Catch A Ride!

An ISU Carpooling Service

Ian Pierce
Monica Kozbial
Kyle Williams
Sarah Files

Table of Contents

Introduction

Project Definition

To put it simply - A carpooling service targeted at university students which connects drivers and riders to share the cost of driving between school and their hometowns.

With limited parking on campus and the added cost of owning a car, it becomes impractical for on-campus students, typically freshmen and sophomores, to bring cars to school. This becomes a problem when a student needs to go home for a school break or maybe just for a weekend. Fortunately, chances are that another student who has a car is already planning on traveling to the same place at the same time. This is where our carpooling web app comes in handy.

For those looking for a ride, the Iowa State Carpooling Service would provide a search service to find drivers who are making a similar trip. For drivers this service would allow them to connect them with eligible riders with the goal of filling open seats in their car. The benefits for this service are two fold - riders are able to find an affordable and convenient means of travel and drivers would be reimbursed for the expense of their trip.

Project Goals

The goal for the carpooling project is to create a web application where potential riders can search for trips and drivers can create trips. The carpooling service would allow users to use their Iowa State accounts to login. Along with obtaining the user's student information for easy registration, this would instill credibility between users by limiting the web app to only ISU students.

Riders can use the search interface to look for available trips, see ratings for the driver, the cost of the trip, the available seatings in the car, and contact information. On the flip side, drivers can create trips and add details to their profile such as their car make/model and driving preferences. After a trip has been taken both riders and drivers will be able to write reviews

about each other. The end goal is to have a fully functional website on multiple devices from desktop to mobile available to all Iowa State students. The next step would be to expand this service to other universities/colleges.

Project Design

Project Requirements

Functional Requirements

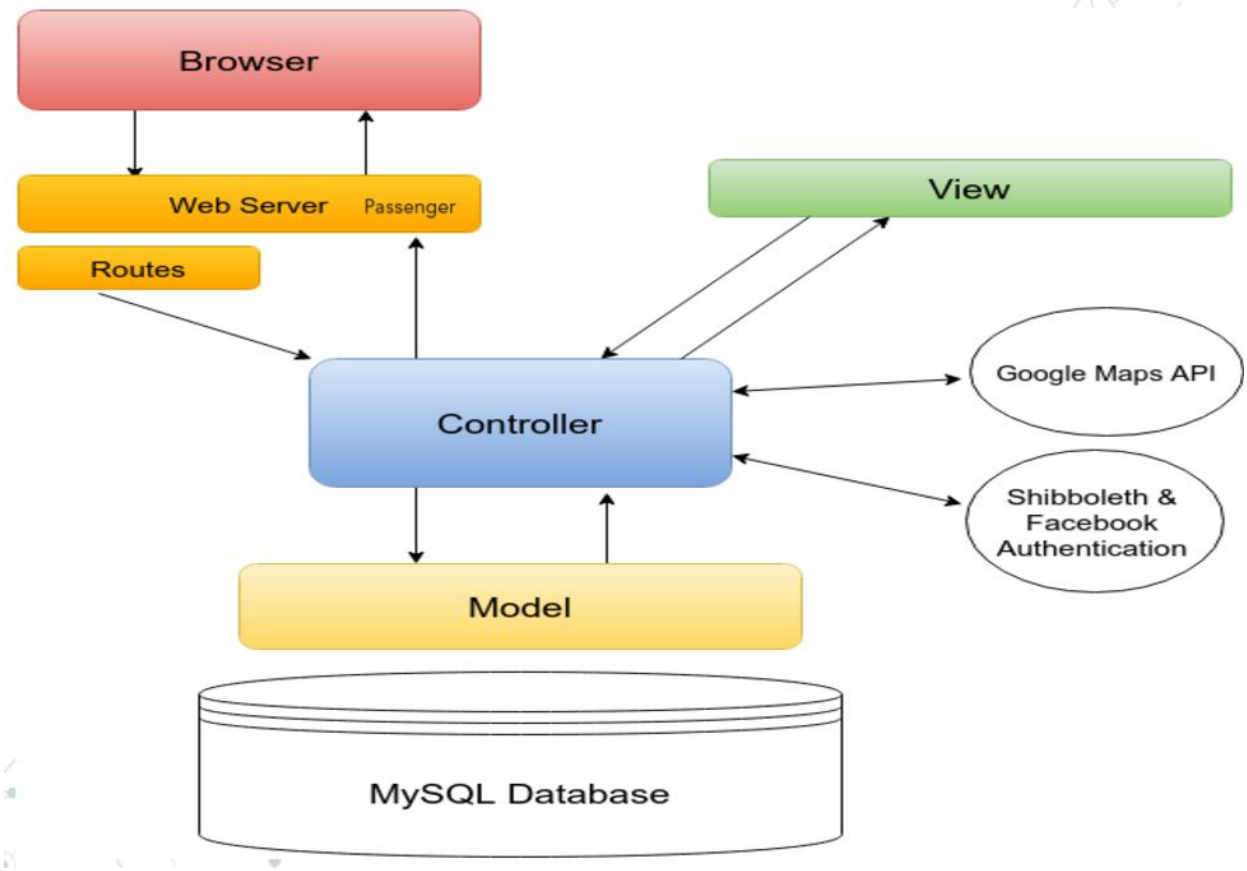
- The application will enable a user to create a trip posting
- The application will enable a user to request to join a trip
- The application will enable a user to accept passengers for a trip
- The application will display user information on a profile
- The application will display trip details on a listing
- The application will enable a user to search the database for trips using filters such as: proximity, date range, price, driver rating
- The application will allow a user to review a trip taken as rider or driver
    - riders review driver
    - driver reviews passengers
    - text component and rate thumbs up thumbs down
- The application will allow a user to login through facebook or shibboleth

Non-functional Requirements

- The application will handle a load of 1000 concurrent users
- All pages will render within 5 seconds
- The application can be used on mobile and desktop browsers
- All connections to the website will be encrypted using SSL

Architecture

The architecture of this project follows the standard Rails Model View Controller (MVC) architecture pattern with the addition of a multiple third party services including the Google Maps API and authentication through Shibboleth and Facebook.



Models are used to represent business data and logic. Results from the database is returned in the form of models. CRUD operations can be done on these models before persisting them to the database.

Views are how pages are presented through the use of HTML5, CSS, and Javascript. Rails makes views dynamic by allowing ruby code to be embedded in a view. This is used to display dynamic content on a page by reading instance variables from its controller.

A controller is essentially the middle man between the model and the view. It queries the database, retrieves model objects and passes them the view. A controller is a Ruby class that contains many ‘actions’ in the form of methods. As stated in the Routing section below, a controller’s action is called when the user navigates to a url. The controller runs the code in the action and renders the corresponding view to the user.

Implementation

Routing

Routing in a Rails application is how controller actions are mapped to a given url. For example, in our application the “/account” url is mapped to the ‘account’ action in the ‘users’ controller. When the user navigates to this url the controller retrieves the appropriate view which in this case is ‘/users/account.html’. For each route a HTTP request method is also specified (GET, POST, etc). Resources can also be used in routes. For example,‘/users/:id’ urls can be passed an ‘:id’ parameter which specifies which user id to be looked up. This project uses the default Rails routing rules.

Models

We use models to store and validate data. For example, the user model validates its email field to verify that it matches a certain regex before it can be persisted to the database. This is especially useful when creating new users. If a user tries to use an invalid email a corresponding error message will be displayed in the form. There are six core models - user, trip, ride, route, review, and vehicle.

user: Contains information about a user. Validates first name, last name, and email fields to confirm that they are in the correct format and length. Before it is persisted to the database, a random hash is generated for the permalink and api key fields. It also has methods to calculate certain attributes of the user such as the driver rating.

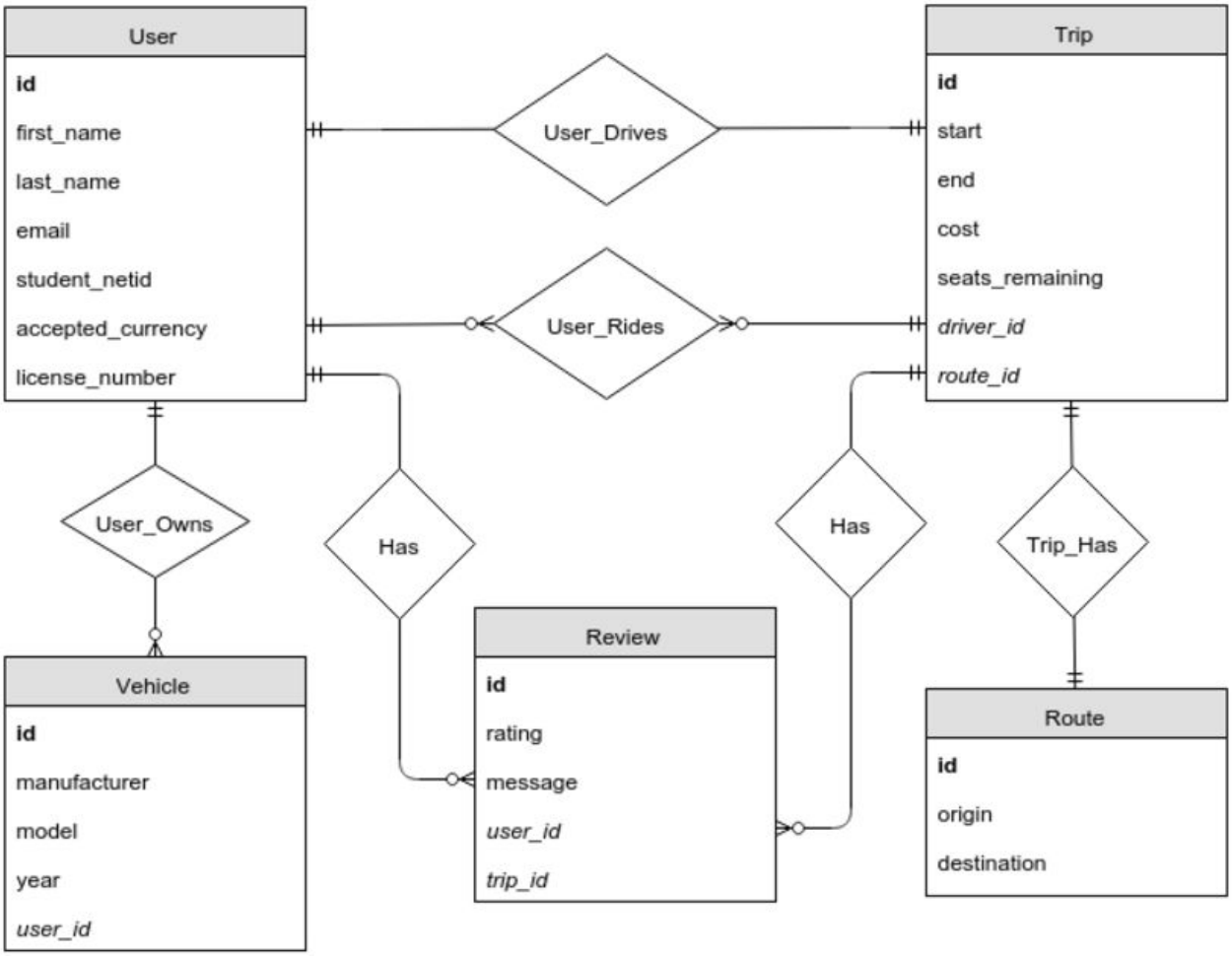
trip: Contains information about a trip including the driver of the trip (a user), route, cost, and vehicle. It validates this model by checking the presence of these attributes before persisting to the database. It has helper methods for adding riders and calculating the number of seats already taken.

ride: A relationship model between a trip and user. When a user requests a ride for a given trip a relationship entry is added to the rides table. This model also stores if the driver has accepted the user as a rider on his/her trip. It also enforces the the rule that a user can only be associated with a given trip once.

route: Each trip has a route. The route contains the origin and destination places of a trip.

review: Contains information of a review written by a user for another user regarding a certain trip. Riders can review a driver and vice versa. Fields in the model include a rating and message.

vehicle: Contains the manufacturer, model, color and year of a vehicle. A vehicle has a many to one association with a user and a one to one association with a trip.



Views

We use views to render page data. Controllers can pass instance variables to views to render dynamic data. For example, a controller could define an instance variable equal to a given user object. The account view could then use this object to display user information on their account page.

Views use the convention over configuration Rails principle to associate views with their controllers. Another example is when the ‘view’ action in the ‘trips’ controller is fired the ‘/trips/view.html’ view is rendered. This convention is standard for all of the views.

Controllers

We use controllers to get models from the database and supply them to the view. For example, if a user views a specific trip, the controller queries the database using the trip id in the url and supplies the model to the view.

Some controllers are used to send requests to third party APIs such as Google Maps. Others serve as API controllers and send back responses using JSON instead of a normal view (HTML)

Google Maps API

Since our application heavily uses GPS locations and routing we decided to use the Google Maps API. To query this API we send a GET request to Google with the appropriate parameters and our API key. A response is sent back in JSON format.

Embed API

The application uses embedded Google Maps on several pages. On the ‘create trip’ page an embed map is used to display the route while a user is filling out the form. An embed map is also used on the trip page to display the route of the given trip.

Distance Matrix API

The distance matrix API is used when searching for trips. When a user searches for a trip the distance between several pairs of cities needs to be calculated in order to filter results. Our application utilizes this API to calculate these distances.

Javascript API

The application uses the Javascript API to display location predictions when a user searches for a trip.

Authentication

This application users OmniAuth, available as a Ruby gem, to handle authentication for users. Omniauth is a library that standardizes multi-provider authentication and was used to enable and manage authentication for users through Shibboleth and Facebook. This strategy allows for easily adding more identity providers, if desired in the future, and was easily integrated into the application. In order to setup authentication, a few steps were needed including defining the identity providers and what type of information could be retrieved from them in an initializer file for omniauth. After this initializer file was created, a redirect was added to /auth/:provider (where “:provider” was either facebook or shibboleth) that redirected users to login with their credentials through the use of the OmniAuth library. After users logged in an authentication hash available

as an environment variable is generated (accessible through `request.env['omniauth.auth']`) from which attributes specified in the initializer file can be extracted and were used in pre-populating the sign-up form.

Facebook

The web application was registered as a Facebook application through Facebook and the application key and secret generated through this process was included in the initializer file for omniauth. Attribute specified for retrieval in the initializer file included name and email.

Shibboleth

The initializer file for shibboleth specified that the given name, surname, and email were attributes that could be retrieved.

Testing

Each developer had their own private and local Ruby on Rails server with which server to work. This allows each person to check any changes immediately, and test code acting as the front end user. This form of intrinsic part of the programming process permits constant testing.

Another aspect of the website application is security. The users are trusting the website with their information, and this requires the utmost care. Multiple online sources exist for testing SSL security of web pages. After a method of logging in was established, the process was tested many ways by available online resources.

Ruby on Rails has measures to make writing unit tests easy and not time consuming. Unit tests brought many bugs to the surface.

Appendix I - Operation Manual

Setup

The application uses a standard Rails setup. However, several third party services were integrated into the application.

Production Environment

There are several components to the production environment. We used a Virtual Machine (VM) which functions as our server and is the foundation of application. The database is hosted on this VM. The programming language, web framework, and dependencies are installed on this VM.

Red Hat Enterprise Linux 6 - The operating system of our VM. The VM has been allocated 8192mb of RAM and 20gb of hard disk storage.

MySQL 5.6.26 - The chosen DBMS of our application.

Ruby 2.2.1 - The latest version of Ruby is being used. Ruby and Rails were installed using RVM (Ruby Version Manager).

Rails 4.2.4 - the latest stable version available.

Extended SSL Certificate - Is necessary in order to use Shibboleth authentication

Phusion Passenger 5.0.21 - A web server and application server for Ruby

Apache 2.2.15

- loads modules for SSL (sll_module) and shibboleth (mod_shib)
- enable UseCanonicalName so that redirects to shibboleth are generated correctly
- with virtual host defined for port 80
    - redirects to SSL connection (to port 433)
- with virtual host defined for port 433
    - defines that it is a rails environment and that it is run in production mode
    - requires shibboleth authentication for path “/auth/shibboleth/callback”

Shibboleth

- requires ntpdate and ntpd to be running to keep system time closely synced with reality
- install shibboleth.repo configuration file appropriate to OS version
- register service provider on ISU AWS and install generated shibboleth configuration data

Demo

Viewing the website is easy! Go to https://sddec1509.ece.iastate.edu/

Appendix II - Alternative Design

The design phase for this project was greatly accelerated due to the one semester timeframe. This created a distinct challenge compared to other groups who had two full semesters to create a working product. Little time existed to experiment with alternate designs, therefore the first design was the one implemented and used. The application uses default Rails configuration and design.

Appendix III - Other considerations

What We Learned

All of us were new to Ruby and the Rails framework so initially it took time to get over this learning curve. The Google Maps API was also fairly new to us as well. Setting up authentication with Shibboleth was also a challenge.