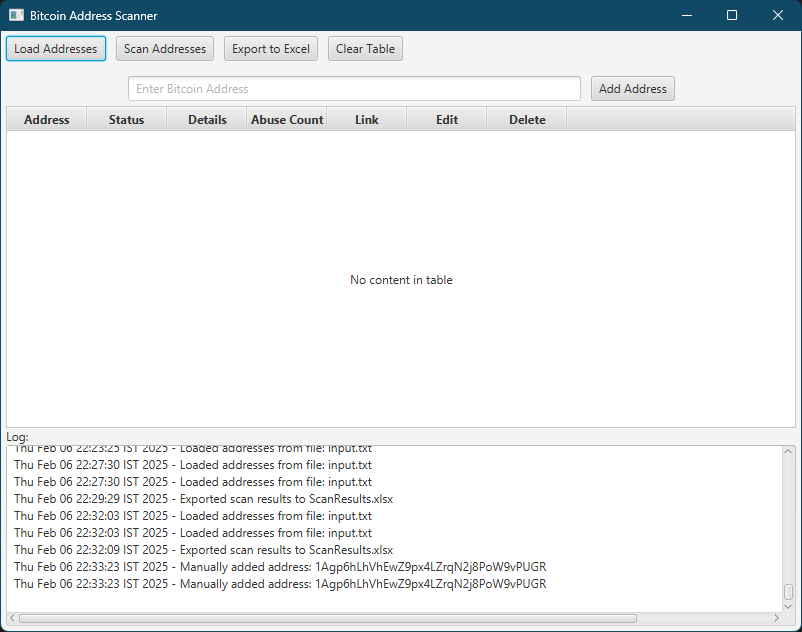
ChainAbuse project

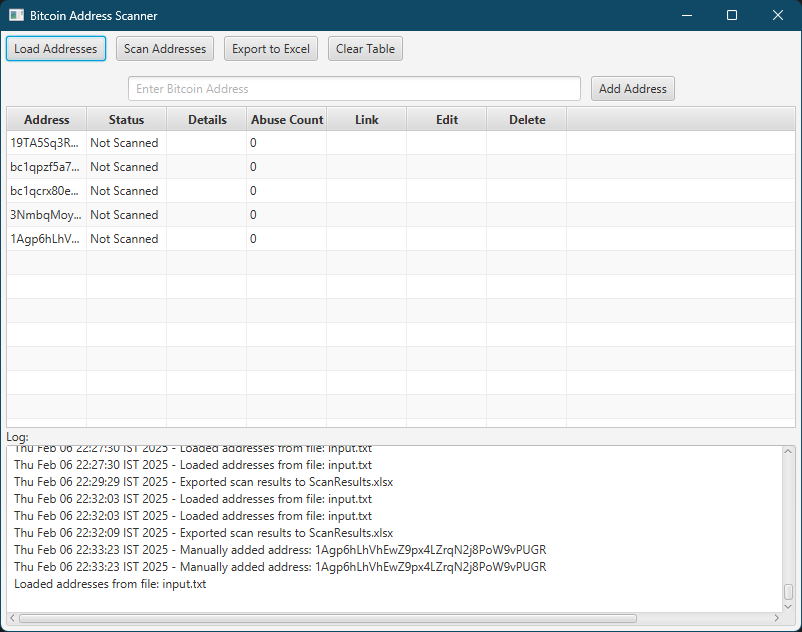
Fredi Bulshtein – 205756794 – [fredi1574@gmail.com](mailto:fredi1574@gmail.com) – 053-2232465  
Sapir Isaac - 318189677  
Almog Elbaz - 213037369

Shaked Oz - 207475591

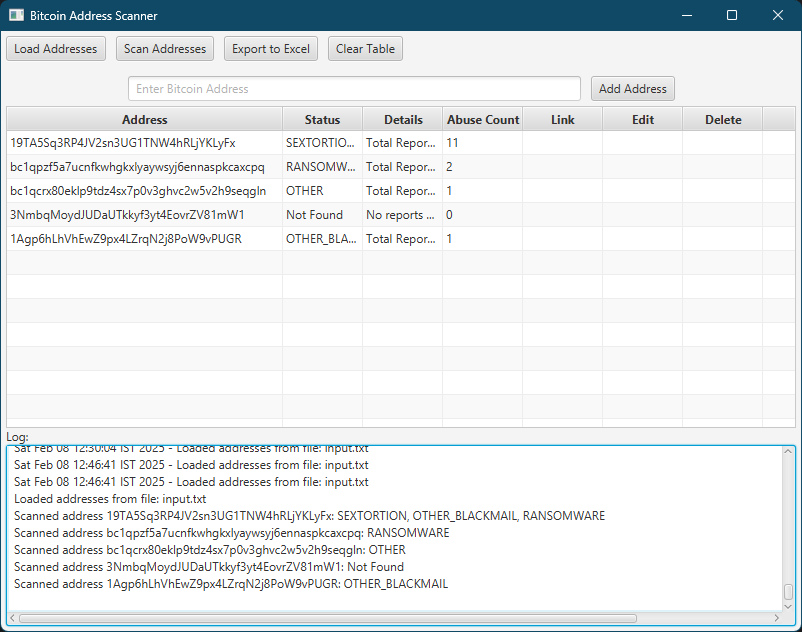
Screenshots:



At first, the table is empty since no address has been added.



We imported the addresses from the “input.txt” file. But we haven’t scanned the addresses yet.



“Scan Addresses” was pressed, and now we see the details of each address.

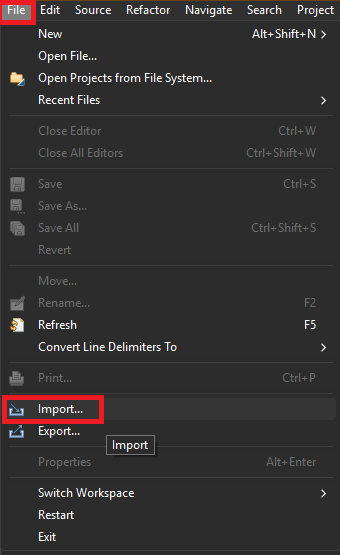
The exported excel file:



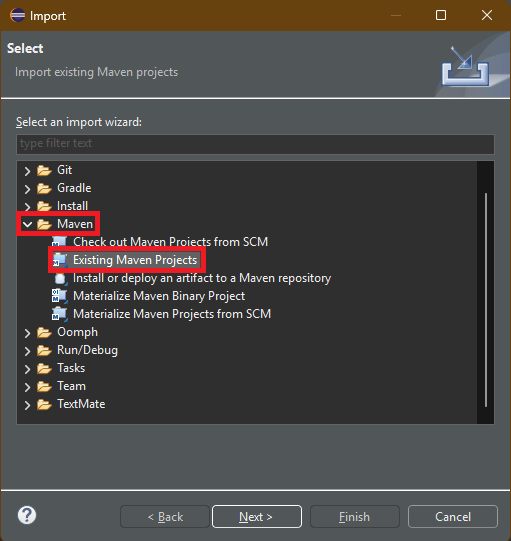
Since our application was built using maven, there are no jar files to be added manually.

If using IntelliJ, just press the “run” button.

If using eclipse, import the project as a maven project (make sure there exists a pom.xml file).

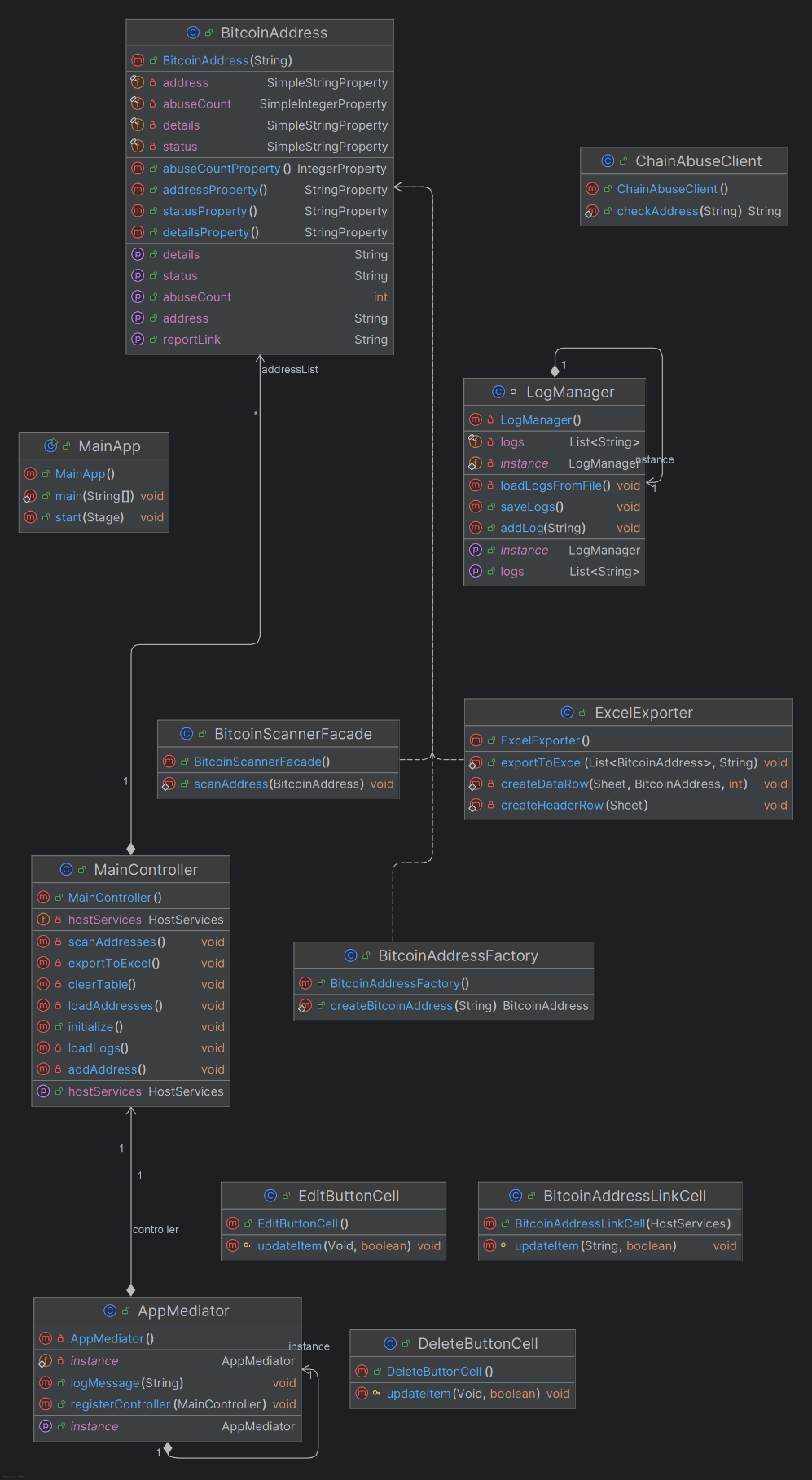
First click on “File”, then “import…”

Open the Maven option and click on “Existing Maven Project”



Open the projects’ folder. The “pom.xml” file automatically adds the dependencies.

Finally run the main file.

Here is the class diagram of our app:

The design patterns we used are:

* Singleton:
* **AppMediator -** ensures a single instance for coordinating communication between components, allowing the MainController to log messages centrally.
* **LogManager -** maintains a single instance to manage logs, ensuring consistency across the application.
* Factory:

Encapsulates object creation, all the logic for creating BitcoinAddress objects is centralized. It is used in BitcoinAddressFactory.java and it helps us by separating the object creation logic from MainController. Now, all the logic is in this file, and any change we need to do, only needed to be updated here.

* Observer:

The observer notifies the UI when a data changes. It is used in **BitcoinAddress**, in JavaFX many fields have listeners implemented, every change in them, will update the UI automatically.

* Facade:

Used in **BitcoinScannerFacade**.

This class provides a simplified interface for interacting with the ChainAbuseClient, abstracting complexities like API requests and JSON parsing, making it easier to scan Bitcoin addresses​

* Mediator:

Instead of components directly interacting with each other, they go through the mediator, reducing dependencies and improving maintainability. In our case the mediator is responsible for the logs of the app, communicates between the **MainController** and **LogManager.**