the Java standard by the manufacturer Oracle at the time of this writing.

Therefore, in order for the user to compile and run the application they must use an implementation of the JVM version 17 and up. Newer versions are supported according to the manufacturer specifications, while older versions are in no way guaranteed to work.

Installation Instructions

Prerequisites:

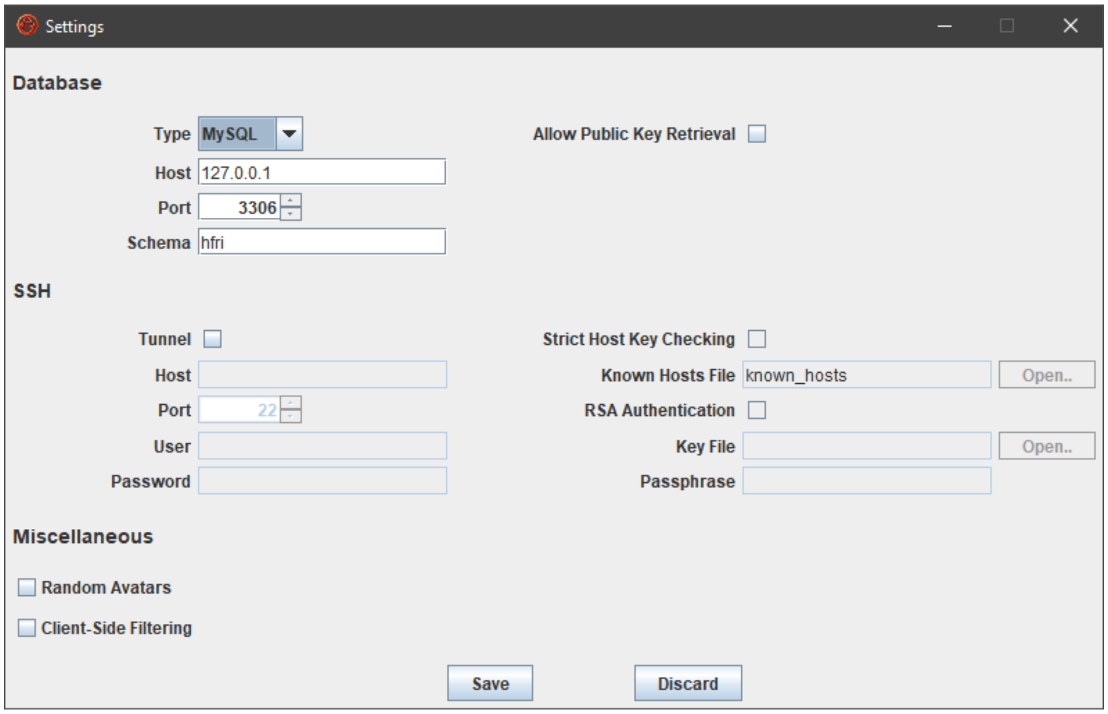
- An SQL database server.
  - **Only MySQL 8.0+ and MariaDB 10.0+ are supported**
  - You must initialize the project database using the DDL/DML scripts beforehand

1. Install any JVM implementation version 17 and up.
   1.1. Oracle Java SE Development Kit is recommended: https://www.oracle.com/java/technologies/downloads/
2. Install the Apache Maven project management tool: https://maven.apache.org/download.cgi
   2.1. Installation instructions: https://maven.apache.org/install.html
3. Install Git: https://git-scm.com/book/en/v2/Getting-Started-Installing-Git
4. Clone GitHub repository and build:

```
$ git clone https://github.com/idressos/hfri-erp.git
$ cd hfri-erp
$ mvn clean install
```

If the building process succeeds (as it should), the application binary will be exported to the folder `target` and will be named according to the format `hfri-erp-*.jar`. This binary file is ready for use and the user can delete all folders (and their contents) that were created during the cloning and compilation processes if they choose to do so.

Configuration



Settings

Database

Type	MySQL	Allow Public Key Retrieval	☐
Host	127.0.0.1		
Port	3306		
Schema	hfri		

SSH

Tunnel	☐	Strict Host Key Checking	☐
Host		Known Hosts File	known_hosts
Port	22	RSA Authentication	☐
User		Key File	
Password		Passphrase	

Miscellaneous

☐ Random Avatars

☐ Client-Side Filtering

Save Discard

The application settings are separated into the following categories:

<u>Database Connection Settings</u>

Type: The database server type (MySQL or MariaDB)
Host: The database server host (IP address or domain)
Port: The port the database server is bound to (listening) on the host
Schema: The database name to use (hfri by default)
Allow Public Key Retrieval: Read below

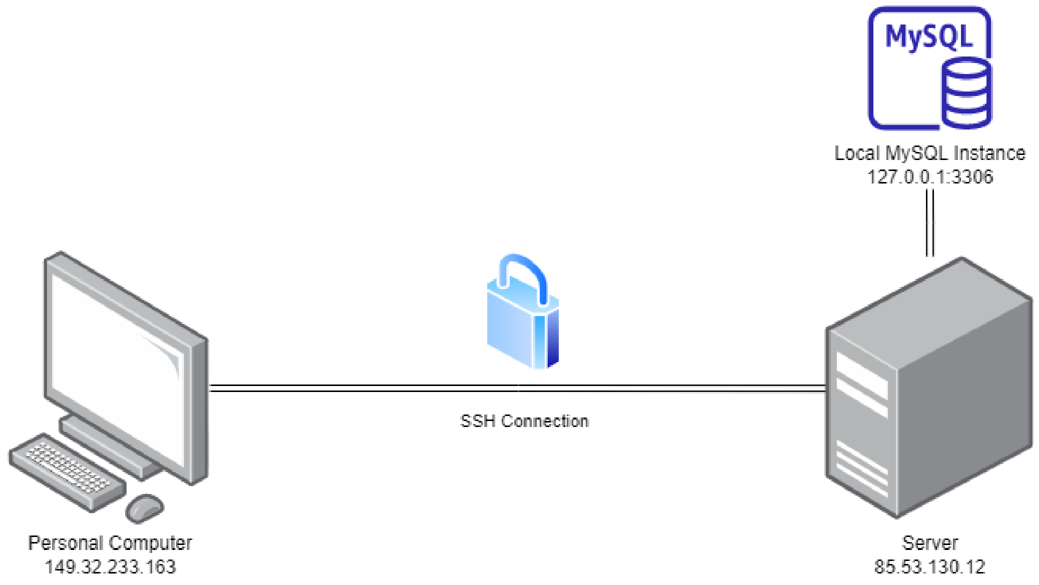
SSH Settings

As mentioned in the technical specifications section, the user has the option to tunnel the connection between the application and the database server through SSH. This can be useful for the following scenarios:

- Encrypting the connection between the application and a database server running on a remote computer.
- Connecting to a database server which is bound locally on a remote computer and is not exposed to the internet.
- Connecting to a database server running behind a firewall on a remote computer.

In case the user chooses to use SSH tunneling, the fields `Database/Host` and `Database/Port` we mentioned above have to be adjusted to the new network configuration of the application.

→ For example, if the value of the field `Database/Host` is 127.0.0.1 (or localhost), it doesn't point to the local computer running the application anymore, but to the SSH server. This is because the database host resolution takes place after connecting to the SSH tunnel.



SSH tunnel between 149.32.233.163 and 85.53.130.12 allows for communication between the personal computer and the MySQL instance running locally on the server

In order for the connection portrayed in the diagram above to succeed, the following values must be set:

```
Database/Host: 127.0.0.1
Database/Port: 3306
SSH/SSH Tunnel: Enabled
SSH/Host: 85.53.130.12
SSH/Port: (Depends on the SSH server configuration - 22 by default)
```

Advanced Settings

Database/Allow Public Key Retrieval
Allows the application to directly ask the database server for the RSA public key.

Warning! This is not recommended because it leaves the connection vulnerable to man-in-the-middle attacks.
This is meant for use in case a respective error occurs while connecting to the database server through the application, but only in local installations for testing purposes.

SSH/Strict Host Key Checking
Every time we connect to an SSH server, we are sent a key that acts as an identifier for that specific server so that we know we are connecting to the computer we meant to and not to a malicious server that is trying to steal our data.

Thus, it is considered good practice to save that key the first time we connect to the SSH server through a trusted network (for example, our home network) so we can perform the security check described above every time we connect to the same server in the future.

This way, if our authentication with the SSH server gets intercepted by a malicious third party, we will be notified because the attacker will most likely send the wrong host key to the SSH client.

If we disable SSH strict host key checking, this procedure is not executed.
This is meant for use the first time we connect to the database server through an SSH tunnel, so the authentication with the SSH server does not fail and we get a chance to save the host key to the file set by the value of `SSH/Known Hosts File`.

SSH/RSA Authentication
To those who frequently use SSH for remote connections, or even configure SSH servers for added security themselves, it is known that the SSH protocol allows for authentication using a better, more secure method than simple password authentication.
This method is called RSA authentication.

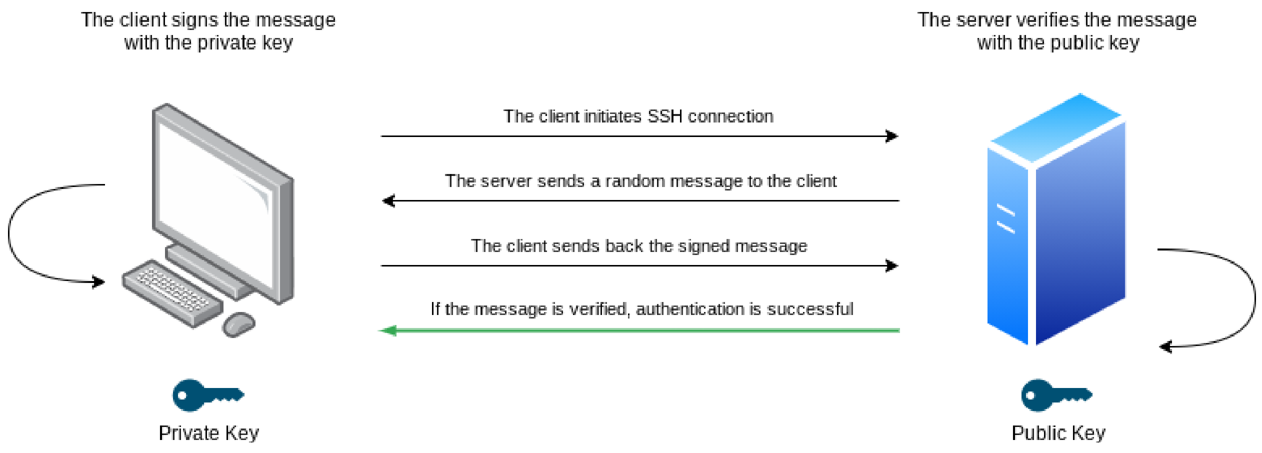
In order for someone to use RSA authentication, they must use an RSA algorithm to create an RSA key pair which is basically two fixed-length random strings.

One of the keys is the private key, which must be kept secret. This key is used by the user to sign messages they send.

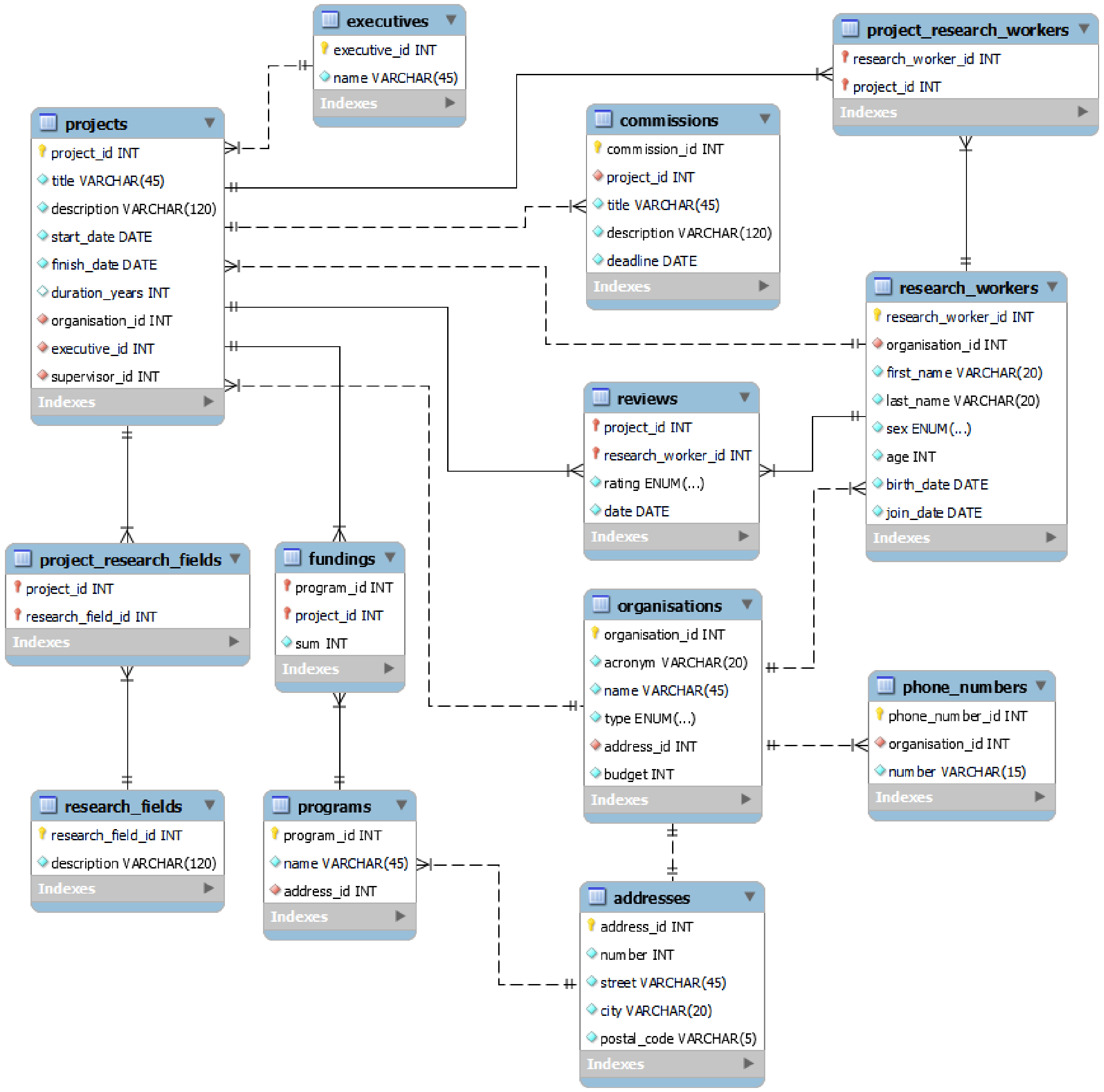
The other key is the public key, which can be used to verify that a user has signed a message. The public key cannot be used to find the private key.

In the case of SSH, when configuring the server you have the option to upload the RSA public key to allow for RSA authentication with the SSH client. You can also completely disable password authentication.

The authentication process is described by the following diagram:


The client signs the message with the private key
The server verifies the message with the public key
The client initiates SSH connection
The server sends a random message to the client
The client sends back the signed message
If the message is verified, authentication is successful
Private Key
Public Key

Database Relational Diagram



The diagram in which we concluded differs from the one recommended in the following manners:

- Projects and fundings are two different entities. A project can only receive one funding from a program.
    - Multiple programs can fund the same project as long as it has been reviewed.

- A research worker can only submit one review for each project.
- The project entity does not have a field describing if it is active or not.
  - This is due to the fact that MySQL does not allow non-deterministic functions like `CURDATE()` in the expressions of derived fields. Because of this we cannot use the project's end date to calculate if it is still active or has been completed.
  - We have, however, included a function to check if a project is still active in our application.
- For the reasons stated above, the age field in the research worker entity is not derived, but rather has to be inserted.
- The `supervisor_id` field in the project entity is a foreign key pointing to the `research_worker_id` field in the research worker entity.

Footnotes

1. Requirement 3.8 is satisfied for 2 or more projects that do not have commissions instead of 5 due to lack of data from the generator.
2. The setting `Miscellaneous/Client-Side Filtering` allows for local filtering of the displayed results without renewing them from the database wherever possible.
   2.1. By default this setting is disabled because all requirements have to be fulfilled using SQL. When this setting is disabled, filtering is achieved using the `WHERE` clause.
   2.2. This is meant to be used in a testing environment where the database is not frequently modified, in order to decrease the number of requests made to the database server.
3. The setting `Miscellaneous/Random Avatars` uses the randomuser.me API to display random avatars for research workers on each refresh. We do not recommend using this on a low bandwidth internet connection.