

DOCUMENTATION

ASSIGNMENT 2

Queue Management Simulator



**TECHNICAL
UNIVERSITY**
OF CLUJ-NAPOCA
ROMANIA

STUDENT NAME: Seserman Victor
GROUP: 30422

CONTENTS

1. Assignment Objective.....	3
2. Problem Analysis, Modeling, Scenarios, Use Cases.....	3
3. Design.....	5
4. Implementation.....	8
6. Bibliography.....	11

1. Assignment Objective

Design and implement a queues management application which assigns clients to queues such that the waiting time is minimized.

2. Problem Analysis, Modeling, Scenarios, Use Cases

a) Analyzing the problem

Queues are commonly used to model real world domains. The main objective of a queue is to provide a place for a "client" to wait before receiving a "service". The management of queue-based systems is interested in minimizing the time amount their "clients" are waiting in queues before they are served. One way to minimize the waiting time is to add more servers, i.e., more queues in the system (each queue is considered as having an associated processor) but this approach increases the costs of the service supplier.

b) Modeling the problem

The queues management application should simulate (by defining a simulation time *tsimulation*) a series of N clients arriving for service, entering Q queues, waiting, being served and finally leaving the queues. All clients are generated when the simulation is started, and are characterized by three parameters: ID (a number between 1 and N), *tarrival* (simulation time when they are ready to enter the queue) and *tservice* (time interval or duration needed to serve the client; i.e. waiting time when the client is in front of the queue). The application tracks the total time spent by every client in the queues and computes the average waiting time. Each client is added to the queue with the minimum waiting time when its *tarrival* time is greater than or equal to the simulation time ($tarrival \geq tsimulation$).

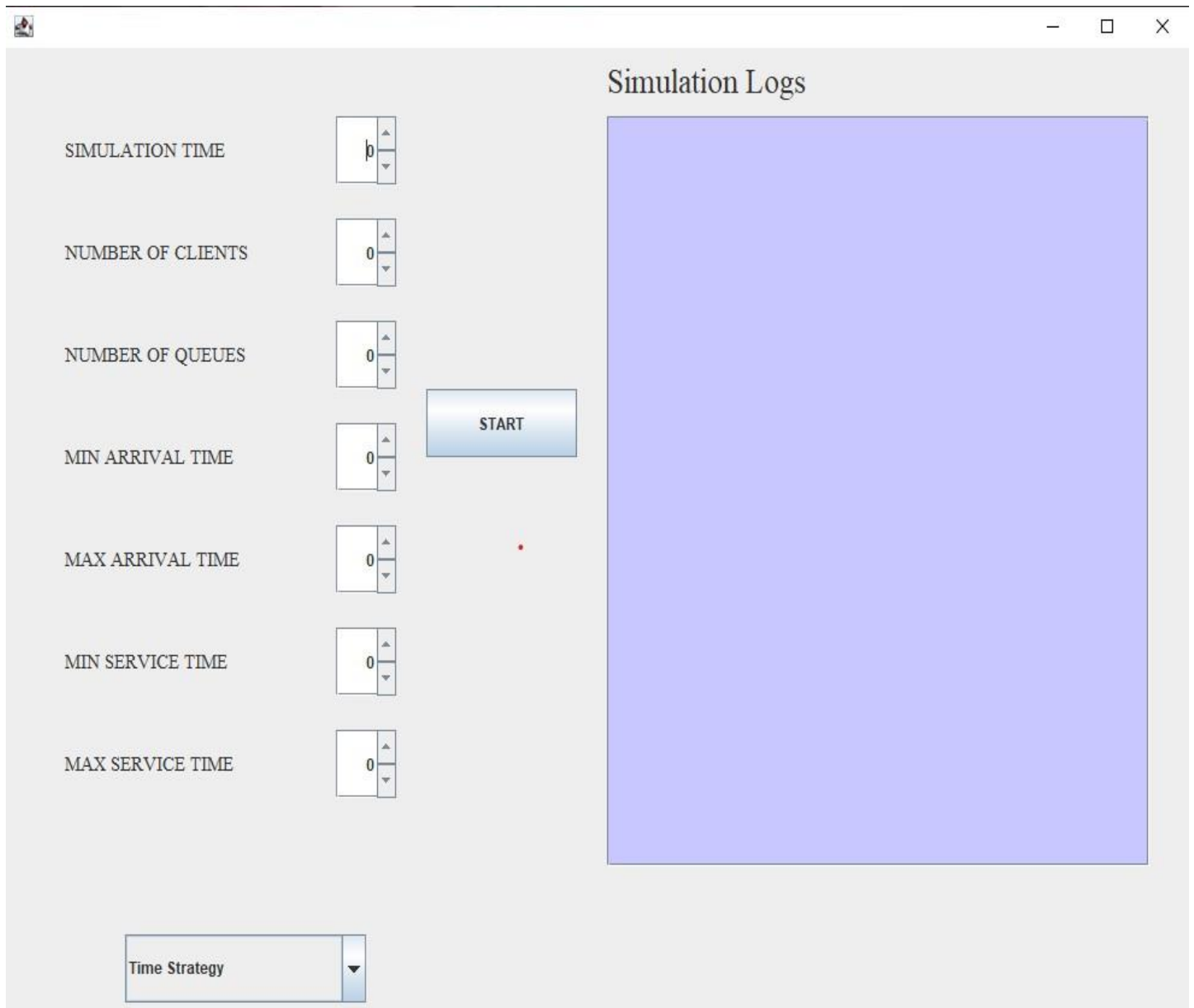
The following data should be considered as input data for the application that should be inserted by the user in the application's user interface:

- Number of clients (N);
- Number of queues (Q);
- Simulation interval (*tsimulation MAX*);
- Minimum and maximum arrival time ($tarrival MIN \leq tarrival \leq tarrival MAX$); -
- Minimum and maximum service time ($tservice MIN \leq tservice \leq tservice MAX$);

c) Scenarios and use cases

A use case is a methodology used in system analysis to identify, clarify and organize system requirements. The use case is made up of a set of possible sequences of interactions between systems and users in a particular environment and related to a particular goal.

The use cases are strongly connected with the user steps. That is the reason why the design of the interface is very friendly and below is the result.



The image shows a software interface titled "Simulation Logs". On the left side, there are seven input fields, each with a label and a numeric spinner control. The labels are: "SIMULATION TIME", "NUMBER OF CLIENTS", "NUMBER OF QUEUES", "MIN ARRIVAL TIME", "MAX ARRIVAL TIME", "MIN SERVICE TIME", and "MAX SERVICE TIME". The spinner controls currently display the values 10, 0, 0, 0, 0, 0, and 0 respectively. To the right of these inputs is a blue "START" button. Below the "START" button is a small red dot. On the right side of the interface is a large, empty light blue rectangular area. At the bottom left, there is a dropdown menu labeled "Time Strategy".

Parameter	Value
SIMULATION TIME	10
NUMBER OF CLIENTS	0
NUMBER OF QUEUES	0
MIN ARRIVAL TIME	0
MAX ARRIVAL TIME	0
MIN SERVICE TIME	0
MAX SERVICE TIME	0

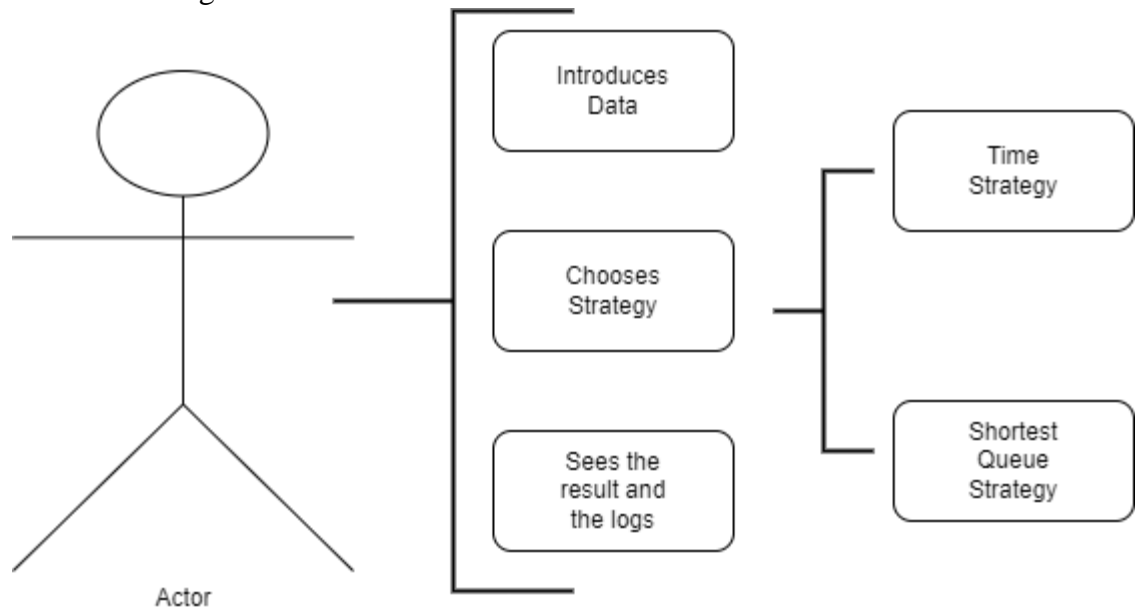
START

Time Strategy

3. Design

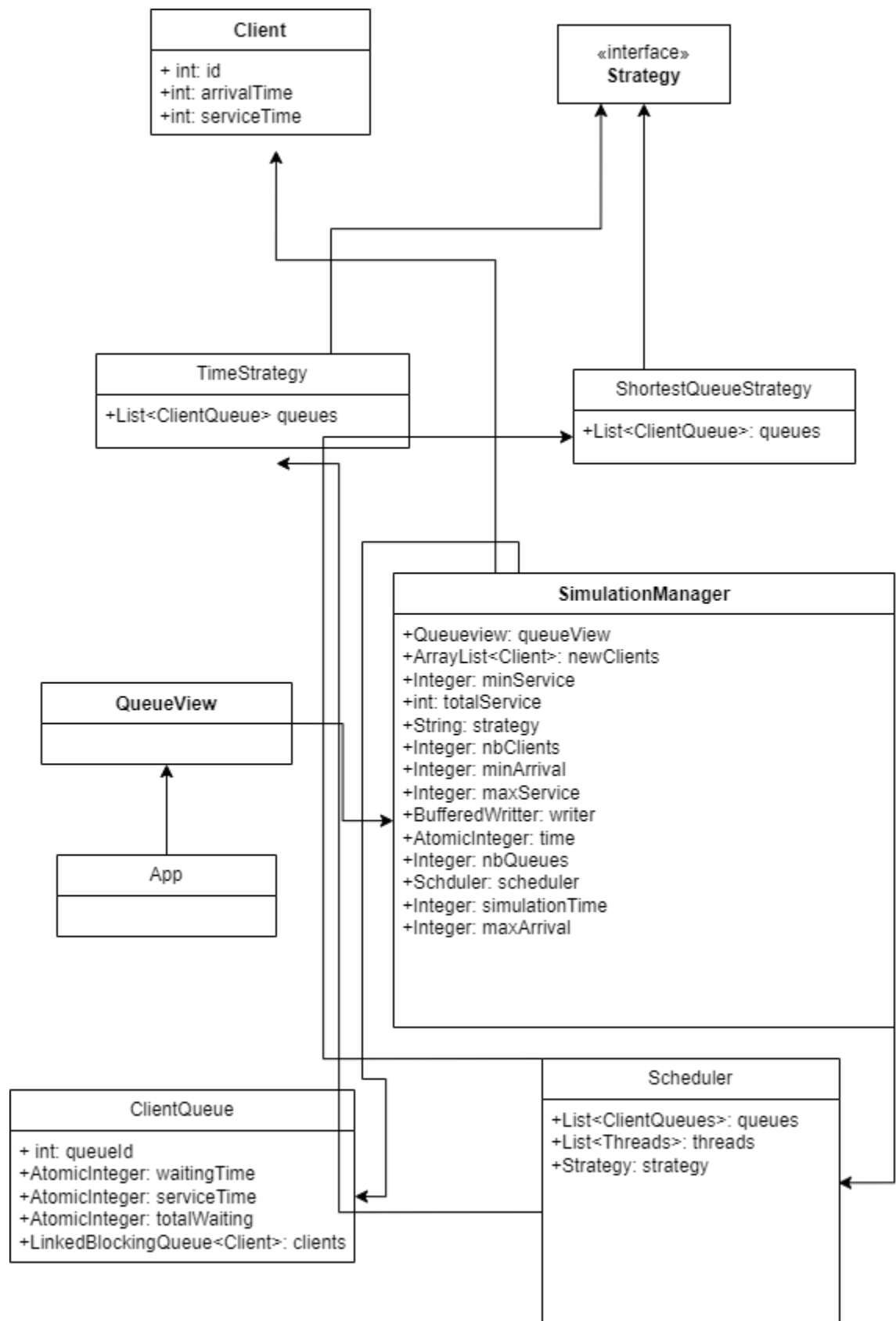
a) Diagrams

Use cases diagram:



The use cases present the user which can perform different actions mentioned above. They can perform several actions on the application and they can also interact with the application in a multitude of ways.

Class Diagram



b) Data Structures

The data structures used in this projects are of two types: primitive, such as Integer, Double and String, and more complex, the only complex data structure used is the LinkedBlockingQueue and ArrayList.

c) Packages

Java packages help in organizing multiple classes into a group such that similar classes can be found in the same place.

In OOP, model-view-controller, or MVC, is the main method of designing projects in a successful and efficient manner. The MVC pattern is widely used in program development with programming languages such as Java, C, C++ etc.

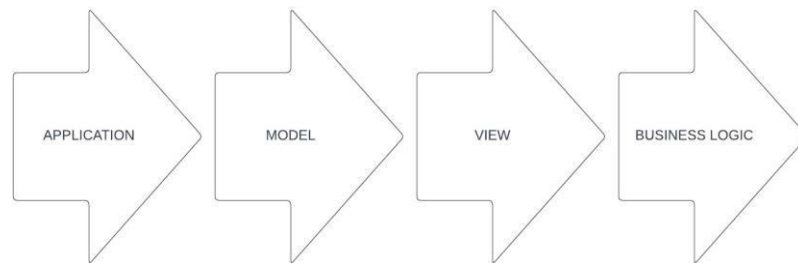
To organize the this project it was used a derived version of the MVC pattern, where the model and the view remain the same, but the controller part is slightly changed, turning it into a package called Business Logic.

Here are the benefits and explanations of each package:

- Model, this represents the logical structure of the project and the high-level class associated with it;

- View, this represents all the user interactions, such as introducing the data, viewing statistics about the simulation and viewing the logs.

- Business Logic: in this package all the algorithms, used for simulating the queues.



Application – not a package, but it can be consider as a imaginary package, here the main can be found, this is the starting point of the application.

Model – in this package the classes Client and ClientQueue can be found, these are the classes which model the whole problem.

View – this is the package which contains the GUI, everything that needs to be seen is found here, this is also the place where the user can input data.

Business Logic – this package contains 5 classes in total, they will be explained later.

4. Implementation

Here is a brief explanation of all the classes and their methods:

- Client

The purpose of this Java class named "Client" is to represent a client with an ID, arrival time, and service time. It contains constructors to set these values upon instantiation, as well as methods to randomly set the arrival and service times. The class also includes getter and setter methods for each of the instance variables. Additionally, the class provides an override of the "toString" method to return a formatted string of the client's attributes.

- ClientQueue

"ClientQueue" represents a queue of clients waiting for service. It utilizes a `LinkedBlockingQueue` to store the clients in the queue, and includes an `AtomicInteger` to keep track of the total waiting time and service time of the queue.

The class includes a constructor that sets the initial values of the queue's attributes, such as its ID. It also includes methods for adding a client to the queue, retrieving the total waiting time, service time, and ID of the queue, as well as returning the clients in the queue.

The class implements the `Runnable` interface, allowing it to be run as a thread. The `run()` method dequeues the clients from the head of the queue and sleeps for the amount of time equal to the service time of the client, decrementing the waiting time accordingly. The class also includes methods to get the number of current clients in the queue and override the `toString()` method to return a formatted string representation of the queue and its clients.

- Strategy

This Java interface named "Strategy" represents a strategy for adding clients to the available queues in a simulation. It declares a single method signature named "addClient" which takes a `Client` object and a `List` of `ClientQueue` objects as arguments. Any class that implements this interface must provide its own implementation for this method to add the client to the appropriate queue(s) based on the specific strategy being used.

- TimeStrategy and ShortestQueueStrategy

These two strategy classes (`TimeStrategy`` and `ShortestQueueStrategy``) are implementing the `Strategy`` interface and are used to define the logic for adding a new client to a queue.

`TimeStrategy`` selects the queue with the minimum waiting time and adds the new client to that queue. This strategy prioritizes the queue with the shortest waiting time for the client.

`ShortestQueueStrategy`` selects the queue with the shortest number of clients in the queue and adds the new client to that queue. This strategy prioritizes the queue with the shortest line for the client.

Both strategies provide a different approach to queue management and can be used interchangeably based on the needs of the simulation.

- Scheduler

The `Scheduler`` class is responsible for creating and managing the client queues and the threads that run them. It also defines the scheduling strategy used to add clients to the queues. Here's a breakdown of the class:

- `queues``: A list of `ClientQueue`` objects representing the queues.
- `threads``: A list of `Thread`` objects that run the `ClientQueue`` objects. -
- `strategy``: A `Strategy`` object representing the scheduling strategy used to add clients to the queues.
- `getQueues()``: A getter method for the `queues`` field.
- `Scheduler(int nrQueues, String strategy)``: The constructor for the `Scheduler`` class. It takes two parameters: the number of client queues to create (`nrQueues``) and a string representing the scheduling strategy to use (`strategy``). The constructor creates the client queues and threads, sets the strategy based on the input string, and starts the threads.
- `toString()``: A method that returns a string representation of the `Scheduler`` object. The string contains the string representation of each `ClientQueue`` object in the `queues`` list, separated by newlines.

Overall, the `Scheduler`` class provides an interface for creating and managing client queues and choosing a scheduling strategy for adding clients to the queues.

- SimulationManager

The `SimulationManager` class creates a `Scheduler` object that manages a set of `ClientQueue` objects. Each `ClientQueue` object represents a queue in the system. The `SimulationManager` class generates random clients and adds them to a list of

waiting clients. Clients are then added to the appropriate queue based on a scheduling strategy. The SimulationManager class logs statistics such as average waiting time, average service time, and peak hour to a text file. The program has a GUI implemented using Java Swing.

The program first sets up the GUI interface using Java Swing components. The user can input the number of clients, number of queues, simulation time, minimum and maximum arrival time, minimum and maximum service time, and scheduling strategy. When the user clicks the "Start Simulation" button, the program begins simulating the queue system.

The program then creates a SimulationManager object and sets the values of the input parameters using the setValues method. The SimulationManager object then generates random clients using the generateClients method and adds them to a list of waiting clients.

The program then enters a while loop that runs until the simulation has reached its maximum time. In each iteration of the loop, the program prints the current time of the simulation to the log using the printWaitingClients method. The printWaitingClients method first checks if there are any waiting clients. If there are, it prints the waiting clients to the log. If the current time of the simulation is greater than or equal to a client's arrival time, the client is added to the appropriate queue based on the scheduling strategy using the Scheduler object.

After the waiting clients have been added to the appropriate queues, the program iterates over each ClientQueue object in the Scheduler object and processes the next client in the queue. If a client has finished being serviced, the program logs the service time and removes the client from the queue.

The program also logs statistics such as average waiting time, average service time, and peak hour to a text file using the writeSimDetails method. Finally, when the simulation has completed, the program closes the log file using the closeWriter method.

- QueueView

This is a Java Swing class that represents a GUI for a simulation of queues. The GUI allows the user to input various parameters for the simulation, such as the number of clients, number of queues, and arrival times. The `start` button triggers the `actionPerformed()` method, which creates a `SimulationManager` object and passes the user inputs to it.

The `SimulationManager` class is not shown, but it is likely responsible for running the simulation and updating the GUI with the results. The `JTextArea` `logText` appears to be where the simulation results are printed.

The `JComboBox` `strategy` allows the user to choose between a time-based strategy and a shortest queue strategy. The `start` button triggers the simulation using the selected strategy.

Overall, this class appears to be a functional GUI for running simulations of queues. However, without knowing the implementation details of the `SimulationManager` class, it is impossible to evaluate the correctness or efficiency of the simulation.

- App

This is a basic Java Main class with a main method that initializes a `QueueView` object from the View package. This is the place where the applications starts.

5. Conclusions

This project was tough, the requirements were harder then expected, but I did learn a lot from this, using threads and I now understand better some concepts of the java collection. Also, I think this project can be very helpful beyond the subject studied at the university, since it is a pretty complex project, it will probably look very well on a CV.

6. Bibliography

- [Wikipedia](#)
- [Programming Techniques Lectures and Laboratories](#)