March 6, 2019

Istvan Zeno Hevesi (izh1g17)

Yavor Edipov (ye1g17)

William Kewell (WK3G17)

Furqan Hashmi (MFH1G17)

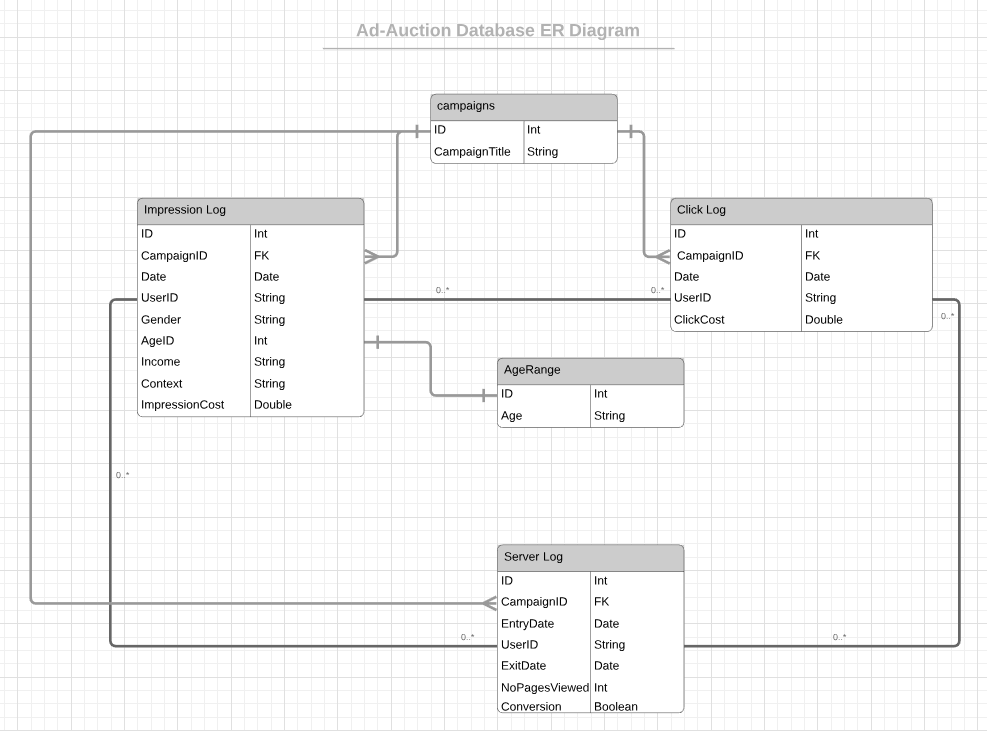
Chris Wang (Lw2N17)

Ad-Auction board Increment 1

Group 13

# Key Design Artifacts and Choices

To begin with, we began the development of application using Java 8. For the project we have decided to use **Swing and AWT** for the UI of the application as most of the team members are familiar with these technologies and this way everybody can easily make contribution. Furthermore, we took advantage of several Design Patterns which allowed us to easily distribute the work and open the application for future extensions. To name them: **Model View Controller and Observer Pattern.** We have also used functional programming in Java **(Streams)** which help with easily and efficiently working on large volumes of data. As requested by the client, we have planned to connect the application with an external database which would allow one to access their data even remotely. What is more, once loaded the data from files into the database, no further loading would be needed and could be instantly worked on. For this purpose, we have decided to use **MySQL Database,** because the advertisers work with structured data which is well suited for such database. This is the ER Diagram we have produced. I believe this will be the final choice, but we are open to changes in the future as well.

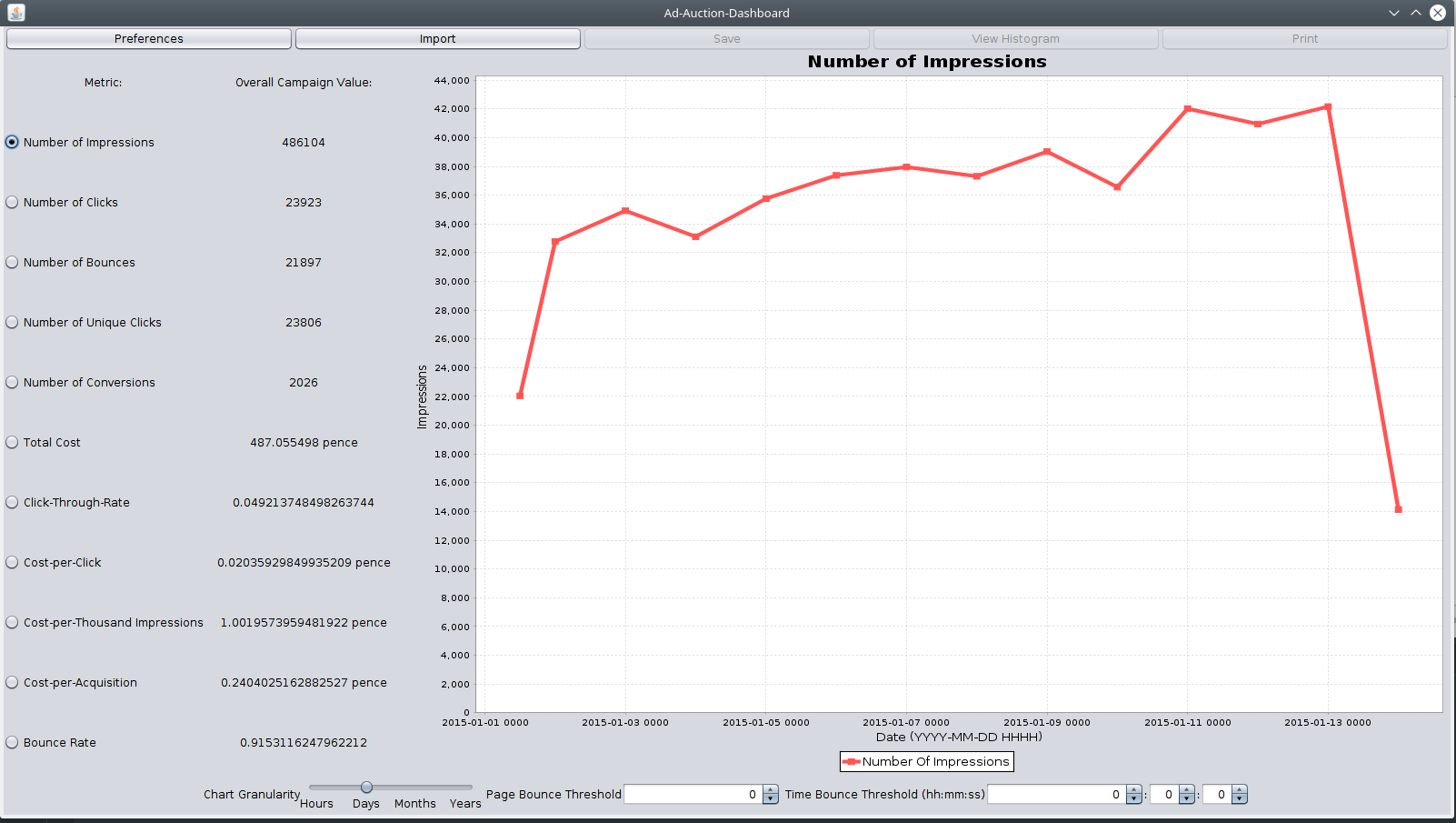


**Impression Log, Click Log and Server Log** all contain **UserID** and the relationship between all 3 tables is Many-to-Many as we have identified that the UserID is not unique.

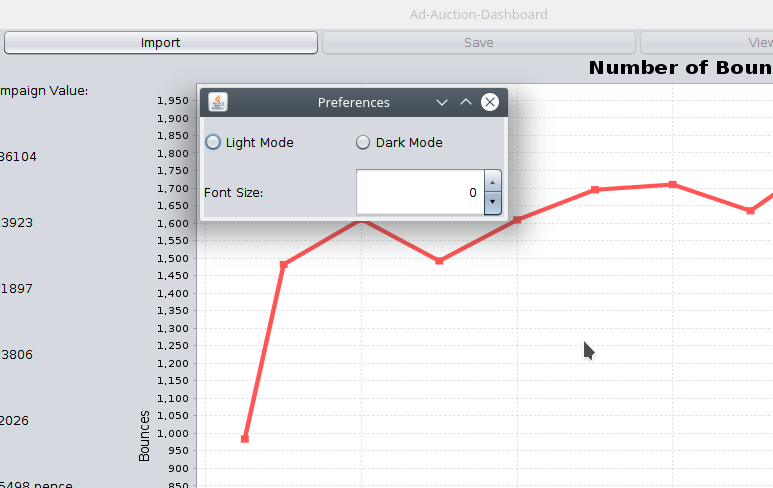
We have decided to separate the **AgeRange** (which is usually given as a string: e.g. <25, 25-34, etc.) into a separate table and take the ID as an indicator of the age range: 1 = “<25’, 2 = “25-34” etc. This way manipulating the data becomes easier because we are comparing Integers. This allows quicker response from the application for visualizing data when applying filters. The **Campaigns** table will keep track of different campaigns loaded so that we can offer a dashboard to the user to choose which preloaded campaign they would work on.

For this Increment we have built a simple GUI which would allow us to test our application and deliver a usable product to the client. In the future the representation of the components and the layout might change in order to help with better User Experience. As it can be seen from the picture, there is a button on top **Import** where the user can upload the data files. Once uploaded, the data is parsed and the metrics of a campaign a displayed.

The user might as well choose which metrics they want to be graphed and control the granularity of the chart, analyze it and apply some simple filters. In the future this will be further extended.

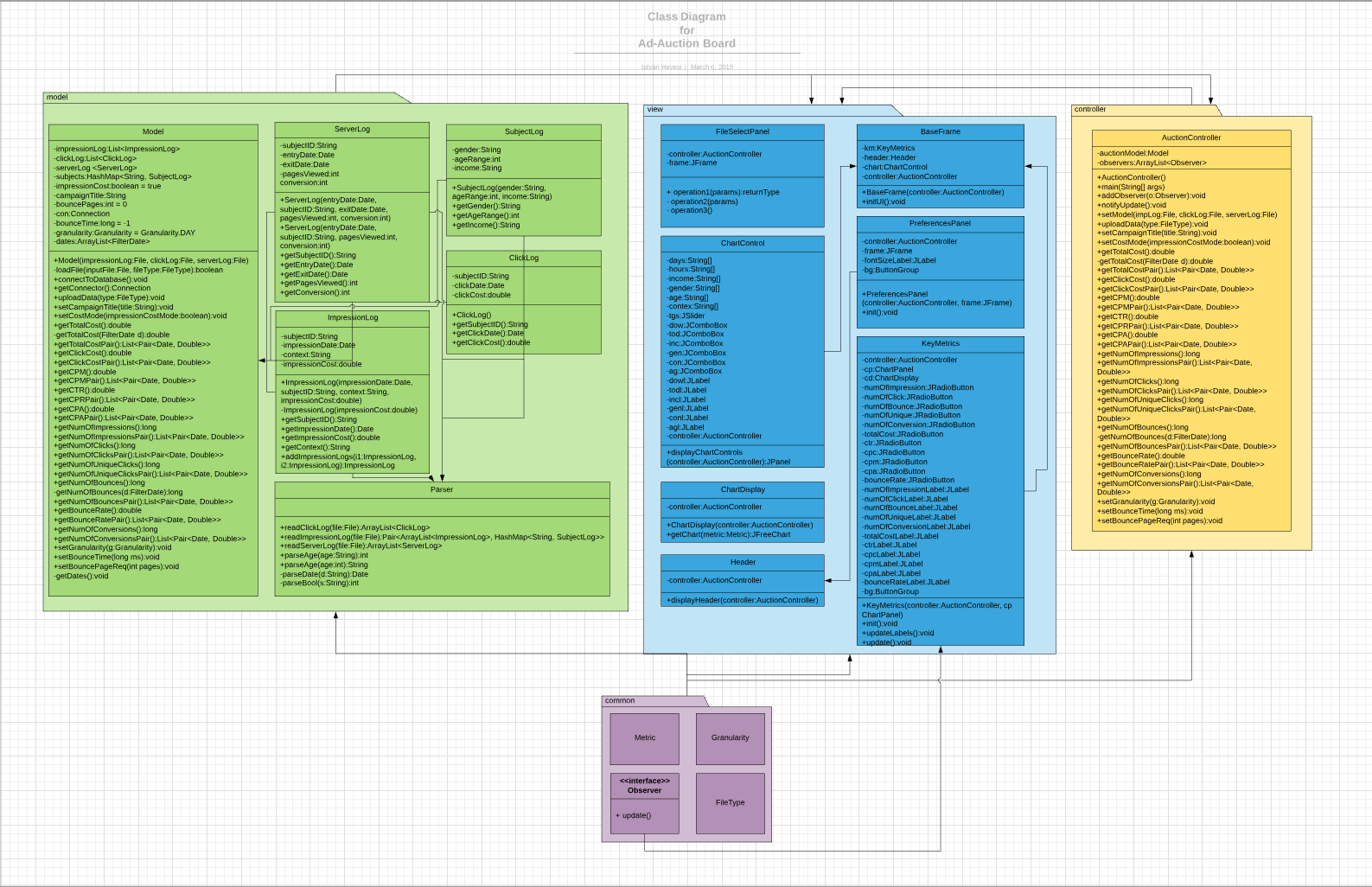


What is more, we have planned in advance to deliver a product that would be accessible to the widest range of people. During development we consider all types of features that would make the product accessible and work on them as well.

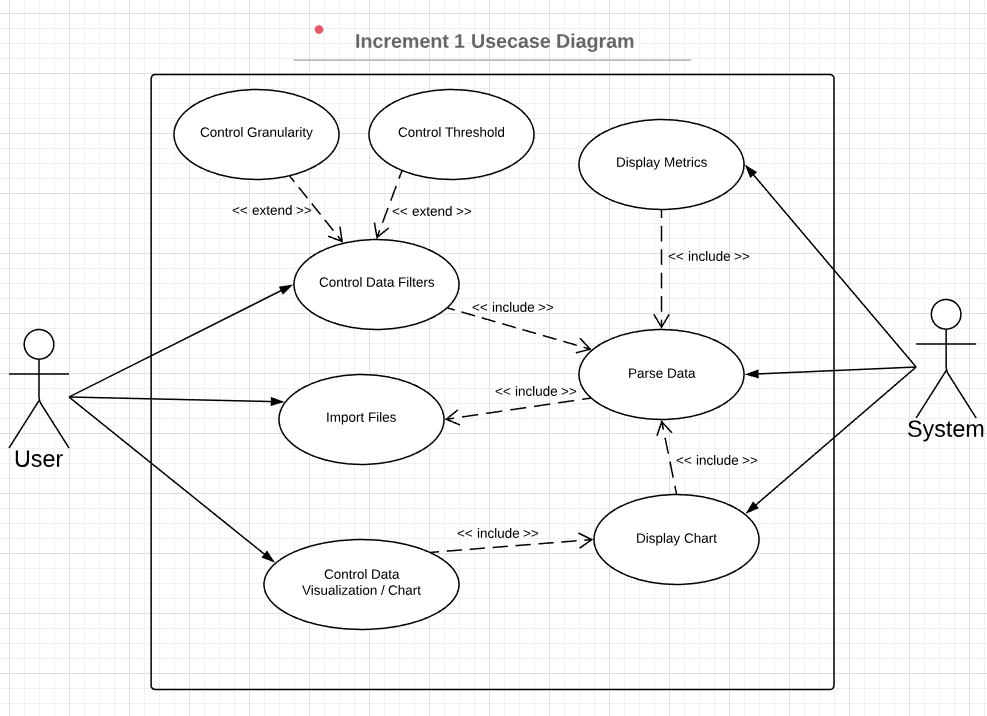


# Key Storyboards

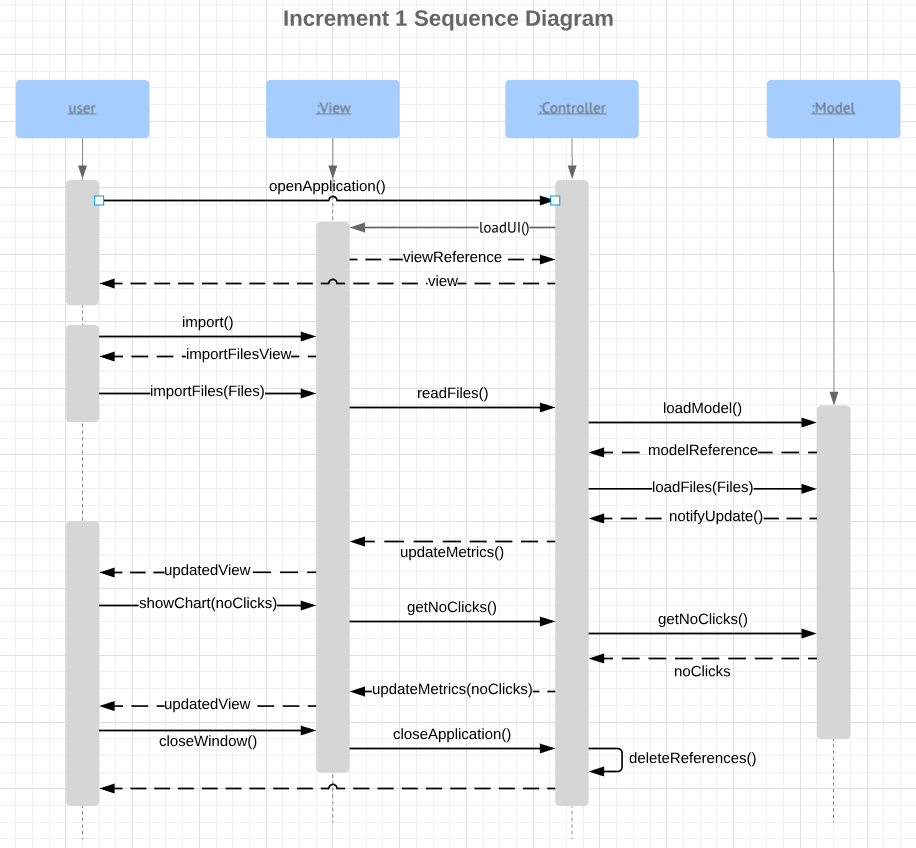
The **UML Class Diagram** describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects. We used it for general conceptual modeling of the structure of the application, and for detailed modeling translating the models into programming code.



We have built a **UML Usecase Diagram** in order to get an overview of the system and how it would perform. It assisted us in picturing the interaction between the user and the system. The user can: Import Files, Control Data Filters and Visualization. Some of the actions do depend on others. For instances, in order for the user to be able to work with the data, it needs first to be parsed. However, in order for the system to parse the data in needs to be imported from files.

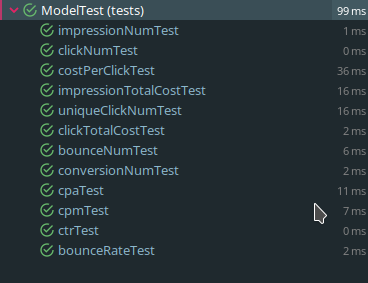


It was useful to build a **UML Sequence Diagram** as well to describe potential actions a user might take. This way we were able to implement needed features with a better understanding of the system and how it is supposed to behave.



# Key Test Outputs

Writing **Junit Tests** helps us prove the accuracy of the application. We have manually computed the expected results of certain operations using other tools and then tested these against the output of our program. As it can be seen from the picture (refer to the tests.ModelTests.java file for more details), the software passes all the written tests.



# Responses to Feedback

We were happy to receive mostly positive feedback. One thing we had missed was to send the document to the client before the review. It took a note of that and for this Increment and further ones, we are sending the document for review to the client in advance. This would help them prepare with any questions that might arise.

# Burndown Chart

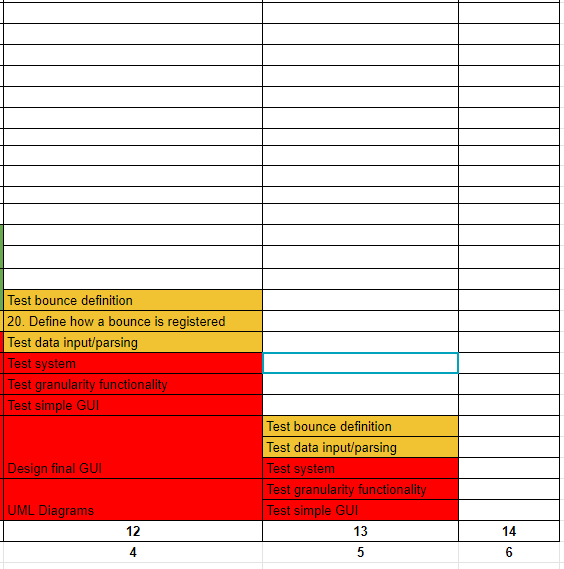
## Increment 1 Burndown Chart







It is important to be noted that we have moved some tasks ***(20. Define how a bounce is registered, Test Bounce Definition),*** which were planned to be done in the Increment 2 to Increment 1. The agile methods allowed for this flexibility and deliver a better product to the customer.



## Increment 2 Burndown Chart

