RIDESHARE: CONNECTING TRAVELERS

EE 364: System Design

Yoni Ben-Meshulam
Garrett Cooper

Team Venture

Dr. Bruce McCann

December 7, 2007

TABLE OF CONTENTS

LIST OF FIGURES

1. Introduction

Rideshare aims to connect drivers who seek passengers with passengers who seek drivers. It will do so in an efficient manner, which minimizes travel time and distance, as well as taking into account the preferences of customers.

In a time when energy and fuel efficiency are pressuring people to leave their car at home and seek alternatives, our application can help people coordinate rideshare programs and plan trips efficiently and effectively. People who share rides regularly will significantly reduce their ecological footprint and save on travel expenses.

The target audiences can be divided into two types:

- Travelers who are interested in a regularly-scheduled rideshare to their place of work or school, or for their children.
- Travelers who seek one-time partners for long (i.e. cross-state) or short (i.e. grocery store) trips.

Many people travel alone in their cars, with much excess capacity wasted. This waste is compounded by the increasing price of gasoline. People who seek alternatives frequently find mass transit to be unreliable and underdeveloped, and must continue to waste their own and the world's resources.

Our solution will enable individuals to communicate their travel needs and find others who are compatible with those needs. It will eliminate much of the coordinating tasks from the rideshare paradigm, and subsequently increase the likelihood that people will choose rideshare. Our hope is that this product will help facilitate a paradigm shift from single passenger travel to a ride sharing culture.

In order to accomplish this, our project must meet certain basic criteria. These criteria will ensure its proper functionality, reliability, and maintainability. In the following sections, we establish this set of criteria, which will guide us in architecting and developing the application. We then lay out the web application design, user interface design, cost analysis, and test plans.

2. Needs Analysis

In order to successfully implement our rideshare application, it will be necessary for each facet of our system to adhere to a set of requirements. The following are what we consider to be basic requirements for a number of crucial elements of the system.

2.1. Matching

When matching riders and drivers, we must implement an algorithm (a logical set of steps which reach a solution to the problem), which generates the most efficient results. Efficiency is a measure of a number of parameters, foremost of which is maximal matching, i.e. the highest number of customers receiving rides. Maximal matching is crucial for the results to be effective and useful for our customers. Route lengths and times must also be minimized by the algorithm, and suggested rideshares should be viable, intelligent solutions. Timeliness of results is also important, since an effective result which doesn't complete in a timely manner is useless. This last requirement can be quantified as an algorithm which runs in polynomial-time, as opposed to exponential-time. This means it will complete some time tolerable by humans. Exponential-time algorithms can potentially take longer to complete than the history of the universe.

The algorithm should also utilize external inputs efficiently. This can be quantified as an algorithm which minimizes route requests to Google-Maps by reducing the search-space, i.e. the number of possible valid matches. For example, if a driver is going from Austin to Houston, the algorithm would refrain from evaluating routes for which the riders' beginning and end points aren't in Texas. Those that are in Texas can further be reduced to riders' who are in central Texas, etc. Once this pre-processing has been completed, we would evaluate only those combinations within the limited search space which has been generated.

2.2. Server

Our final product will need a server that can support the projected bandwidth and storage requirements. The server must have an application server, preferably Apache, installed on it. Apache can run our implementation, which uses an SQL database and the programming language Ruby. We will need an application development environment such as Eclipse, which supports web

application development, including Ruby, SQL, and JavaScript. Our choice must be evaluated on its reliability, effectiveness, usability, and cost. Within this environment, we will need to enforce good programming practices, including a secure wiki and file server for collaboration, naming conventions, revision control, and modularization of programming tasks.

2.3. Software

The final product will also need to meet certain reliability and availability parameters. Reliability is the measure of how stable the program is. An unstable program will crash or generate erroneous results, whereas a reliable one will perform with minimal interruptions. The program should also be free of bugs. Availability is the measure of how much down-time, measured as the percentage of time during which our website is unavailable, we can expect to have. These will both ultimately depend on our ability to test for problems early on, and be able to quickly fix any bugs.

Rideshare needs to implement a solid user interface. This includes designing an intuitive menu and a friendly graphical interface with usability in mind. We must minimize the number of menu levels, as well as the number of operations required by users.

It is also imperative that we establish application-level security in order to protect our users' privacy, their personal computer, and the application server from malevolent entities. This requires that we use a secure web server and design all incoming and outgoing communications with security concerns taken into account. We must evaluate the web server and implementation methods based on this requirement.

A good testing platform will be necessary in order to evaluate each module of the application. To accomplish this goal we must design the software with testing in mind, also called design-for-test. Each testing requirement must be quantified in terms of our final software architecture. We have included an extensive testing description in section 11.

All of these requirements will need to be incorporated into a high-level application architecture and module-level implementation. At each point in the design and implementation phases, we will need to assess our adherence to these requirements.

Assuming these needs are met, we will need a broad user-base in order to ensure that all users actually find ride-share solutions. Given our application's social nature, the more people we have in our user base, the better the solution we will be able to provide in terms of cost, distance, and compatibility.

3. Deliverables

At the culmination of the project, we will provide the following set of deliverables, which will encompass the totality of the project, and allow a peer to completely understand our application and implementation methods.

Foremost, we will provide the functional rideshare application running on a server, which users can utilize to find and offer rideshares. We will offer documentation of the high-level software architecture, as well as the low-level module implementation. The actual code will also document the programming structures and functions used for implementing the low-level modules. We will also provide a comprehensive demonstration of this application.

In addition, we will provide a use-case diagram for the application, defined as "a type of behavioral diagram...designed to present a graphical overview of the functionality provided by a system in terms of actors, their goals — represented as use cases — and any dependencies between those use cases." [1]

In order to demonstrate the practical functionality of the website, we will provide a site map, including the menu architecture and description, appended by a link graph showing the relationships between different pages of the site.

We will include a maintainability document to ensure the proper operation and maintenance of the site. This document will cover the information necessary to implement the application on a server, including a connectivity graph of application components, bandwidth and storage requirements, compatibility issues, and security requirements.

4. User Interface Requirements

What sets our project apart from other websites, such as eRideShare [2], is the user interface. By giving the map a central role in the user interactions, we create a unique user experience that will facilitate simple and effective ridesharing. We aim to display routes similarly to Google Transit [3], with menu options available as a side-bar. Features like zooming and turn-by-turn directions will be implemented in the same manner. Figure 4.1 shows a typical map-based application on Google's own website.

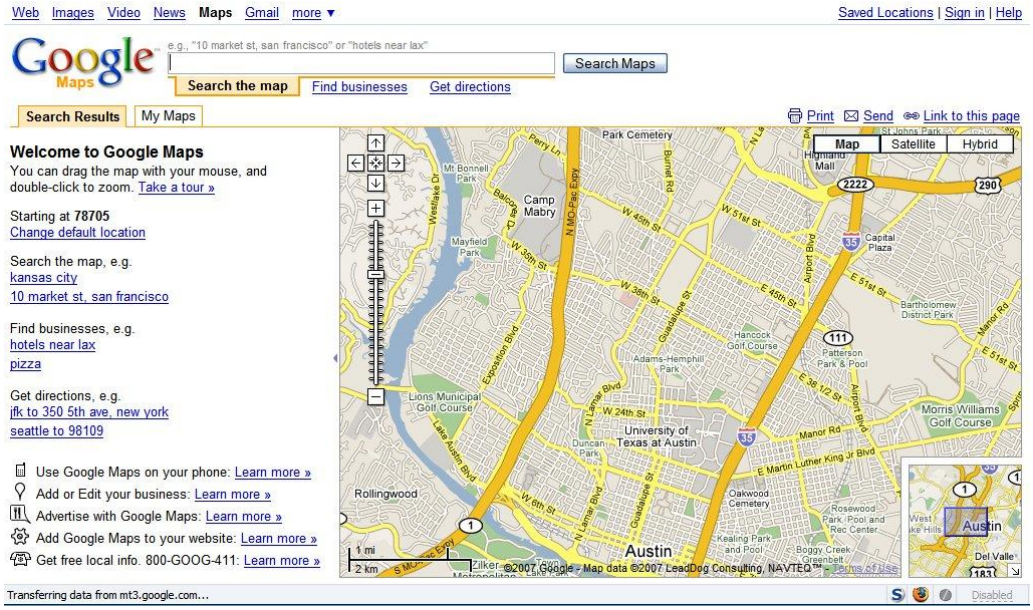

Figure 4.1: Google Maps Application Example

4.1. User Experience

The following paragraphs describe a typical user experience on the rideshare website, though the exact mechanics may change.

The driver or passenger logs in to the secure website. The user then proceeds to create a new rideshare, or revise a previous rideshare from his history tab. This entails setting the beginning and ending points, as well as designating a window of time for when he plans to leave. He may also designate the rideshare as public or private, indicating if matches may be found only within his "friends" list, or within the general user base. He may set preferences for how far out of the way he is willing to pick people up, as well as for the gender or age of his counterparts.

The user experience will now split into two cases. In the case of a rideshare which will occur on short notice, the user will receive either an immediate match or a message that no match has been found, given the time-frame requested. He can then either alter his rideshare information and search for a new match, or leave a "contact directly" message, so that potential riders can phone or text him to coordinate a pick-up.

In the case of a ride with relatively long notice, the user will receive a confirmation message and email for his request. He will then need to wait for an email indicating a match, or check back with the website for matches. When a passenger is added to the driver's route or vice versa, an email or text message will be sent to both parties, and each user can go to the site and review their counterparts. If the user deems the match acceptable, he or she can accept the rideshare or reject it and move on to a new search. This sequence of events is illustrated in Figure 4.2.

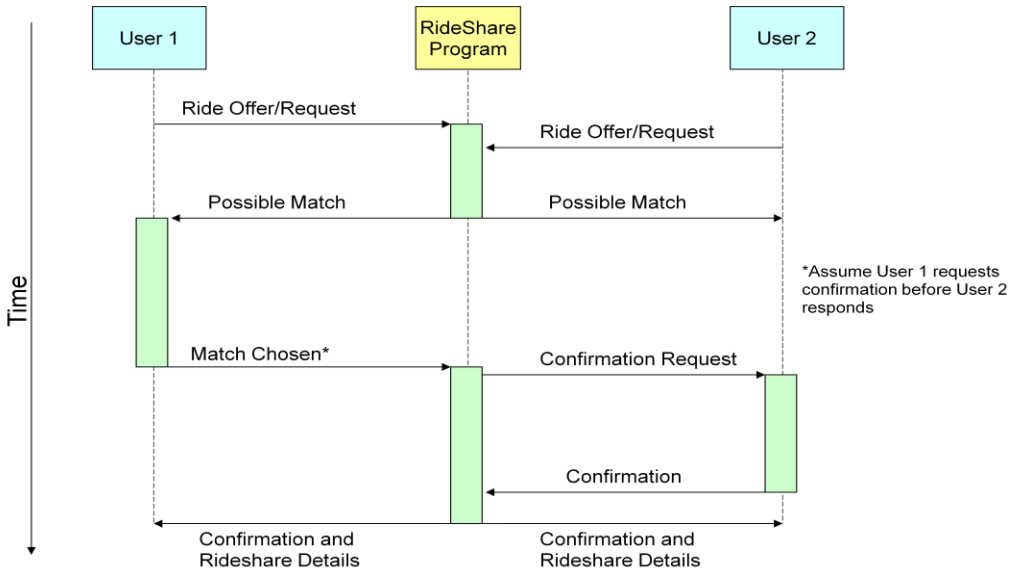

Figure 4.2: Rideshare Sequencing Diagram

Once a rideshare has been accepted by both parties, a ride confirmation message will be sent to them. This will include all contact information, along with an annotated map of the rideshare, indicating the route and times for all passengers. If a user cancels a rideshare, the website will request they specify a reason to be sent to their counterparts. After a certain number of cancellations or negative reviews, a user's profile may be locked for adding new rideshares until it has been reviewed by a site administrator.

4.2. Options

Users may also manage their profile and review or alter different options. These options may include the following: a user's personal information, such as name, address, gender, and age; general preferences, such as the look and feel of the website, starting page, and communication methods; and rideshare preferences, such as gender, age, and driver ratings. The user may also take advantage of other rideshare tools to alter pending rideshares or set ratings/comments for previous rideshares.

5. Key Operational Specifications

In order for our program to ultimately work, each of its pieces must perform to a certain standard. To begin with, our user interface must be secure. Each user's login information and preferences must be saved, along with their active rideshares. We can test this using human beta testing as well as attempting to rudimentarily hack user accounts, changing user settings around and verifying that they have been saved.

Additionally, our application server must measure up to several standards. The database it contains must be secure, in that no one outside our program can query it for data, which we can quickly test by setting a password and trying to bypass it. The server's hardware must also perform well; we must be able to add and drop entries from the table quickly and to query data quickly. The server overall must have a large data capacity in terms of hard drive and random access memory. In addition, it must be connected to a high bandwidth internet connection.

Our algorithm must be efficient. This implies that it should generate a maximal solution in terms of the number of users served and the number of passengers per car, while providing those users with the shortest possible routes. We must be sure that our algorithm can intelligently compare routes, mostly in terms of length and time, doing all this in a reasonable amount of time.

Finally, one of the most important functions of our website is its user interface. The interface must be user-friendly and intuitive, which we can measure by timing how long people spend on each page and where they get stuck. Tester feedback will be essential for improving the user interface, as they can

inform us exactly where the website's mechanics aren't working, as well as qualitative things such as if the user did not like the color scheme, or why they chose to leave the website.

6. Key Environmental Specifications

On the hardware level, the environmental requirements for our ride sharing program are the same as any other computer: electricity and a cool, dry room to operate in, as well as a connection to the internet. As this is a software-based project, it will have to stand up to some electronic rigors. Our program will need to adhere to some capacity limitations, probably measured in total number of users and routes saved in our database. Because of hard drive space constraints, our database will only be able to hold so many such users and routes without needing to add extra capacity.

Also, the server can only handle some finite number of requests in a short amount of time without causing long waits, because the server's processing power and communication capacity are bounded.

7. System Design

In order to provide the functionality described in the previous sections, the Rideshare application includes several application-level modules. Each of these modules in turn includes sub-modules and utilities which implement lower-level functionalities.

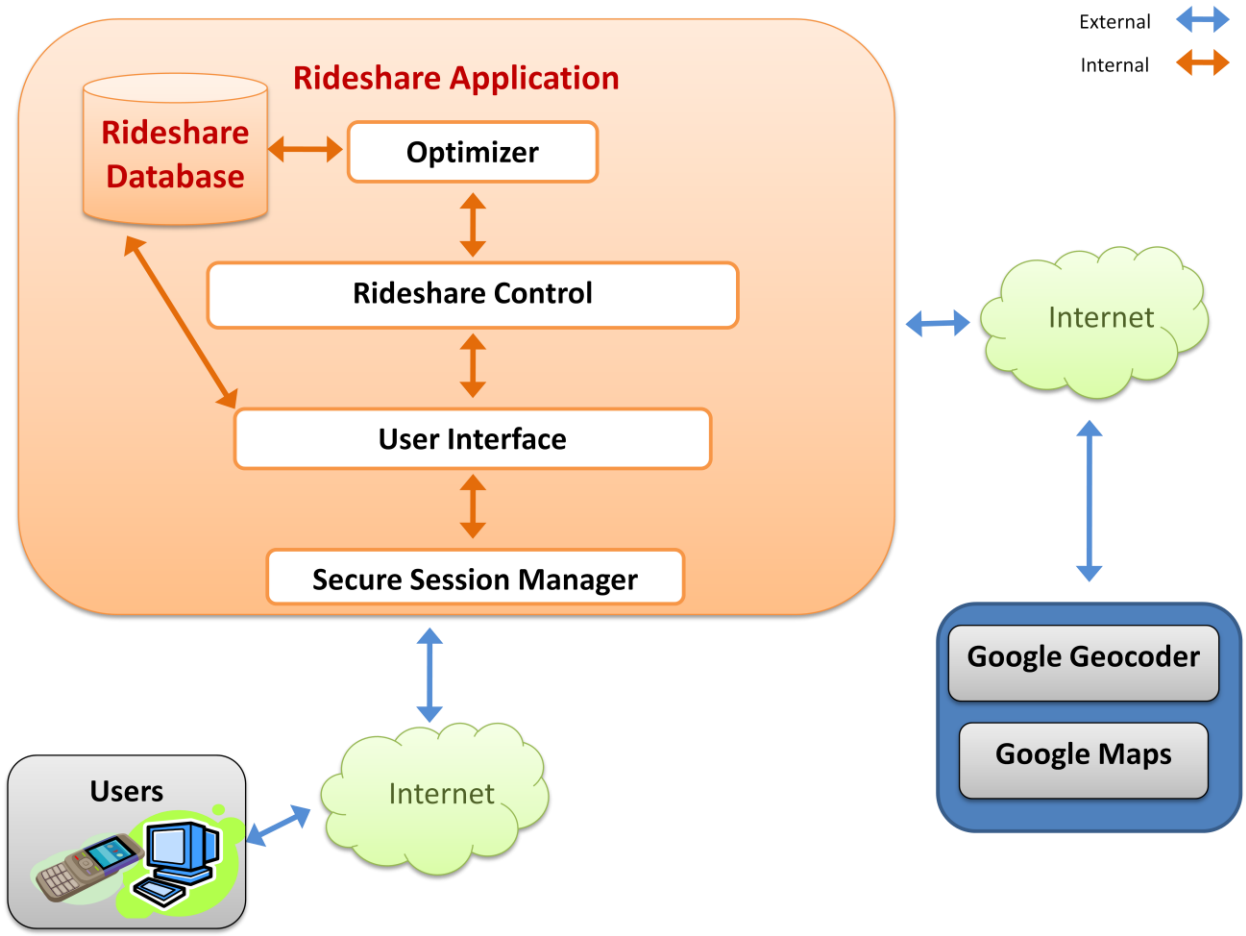

Figure 7.1: Top-level software architecture

Figure 7.1 describes the top level view of the software architecture. Note the differences between the arrow heads and arrow colors. Information which flows locally, i.e. between modules of our application, and provides regular functionality will be marked with orange arrows. Information originating or terminating in external sources is marked with blue arrows.

For ease of viewing the application architecture, we have omitted the “internet cloud” from the following diagrams, though its presence is implied.

The main modules in our application, marked with orange outlines throughout the rest of this section, will be rideshare control, user interface, secure session manager, optimizer, and testing suite. These modules provide the core functionalities of the Rideshare Application, and coordinate all local and external communications.

Each module utilizes information supplied from local (other application-level modules) or external (users, third-party) sources. They also utilize an application-level database, denoted rideshare database, to store and retrieve information. Application-level modules will access only those tables relevant to them, as we will show in the module-level designs.
Descriptions of each module, sub-module, external source, and database tables will be provided in the following sub-sections.

7.1. Rideshare Control

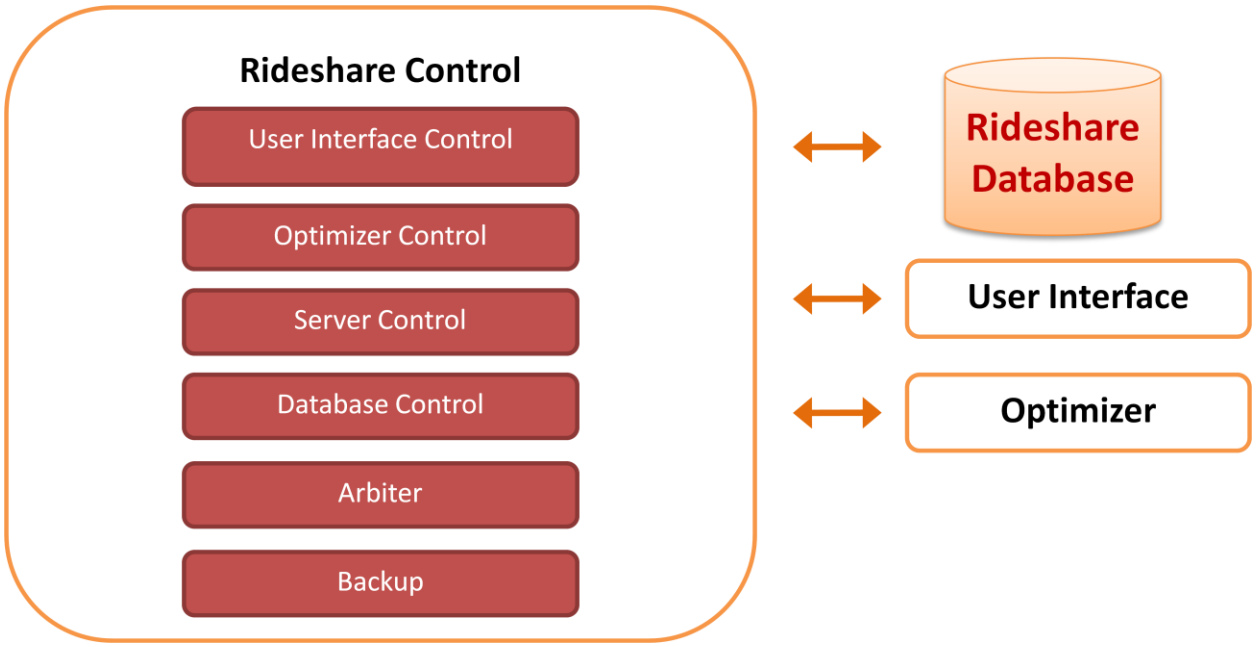

Figure 7.1: Rideshare Control Module

The rideshare control module (Figure 7.1) can be viewed as the "conductor" of the rideshare application. It is in charge of running and maintaining all of the other application-level modules. The following sub-modules comprise rideshare control.

7.1.1. User Interface Control

This control sub-module will initialize and allocate resources for proper functionality of the user interface. The user interface controller will be the program which “calls”, or in other words runs, the user interface.

Any maintenance required for the user interface will be provided by the user interface control. For example, if the user interface has surpassed its capacity, the control module will allocate more resources, or, if necessary, terminate the user interface module.

7.1.2. Optimizer Control

The optimizer controller “calls” the optimizer module when matching results are needed. It relates to the optimizer which sets of ride requests are to be optimized, and what time priority they have. This queuing method ensures that results are generated in a timely manner, depending on need.

The controller will also initialize and maintain any resources necessary for the proper operation of the optimizer, similarly to what was described in the user interface controller.

7.1.3. Server Control

This sub-module will perform all initialization and maintenance routines for the rideshare application. Any server-level interactions, besides normal internet communications, will be handled through the server controller.

For example, if the user interface and the optimizer both require additional resources, the server controller will be the arbiter, and reallocate the server resources according to some policy.

7.1.4. Database Control

This controller will initialize and maintain the database. It will also set restrictions on database access at the application-level.

7.1.5. Arbiter

There are potential cases of “critical sections” between the modules. This is a case when one module alters data which is being used by another module. The second module will subsequently be using “stale”, or old, data.

An example of this can be seen in the case of a user changing their ride-request details while the optimizer is running the corresponding request. The arbiter will inform the optimizer of the critical section, and in some cases allow it to include the “fresh”, or new, data. Wherever similar situations arise, the arbiter will perform similar duties.

7.1.6. Backup Utility

This sub-module will backup all rideshare application information, including the database, log files, statistical data, and server log files.

Source-code files, i.e. our software development project, will be backed up separately.

7.2. User Interface

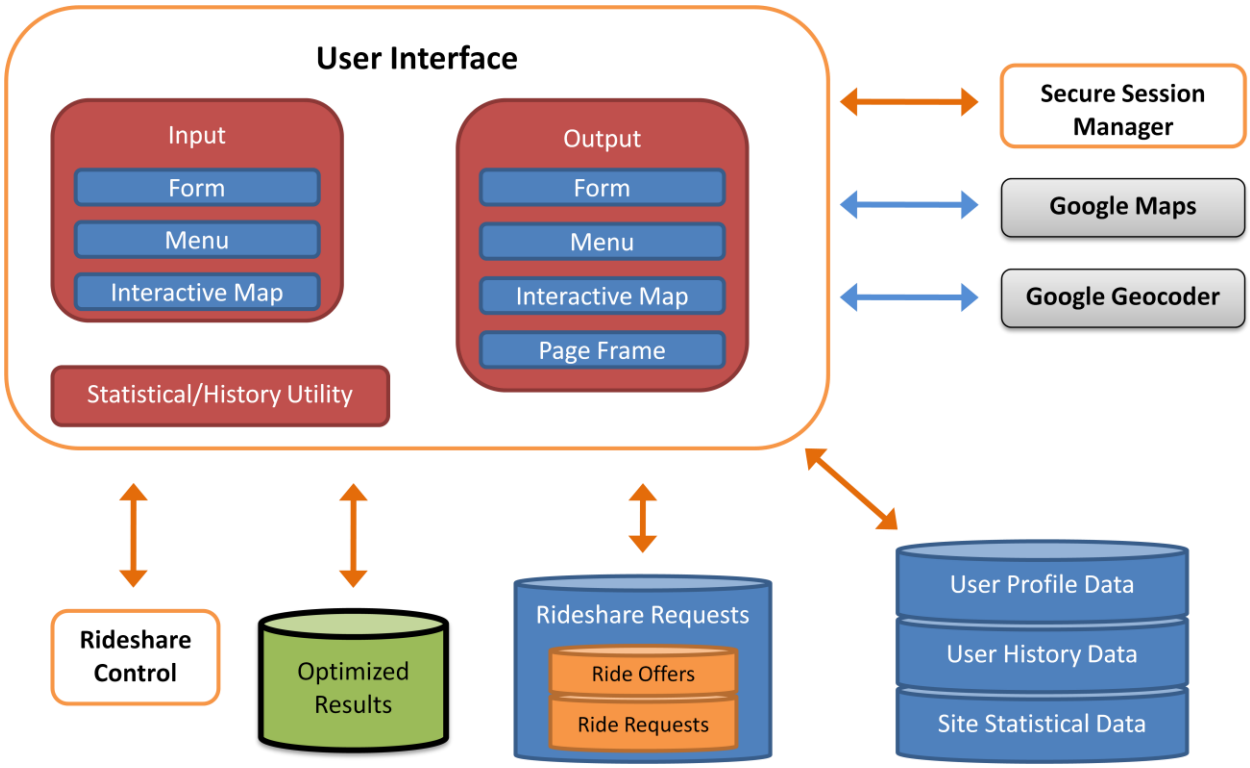

Figure 7.2: User Interface Module

The user interface control module can be viewed as the "agent" connecting the rideshare application and the users. It runs separate user interface sessions for each unique user, interacting with the user through a set of menus, forms, and maps. All information exchange and communication with the internet browser (i.e. Firefox, Safari) will be sent to the user interface via the secure session manager. Output and data storage tasks will be performed based on user input. The following sub-modules comprise user interface.

7.2.1. Input

- **Form**

All user form entries will be processed using a set of functions devoted to this task. This utility will also store form entries for each user. This will allow for automatically filling in fields on similar forms.

- **Interactive Map**

Any user map inputs will be captured by this utility. Support for all Google Maps interactions will be necessary for the proper functionality of the map interface.

The map interactions utility will store statistical data on these interactions, as well as user-specific data. This will allow for site administration and testing, as well as a user-friendly map, i.e. one which "remembers" where the user's usual locations are.

- **Menu Interactions**

Any user input to menus will be captured and processed by this utility. This includes all navigation and actions, i.e. links and clickable "buttons."

The map interactions utility will store statistical data on these interactions, as well as user-specific data. This will allow for site administration and testing. In addition, it will allow for a better user experience, since the utility "remembers" the user's favorite pages on the site, and thus saves the user repetitive navigation of the site to get to these pages.

7.2.2. Output

- **Form**

The form utility will generate the form according to the page-relevant form needed. If the user has previously filled similar fields, the utility will fill the fields for the user, according to the values which the user "prefers."

For example, if the user's history for the field "Name: _____" is "Garrett","Garrett", and "Yoni," then the field would read "Garrett" the next time the user sees it.

- **Menu**

The menu output utility will generate the relevant menus for the user, according to the page-relevant menu, as well as the user's "favorite" links.

- **Interactive Map**

All map outputs will be handled by this utility. This includes map overlays and menus. Map outputs will be handled by this utility using the standard Google Maps API.

- **Page Frame**

The actual website that the user sees will be handled by the page frame utility. This utility can be viewed as a set of blueprints for the site, with specifications for the placement of content. Each blueprint corresponds to a different set of page configurations which are presented to the user.

The form, menu, and interactive map utilities will output their content to the browser via the page frame utility.

The page frame utility will record statistical data for uses in site administration. It will also store user-specific information to regenerate user-specific "favorite" page-frames. For example, if a user were to usually open an "advanced options" version of the ride request form, then the user's default page-frame for that page would already have advanced options visible.

7.2.3. Statistical/History Utility

This utility provides the input and output sub-modules with the routines for storing and retrieving statistical and user-specific data. It will provide functionalities corresponding to the above-mentioned usability and administrative features.

7.3. Optimizer

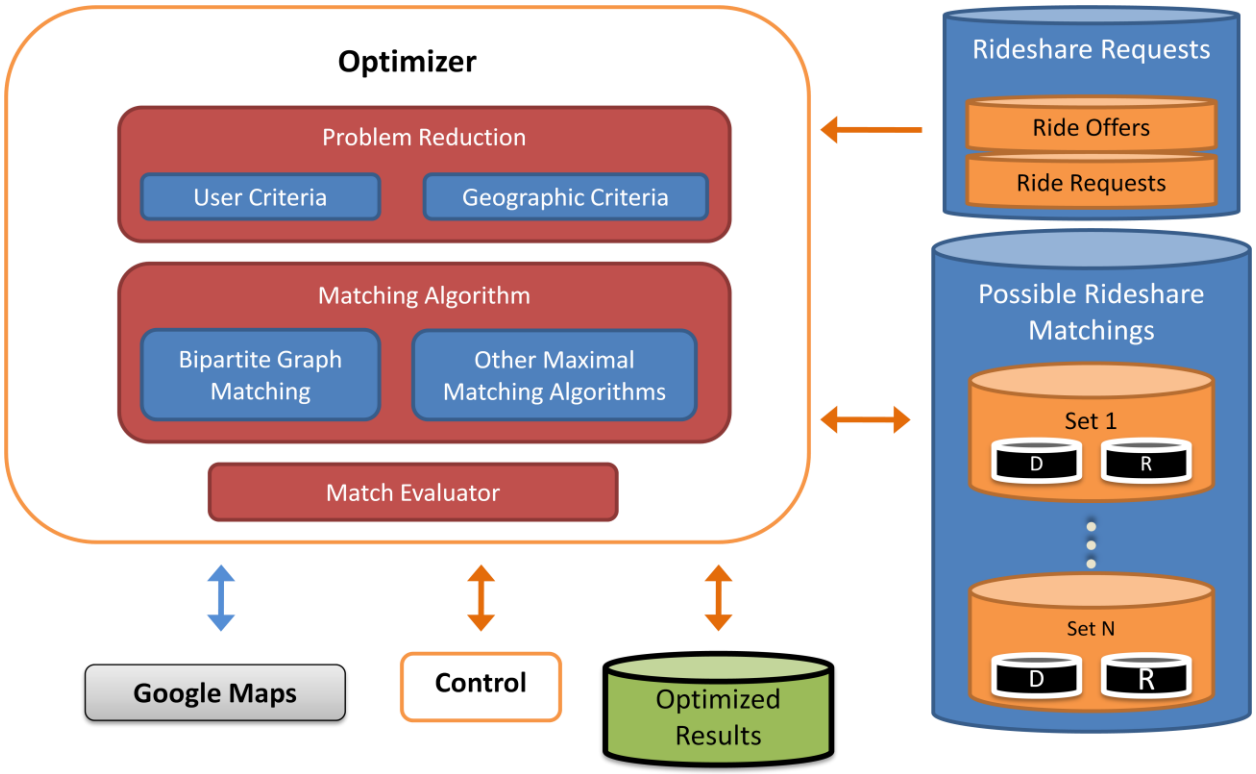

Figure 7.3: Optimizer Module

The optimizer module is responsible for generating rideshares which are both efficient and timely. It takes as inputs the sets of all ride offers and ride requests, and outputs the optimized results in the form of possible matchings between riders and drivers.

7.3.1. Theoretical overview of the optimization problem

Some understanding of elementary graph theory is necessary in order to solve our optimization problem. We include this section to assist the reader in understanding the optimization problem at hand, and thus gain an understanding of the optimizer module's required functionalities. Mathematical definitions are marked with bullets.

Definition of a graph [4]:

- A *graph* is defined as a set of *vertices* $V=\{v_1,v_2,\ldots,v_n\}$ and a set of *edges* $E=\{e_1,e_2,\ldots,e_m\}$, for some positive integers m and n.
- The vertices can be thought of as dots, and the edges as lines connecting those dots, as shown in Figure 7.3.1

Our optimization problem can be viewed as a graph theory problem. In our case, the drivers (D) and riders (R) can be viewed as vertices, where two vertices are connected by an edge if they can be partners in a rideshare.

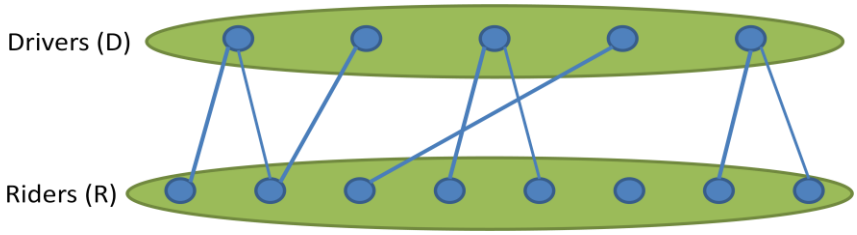

Figure 7.4: Bipartite graph of drivers and riders

- A *matching* is a set of vertices and edges. For a matching M, two vertices are said to be matched if M contains an edge which connects them.

Our optimization problem can be viewed as equivalent to the maximal matching problem. The maximal matching problem is the problem of finding a matching within a graph which includes the maximum number of vertices. Translated to rideshare, this means finding the most matches between drivers and riders, and thus providing rideshares for as many people as possible.

In particular, our optimization problem can be viewed as equivalent to a bipartite graph optimization problem.

- A *bipartite graph* is a graph in which the vertices of the graph can be divided into two disjoint sets, where no two vertices in the same set have an edge between them.

In our case, we have two sets of vertices, D (drivers) and R (riders), which can never have rideshares within the sets, i.e. no two drivers will be in a rideshare together, and the same holds for riders. This relationship can be seen in figure 7.3.1, where no edges are visible within the sets D and R.

7.3.2. Problem Reduction

In order to find a maximal matching, we first need to reduce the search space of the problem at hand. Search space reduction is the process of eliminating possibilities which are clearly invalid, and thus reducing the amount of work necessary for finding a matching.

- **User Criteria**

  In this utility, user preferences will be used to eliminate possible matchings. For example, if a user specifies that they will ride only with females, any matching which includes males will be eliminated from the search space. In graph theory terms, the user criteria search space reduction removes edges between the vertices of incompatible users.

  Similarly, users with conflicting time preferences or age preferences will not be considered for a rideshare with one another.

- **Geographic Criteria**

  As with user criteria-based problem reduction, it is possible to remove those solutions which are obviously incompatible on the basis of geographic distance. For example, if Jane is driving from New York City to New Jersey, and John needs a ride from downtown Austin to North Austin, then Jane and John are obviously incompatible for a rideshare.

The output of the problem reducer sub-module will be a set of smaller problems which can then be processed by the optimization algorithm. This will look like a collection of sets which need to be optimized. Each of these sets will represent one bipartite graph of drivers and riders who have passed the initial search space reduction.

7.3.3. Matching Algorithm

After we have reduced the search space in the problem reduction utility, we will need to employ an algorithm to find matchings for each of the sets generated. An algorithm is a set of consecutive logical operations which guarantee a certain result.

The result in our case is a maximal matching for a set of drivers and riders. For this task, we have found a maximal matching algorithm which is designed specifically for bipartite graphs.

We plan to use the algorithm specified in [4]. Specifically, we will use the algorithm titled “Bipartite Graphs.” This algorithm is proven to run in polynomial time, meaning it is possible to run it on a large amount of data, while completing the optimization in a reasonable amount of time. For a given algorithm, the time-complexity can be evaluated in big-O notation, where time-complexity refers to the amount of time necessary for the algorithm to solve a given

problem. In our case, we have a problem of size |{D,R}|, i.e. the number of drivers and the number of riders. The algorithm laid out in [4] has been proven to run in O(D$*\sqrt{R}$). This means that as the number of drivers and riders increases, we can expect the amount of time it takes for the algorithm to run will be on the order of D$*\sqrt{R}$.

This is good news, since it implies that our optimizer will be able to run in an amount of time which is reasonable by human measures. If, for example, the amount of time necessary was on the order of O(2^{D*R}), also called exponential time, the algorithm might take longer to run than the existence of the universe.

We may also employ other optimization methods as we continue to develop our specific matching algorithm.

7.4. Secure Session Manager

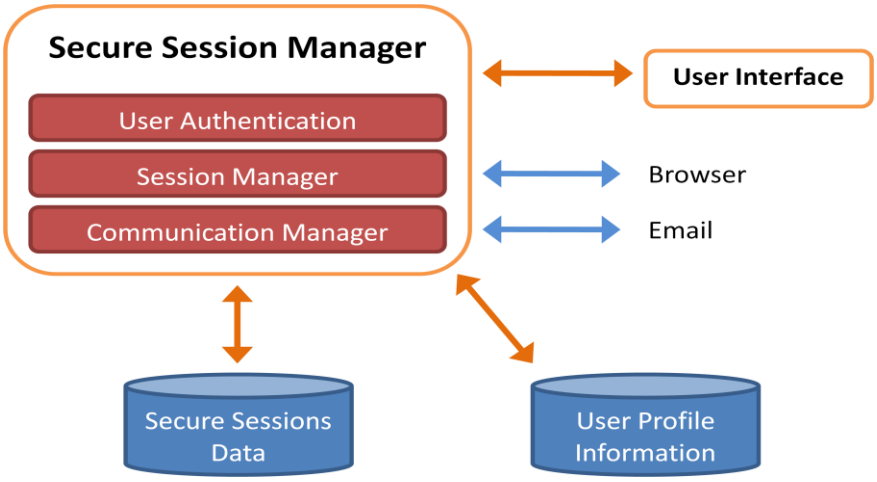

Figure 7.5: Secure Session Manager

In order to ensure the privacy of our clients, all communications between the application and users will be performed via a secure session. The secure session manager, Figure 7.5, will provide secure session services to all other modules requiring outside data or interaction.

It will do so by creating secure sessions for each module, and possibly many sessions for a single module. For example, the user interface will require a secure session to be created and managed for each user who is interacting with the application.

7.5. Rideshare Database

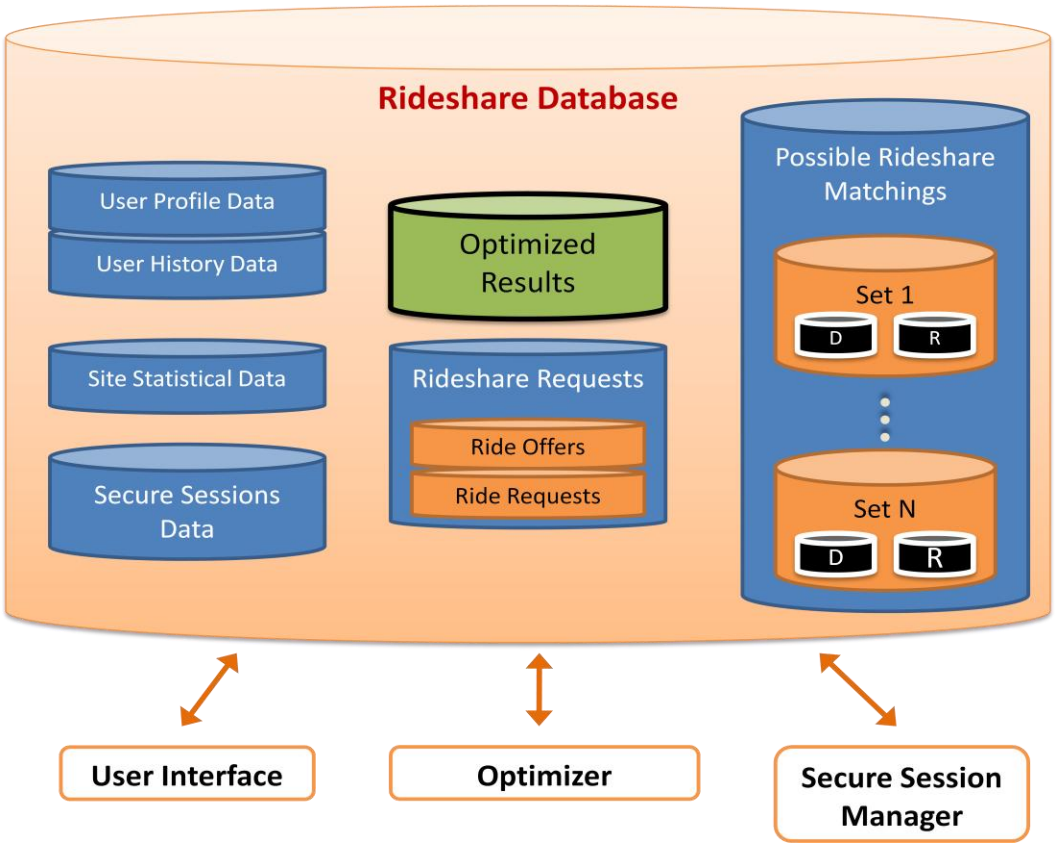

Figure 7.6: Optimizer Module

Each of the above mentioned modules has had one or more databases which it accesses, and uses to store and retrieve data. A database is a set of tables, which contain fields. The tables and fields are related to one another in various logical ways. Entries can be stored in the tables as rows, with each row containing that entry's field data.

The rideshare database, Figure 7.6, will contain all of these databases. Each cylinder in the figure represents a set of tables, which in turn contain a set of fields, and a set of entries. We will utilize a MySQL database to create and maintain the rideshare database.

The specific tables for each database will be determined during the implementation phase of the project.

8. User Interface Design

The quality of the user interface for our application is crucial, since the usefulness of our program will be directly correlated with the size of our user-base. The easier our website is to use, the more people will use it. The login, rideshare creation, and confirmation pages are outlined below.

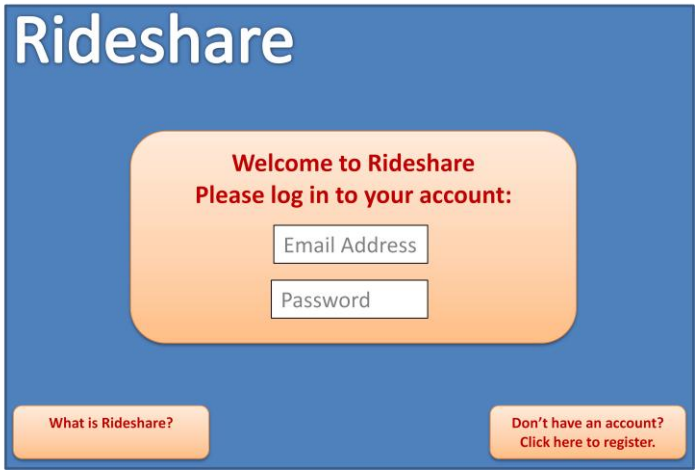

Figure 8.1: Login Screen

The login screen is fairly basic, but it illustrates an important point. Every user will be required to log in, entering the website through the secure session manager. If the user does not already have an account, they will be able to register through the link on the bottom right. Beyond this, the top of this page will have a logo and attractive heading for the website. The bottom left will have a link to an unsecured *About* page, where the user can read basic information about what the website does.

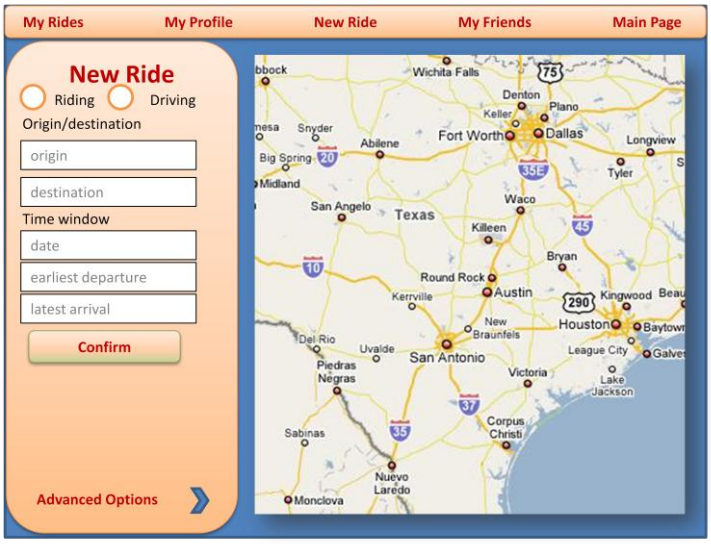

Figure 8.2: The *New Ride* Page

On this page, the user can input whether they are a driver or a rider, which will open the relevant options, as shown by figures 8.3 and 8.4 below.

The user will then input their route, in terms of where their origin and destination in the form fields, or by using the map interface. They will also be required to enter a time window for their trip.

The time window must be specified at least partially, but the user does not necessarily need to enter both. For example, if the user specifies only the departure time, then the application will be able to use the estimated drive time to specify the arrival time.

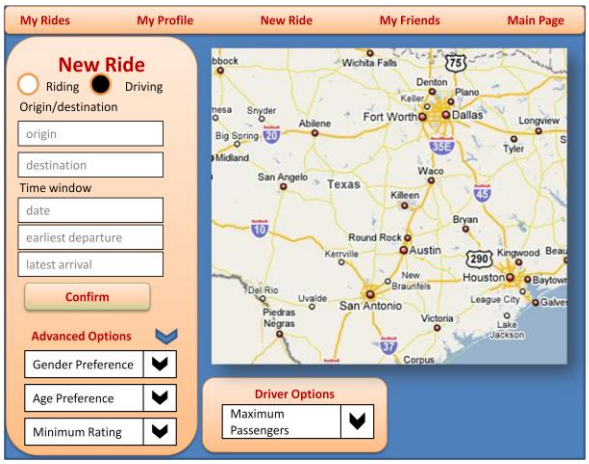

Figure 8.3: Driver's New Ride

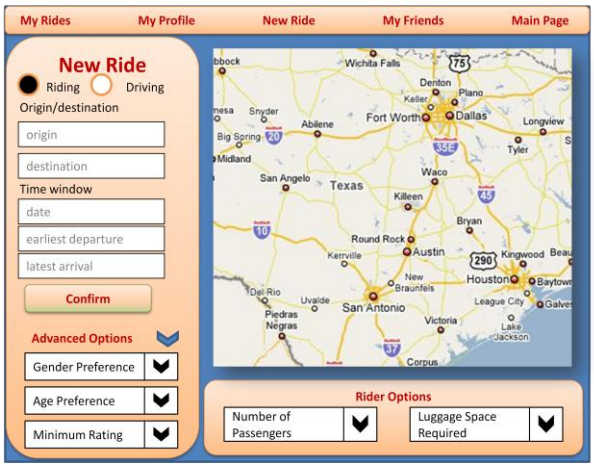

Figure 8.4: Rider's New Ride

Each user must enter whether they are a rider or a driver so that the program can enter these role-specific options. The algorithm needs to know how many passengers a driver can accommodate, as well as how many passengers and luggage the rider will be including.

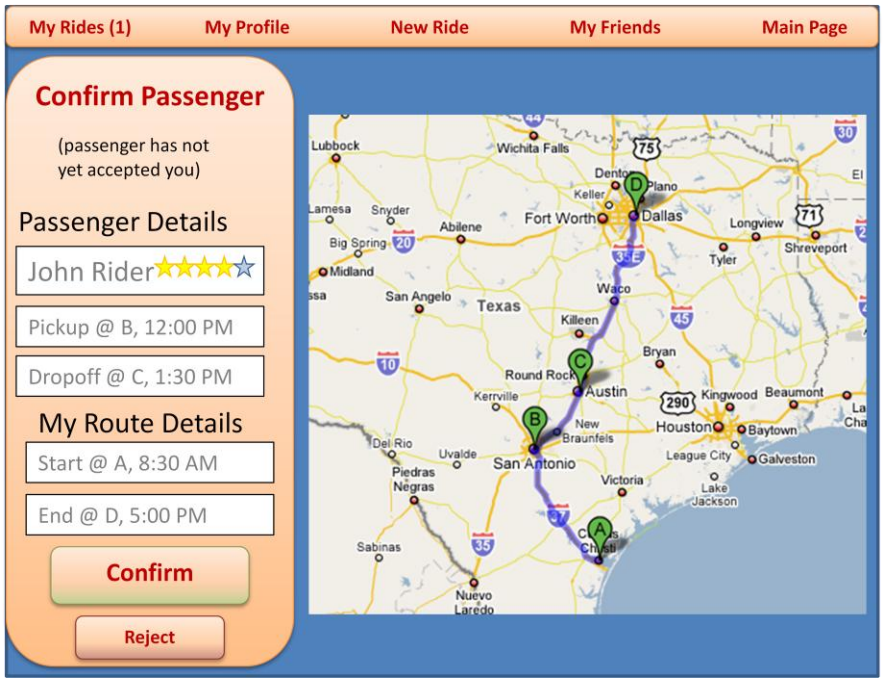

Figure 8.5: Passenger Confirmation Page

Figure 9.5 shows how a passenger confirmation page will look. This page allows a user to view a potential passenger, their details, and how the combined route would look. The passenger's name (in this case "John Rider" will be a link to their profile, where the driver can view their details. The stars by the name represent the user's rating, like eBay. In this case the user has a rather good rating of 4 out of 5.

The example page indicates that the passenger has not yet accepted the driver, which is indicative of the double-handshake process in which both the passenger and the driver have to approve each other before the rideshare can take place. If the passenger had accepted the driver before he saw this page, it would say so. A similar page will be created for passengers approving drivers.

Notice that each of these pages has the home bar at the top. Once logged in, every page the user sees will have this bar visible at the top. Through it, the user can navigate through the website and view the following pages:

- My Rides, which will display the status of the rides the user has created or been added to.
- My Profile, where the user can change their details and set personal options.
- New Ride, which would take the user to the new rideshare page, Figure 8.2.
- My Friends, which will let the user easily communicate and set up rideshares with people they know.
- Main Page, which will take the user back to the front page, where they can log out or read the website's current news.

Once sufficient user behavior information has been gathered, these links will change to accommodate for a faster and more user-friendly experience.

Additionally, when a users are matched to a new rideshare, they will be able to set an option to receive notification through the site, email or even text message.

9. Prototype Cost

We will likely need to purchase several tools to put our first run program together. Initially, we would need an internet-enabled server computer to run our programs, as well as crunch all the numbers. We could rent a server from several different companies for somewhere around $250 per month [5], or at least a virtualized portion of one server on which we could run just what we needed to. Virtualization allows multiple people to have access to the same computer as if they were separate physical machines. Alternately, the University could furnish us with a virtual server for essentially no cost to us.

The server also requires a few programs to run our project. Tomcat, a program which runs a Java web server, is freely downloadable on Apache's website [6]. Additionally, we will require support for the structured query language, or SQL, database program MySQL [7], which is free. The database program will allow us easy storage and retrieval of hundreds of thousands of data entries, which is necessary to support a large number of users.

We may also need to acquire a Software Development Environment (SDE), such as Microsoft Visual Studio, which could cost us as much as $300 [8]. This compiler and development environment will allow us to actually put together and publish our code, and it would be very useful in debugging. Alternately, we can likely use University resources such that this cost would not be incurred, or use an open-source SDE such as Eclipse.

To communicate with Google Maps, we will need to use their application programming interface, or API. To obtain this, we will use an API key provided by Google, which is free so long as we do not charge for our service [9]. However, if we do charge for our RideShare program, we will have to purchase an Enterprise API Key from Google for $10,000.

All totaled, our costs could be over ten thousand dollars, or free, depending on the university.

10. Project Cost and Schedule

Since we are implementing a software project, the product itself doesn't cost anything; it's the tools we require to create it. For implementing the rideshare application, we will use free software. To continually run the site, we will need a server, a broadband internet connection, and electricity to keep the server running.

The schedule, however, requires more consideration. The first week we come back to school, we'll meet up with Patrick, who will be our third member for EE464 – Senior Design. We will review the schedule set forth on the Gantt chart in Appendix C and make changes as needed. The next week and a half or so will be for setting up the server, in terms of actually physically obtaining it and then setting up the various programs on it. After that, we'll begin the first step of the actual website and the secure session manager.

The next week will be devoted to setting forth naming and version control conventions to ensure a well-organized development environment. We will also explicitly define how we will interface the modules, i.e. the rules by which they will interact with one another. This will determine which functions each module can call from the other functions. In addition, we will design the database tables required for implementing the functionality outlined in section 7.

This next week will also involve verification of our server and related software. We hope to iron out any errors from the initial bring up.

While Garrett and Patrick are working on the database, Yoni can start working on the optimization algorithm. After that, we can work on communication with Google as well as basic control and matching. Once this is done, we have our first real milestone: all the basic functionality will be in place, so we can test the program as a whole.

The rest of semester we will work on the advanced features, as well as refinement or debugging of features we have already implemented. The last few weeks, we can work on documentation and aesthetics.

11. Test Plan

In order to develop a stable and reliable application which fulfills the set of requirements outlined in previous sections, we will need a comprehensive testing plan. The testing utility will need to comply with all of the requirements mentioned in the needs analysis section of this document.

The implementation of each module will include a set of functions specifically designed for testing that module. This method of testing is called unit-testing. These functions should include the following test functions:

- Functionality
  Each module will include methods for evaluating the inputs and outputs of that module and assessing their correctness. It will likely prove difficult to do this for the optimization algorithm, so we will need to be prepared to spend more time creating and validating the test cases.
- Performance
  The testing suite should include methods for evaluating the timing behavior, i.e. how long it takes to complete, of the modules under various test cases, i.e. testing conditions.
- Stress
  Each module will include a method for creating a large number of requests relevant to that module. This will allow us to test for and approximate the maximum number of users, server-requests, etc. that our application can handle. It will also enable us to quantify the required resources for any given number of users and ride requests, in terms of the database size, internet bandwidth, and processor speed requirements.
- Security
  Evaluates server-level security and application level security. This allows us to establish a definitive security framework within which we are operating. We will design a security testing module which attempts to bypass each security feature in the application.
- Usability
  We can assess the usability of the application by having test-users enter and navigate the application, and comment on the menus, look and feel, options, and navigability. We can then alter or remove content based on these results.

The foremost important testing guideline is that we must design the software with testing in mind, also called “design-for-test.” For each module we will assign a set of test-cases relevant to that module’s functionality.

We will also employ a strict version control system, in which we will all be responsible for assigning test versions for each module, and subsequently communicating its readiness for launching the final product.

Once a stable version of the product is ready, we will begin to conduct alpha and beta testing. Alpha testing is simulated or operational testing by potential users or the development team, whereas beta testing, which succeeds alpha testing, is a release of the product to a limited audience [10].

12. Conclusion

We believe rideshare can help ameliorate the energy problem by facilitating cooperation between individuals, and in addition help to decrease their environmental impact. Our approach to facilitating rideshares is different from anything currently available. The core of our design approach is the user experience and effectiveness of results.

Each section of this document should be referenced at every step of the implementation process for evaluating the progress and tasks ahead. Software development methods must be considered on the basis of their adherence to these core requirements.

It is clear that the underlying optimization problem is a design challenge in itself, but we have found sufficient evidence of the solvability of this class of problems [4]. We have also employed a method of "divide-and-conquer," by first reducing the search space. We think this method will be the key to generating an effective and timely solution for our users.

The user interface will be minimal, and have map interaction as a core functionality. It will send and receive communications through a secure session management system, ensuring the privacy of our users and decreasing susceptibility to malevolent entities.

Design-for-test is at the core of our design process. Interfacing between software components must be well thought out and agreed upon before implementation can begin.

Most of the tools we require for completing the rideshare application are freely available. The server and internet connection are the only element for which there is not a freely available resource. If the Electrical and Computer Engineering department gives us access to a server, our expenses would likely decrease to zero.

At this point, we believe that our system design is likely to succeed in implementing rideshare and that our design-for-testability approach will ensure its success. We are confident that our collective abilities will be suitable to complete the project within the allotted time.

13. References

[1] "Use Case Diagram" [Online document], 2007, [cited 2007 Nov 6],
Available HTTP: http://en.wikipedia.org/wiki/Use_case_diagram

[2] "E-Rideshare" [Online document], 2005, [cited 2007 Nov 6],
Available HTTP: http://www.erideshare.com/

[3] "Google Transit Beta" [Online document], 2007, [cited 2007 Nov 6],
Available HTTP: http://www.google.com/transit

[4] Santosh Vempala, "Matching Algorithms", Combinatorial Optimization Lecture Notes, 07-Oct-2004,
Available HTTP: http://www.cs.dartmouth.edu/~ac/Teach/CS105-Winter05/Handouts/

[5] "eApps Hosting Plans," [Online document], 2007, [cited 2007 Nov 4],
Available HTTP: http://www.eapps.com/Docs/Documents.jsp

[6] "Apache Tomcat," [Online document], 2007, [cited 2007 Nov 5],
Available HTTP: http://tomcat.apache.org/

[7] "MySQL, The world's most popular open source database " [Online document], 2007,
[cited 2007 Nov 4], Available HTTP: http://www.mysql.com/

[8] "Visual Studio 2005 Standard Edition" [Online document], 2007, [cited 2007 Nov 4],
Available HTTP: http://msdn2.microsoft.com/en-us/vstudio/aa718672.aspx

[9] "Google Enterprise Website Search Solutions" [Online document], 2007, [cited 2007 Nov 3],
Available HTTP: http://www.google.com/enterprise/maps/faq.html

[10] Wikipedia contributors, "Software testing," *Wikipedia, The Free Encyclopedia*,
last revision: 6 December 2007,
Available HTTP: http://en.wikipedia.org/w/index.php?title=Software_testing&oldid=176241492

[11] The Access Board, "Web-based Intranet and Internet Information and Applications (1194.22)"
[Online document], 2001, June 21, [cited 2007 Dec 5], Available HTTP: http://www.access-board.gov/sec508/guide/1194.22.htm

Appendix A: Relevant Standards

The internet is a largely unregulated means of communication; as long people can find hosting, they can post whatever they want. However, there are some voluntary regulations that we intend to follow. For instance, Section 508 of the Rehabilitation Act set forth by Congress is mandatory only for government websites, but is a good guideline for any internet developer [11].

Section 508 defines a minimum accessibility level for websites, making sure the hearing, motor control and sight impaired can still use the site. One of the suggestions mentions that any navigation or control scheme should not rely only on color, but should at least have some sort of textual clue as to what a button does. Colorblind users cannot tell whether a button is the start button unless it is labeled, whereas most people might be able to guess if the button was green.

Each of the suggestions in Section 508 are not mandatory for us, but would help include all audiences, also allowing our website reach a larger audience.

Appendix B: Bill of Materials

1) Server; One; Modern PC performance, 100GB+ hard drive capacity
2) Internet Connection; One; High-speed internet connection, especially upload speed
3) Electricity , and a place to put the server; somewhere the server can provide uninterrupted service
4) Instant Rails Content Management System; One; almost all of the software we need to put the application together, including:
    a. Apache; the main server program that coordinates everything
    b. MySQL; the database program
    c. Ruby Software Development Kit; and as much documentation as we can find
    d. Eclipse Software Development Environment; One; helps us code the program
    e. Google Maps API Key; One; lets us generate content from Google

Appendix C: Gantt Chart

ID	Task Name	Owner	Duration (Weeks)
1	Touch Base & Plan	All	1
2	Server Bringup	All	1.5
2.1	Physical Setup	All	0.5
2.2	CMS & Framework	All	0.5
2.3	Database Bringup	All	0.5
3	Secure Session Manage	All	1
4	Interfacing Definitions	All	1
5	Database Refinement	Garrett & Patrick	1
6	Optimization Interface	Yoni	3
7	Google Communication	Garrett & Patrick	2
8	Basic Control	Garrett	1
9	Basic Matching	Yoni & Patrick	2
9.1	Matching	Garrett	1
10	First Total Testing	All	2
10.1	...and Debugging	All	1.5
11	Advanced Control	Garrett	2
12	Advanced User Interface	Patrick	1
13	Advanced Matching	Yoni	3
13.1	Matching	Patrick	2
13.2	Matching	Garrett	1
14	Final Debug & Documentation	All	1
15	Design for Testability	All	All