**<https://www.cs.uoregon.edu/Classes/16S/cis422/docs/proposal.php>**

**Purpose of the proposed system**

The purpose of the Safe Ride project is to create a web-based app and an accompanying android app that makes it easier for the users to make a reservation for a ride with Safe Ride at night. The intended audience for this app is the students, faculty and staff of the University of Oregon and the employees of Safe Ride.

**Stakeholders**

* Users:
  + Riders: Efficiently request a ride over the phone or computer
  + Dispatch: Have request information clearly laid out for ease of scheduling rides
* Other stakeholders.

**System context**

We are building our program with HTML, CSS and Java Script on our front-end and then Express, Mongoose and Mongodb on our back-end. For this updated project we plan to add a map feature to it so we will be using Google Maps API.

Your software will generally exist in a context of other software, hardware, and human systems. This might include:

* For a video game: Hardware and software platforms (e.g, Xbox, Occulus Rift, Unity) on which the game will run, or distribution through Steam.
* For a web application: Hosting platform, either specifically (e.g., Digital Ocean, AWS, Heroku) or the characteristics that your hosting platform must have (e.g., support for Python 3 and SQLite).
* For any application that uses an external service or API like the Google Maps API: What information your application sends to the API or service and what it receives.

**Behavioral requirements**

Use case: request a pick up:

*Input fields*

- Name

- UO ID number

- Phone number

- Party size

- Desired pick-up time

- Pick-up location

- Drop-off location

- Any other additional information (like having a bike)

Use Case: dispatcher retrieve request information:

- Go to Saferide & submit login information

- Click button to retrieve list of requests

- The above information in list form of all rider requests

**Other requirements**

- Only dispatchers will be only ones allowed to see who has requested a ride and where they are going

- Allow an ‘undue’ / ‘edit’ / ‘delete’ capabilities for both the riders and dispatchers

- Adding Google mapping to the web-based app to make it easier for riders to say where to be picked-up/dropped-off.

Many requirements of a software system may not be described as required behaviors for particular use cases. This includes in particular certain *negative* or *restrictive* requirements, such as “*Only authorized dispatchers may view ride requests other than their own*.” In addition, non-behavioral requirements may include properties about the development, build, or deployment processes, which are not directly visible to users of the system.

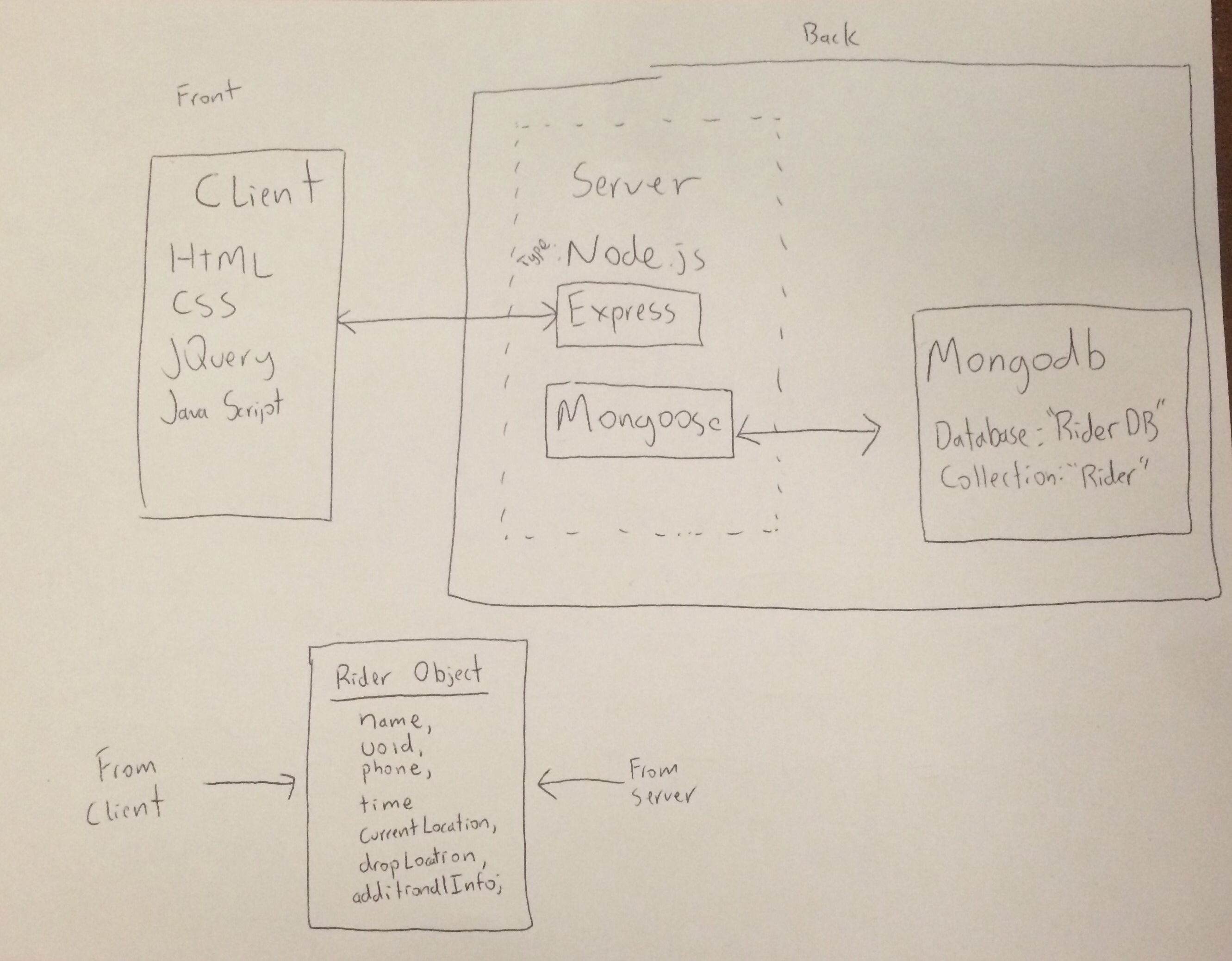
**Value**

This app will make it not only easier for students, faculty or staff to request a ride with Safe Ride but will also make it easier for dispatchers to organize rides in the most efficient way possible. This web-based app will become most useful during the busiest times for Safe Ride every night, 9-10pm Sunday-Thursday and 8-11pm Friday-Saturday. It will make it so riders will be able to get a confirmation as to whether or not they can schedule a ride that fits into the schedule. It will also make scheduling smoother for dispatchers in the office during these busy times.

Why is the system you plan to build worthwhile? What is its value to the stakeholders described above? For a Safe Ride app, this would be a short explanation of how the application will benefit riders and dispatchers. For a game, it would be a short explanation of what will be attractive and unique about the game.

**Design Concept**

Diagram:



Consider this a brief argument that your project is feasible given your resources, including time available. Usually this will include:

* Architectural design sketch. By “sketch” I do not necessarily mean a diagram, although a diagram may be useful. With or without a diagram, there should be a brief textual description of what the major components of your system are, and how you plan to build them (e.g., you might say that a data store will be implemented as a module wrapping an interface to MySQL).
* Main dependencies. These may be at the level of subsystems and major components, but usually they will be most useful at the level of features or capabilities of those components (feature X of component C requires capability Y of component D), because dependencies at this level help determine potential *build orders*.
* Roughly assessment of the work and risk of each major component. Where there is most risk, it may be useful to break down the work and risk by feature (e.g., “we are confident of building a basic 2D scatter plot quickly, but consider the 3D holographic display a risk for completion in the second week”).