1. INTRODUCTION

Purpose

This software design document describes the architecture, and design of the system, which tries to utilize smart algorithms to create a better public transport solution.

Scope

Our project's initial scope is a single urban city.
Today users of the public transport system are sentenced to spend a lot of time on busses, passing through many unneeded stops as the busses are constrained to a specific route.
Also, bus lines may be quite empty regularly.
We aim to develop an alternative transport system, where bus routes are ever-changing by demand, rarely empty, and are time efficient.

Overview

Our product aspires to create a better solution for public transport than what the existing infrastructure can suggest.
The current solution is not good enough – this is shown by many people choosing to get private vehicles, unlike many other parts of the world where public transport is the default choice.
As we see it, the problem is that it is too slow, and this is caused by many unnecessary stops, as well as routes that cannot change dynamically to avoid traffic.
The empty buses driving around, and many private cars used every day are causing both air pollution and frustration and time-wasting traffic jams.
We aim to improve on that using smart algorithms utilized in a dynamic public transport system.

2. System Overview

Functionality:

Client-side:

User side:

- input_validation(src ,dest): @return boolean
  Making sure that the user src , and dest is well defined , and understood by our system.

- order_future_ride(user, src ,dest , arrival_time ,arrival_date)
  @return ride_id

  Processing users application for a ride, checking fields validity, sending data to the server, and wait until the server returns “ride_id”.
  If all that happened, return the created ride_id.

- Cancel_ride(user, ride_id)
  @return Boolean
  if the ride_id match the user, and there is more then 24h to the ride, refund the user, and cancel the ride from the system.
  If all that happened, return true.

- More functions TBD

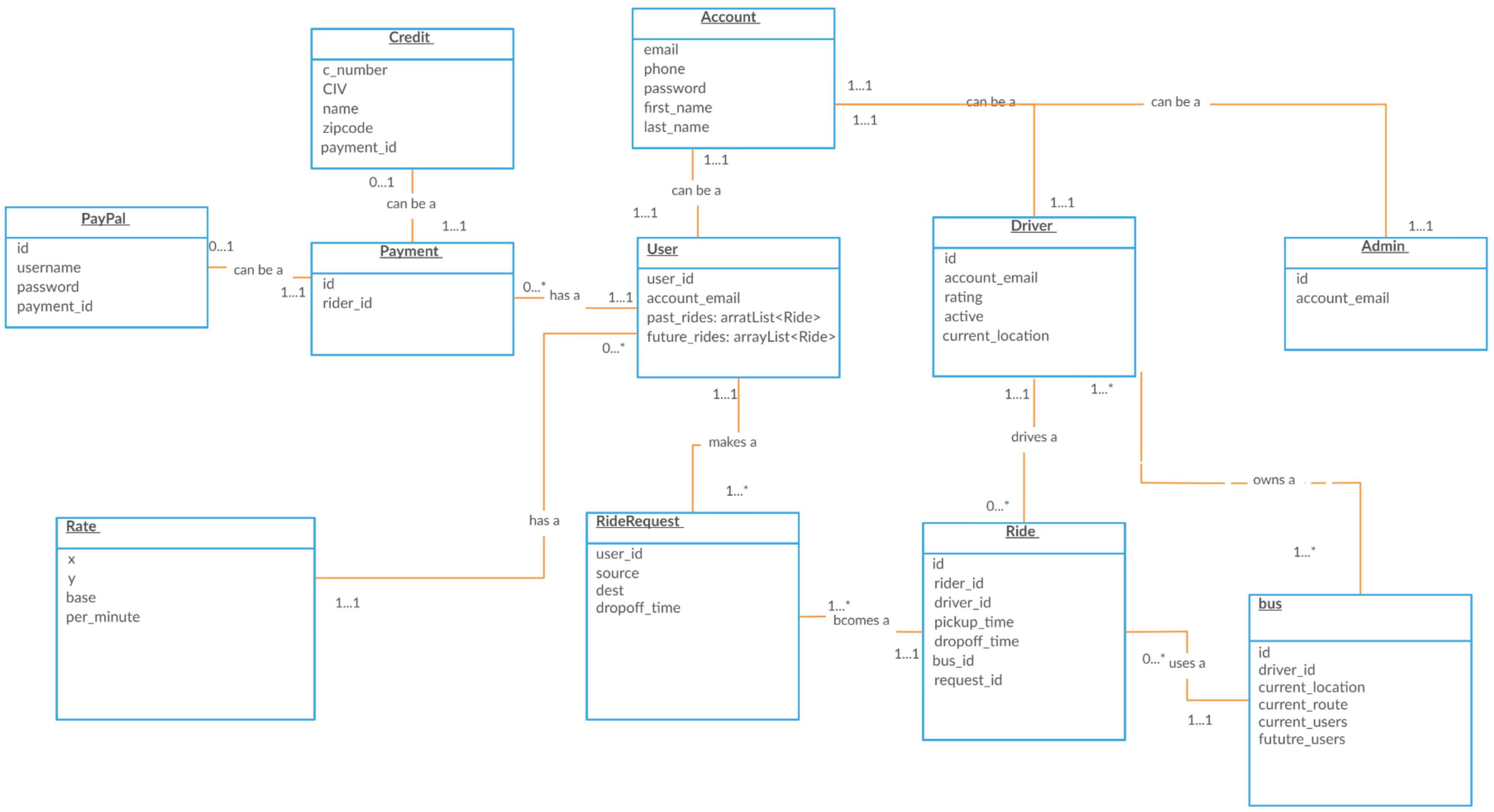
Server-side:

- process_ride_order(user, src ,dest , arrival_time ,arrival_date): @return ride
  The data sent to the server is a valid data (checked in user side) and the server will calculate the best “ride” possible for the users data, returning the matched “ride_id”

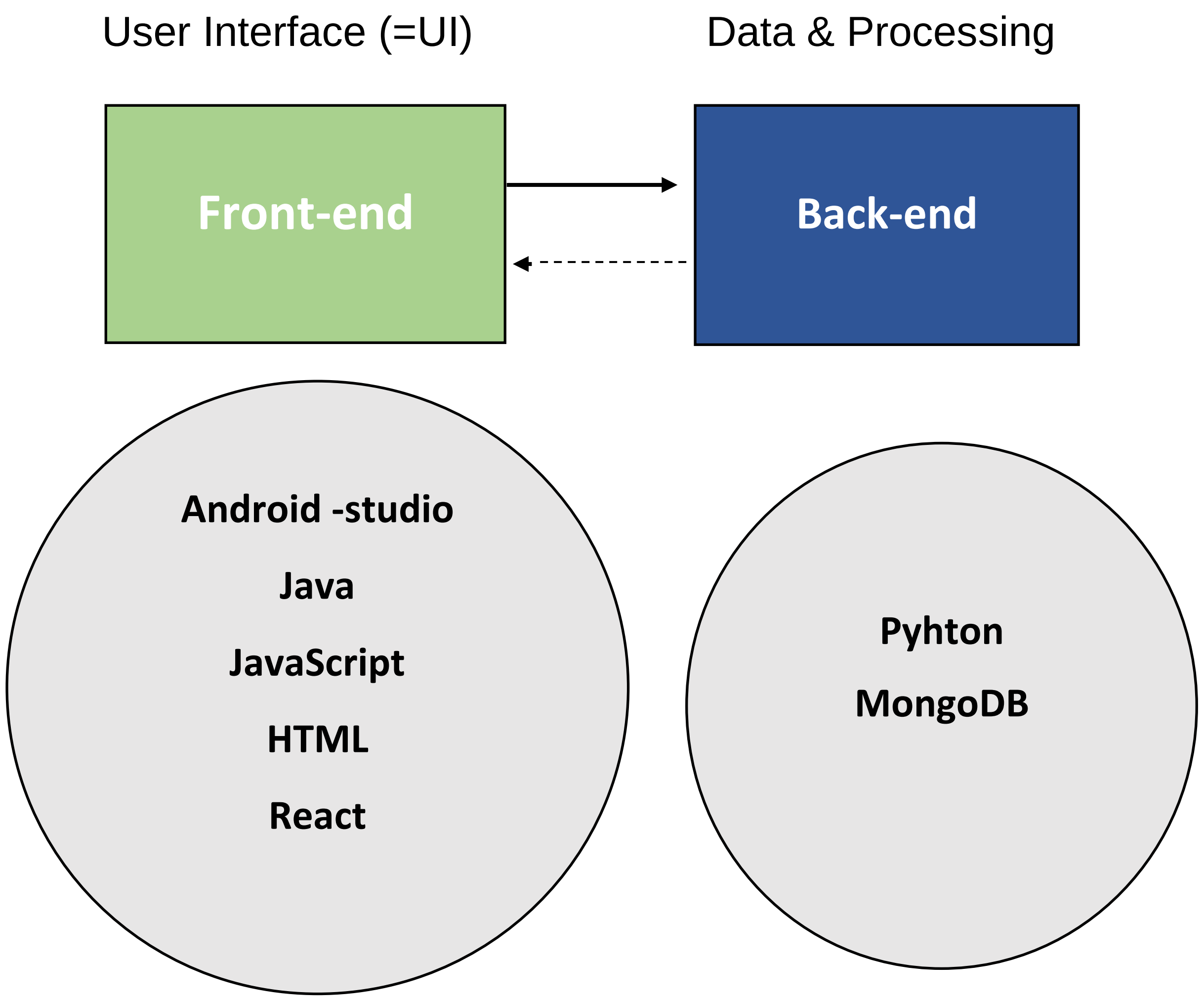
- Cancel_order(user,ride_id)@return Boolean
  The server checks if ride_id and the user match, if there is less then 24h before the ride, fully de-allocate this the ride from the system.
- More functions TBD

בע"ה

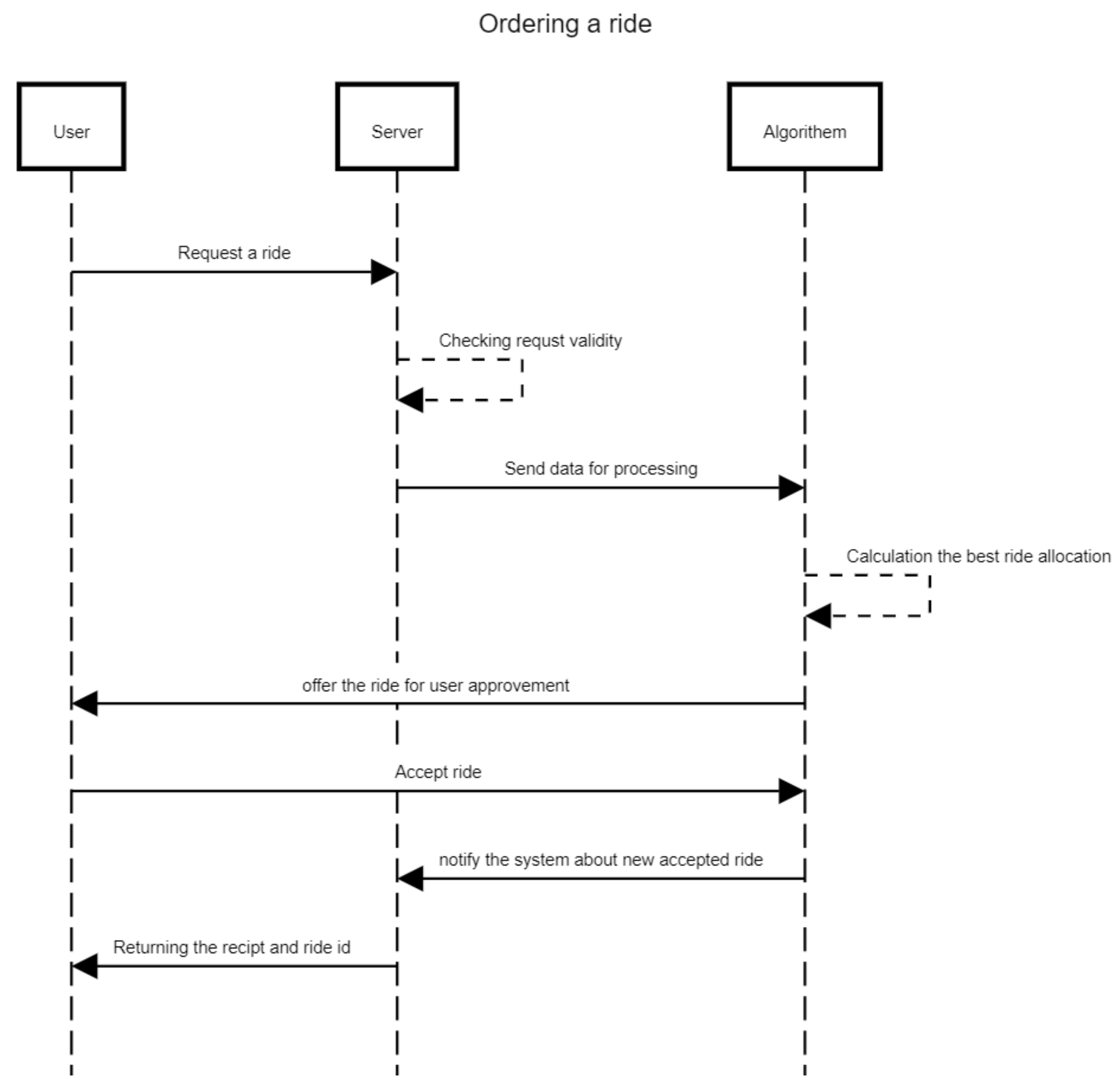
Object Diagram:



3. System Architecture



3.2 Decomposition Description


Ordering a ride
User
Server
Algorithem
Request a ride
Checking requst validity
Send data for processing
Calculation the best ride allocation
offer the ride for user approvement
Accept ride
notify the system about new accepted ride
Returning the recipt and ride id

4. DATA DESIGN

Data design is still a work in progress as we learn MongoDB and other candidates for our data storage.

5. HUMAN INTERFACE DESIGN

5.1 Overview of User Interface

First time: a new User is created with an email & password for authentication.
And enters the preferred payment method.

General use: The user is authenticated, and can view his future and past rides if they exist.
A button “Order a ride” can be pressed to start ordering.

The User chooses his pickup location, desired destination, and preferred ride time and date, and sends it to the system.

The system returns the best ride match, as “pickup station”,” pickup time”,” drop-off station”, and ”drop-off time”.

if a user agrees to these parameters, his ride is updated in the system and he receives a receipt and ride_id.

5.2 Screen Images

Will be added when the system will be built.

6. Division of Work

Both of us, Roie Malykin & Ofir Peller, are working through all the stages together. We see this as a great opportunity to gain experience and many different systems.