

# RideShare

*Helping Connect People and Places*

## Objective

Do you ever wish it were easier to share rides? Have you ever been frustrated when you can't find a ride to get a place you really want to go? Well, there's RideShare.

RideShare is an open source web application that allows people from all around the world to share rides in an easy and effective way. Its simple-to-use interface and intuitive reservation system allows users to get to where they need to go without the frustration of finding a ride. It's the couch surfing of carpooling and is part of an initiative to create software that aims at reducing global pollutants that contribute to climate change.

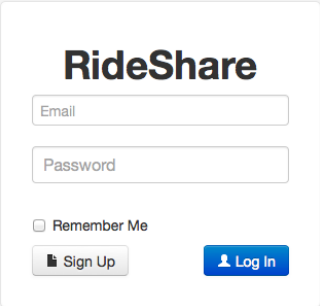
RideShare, inspired by Dr. Mark Pearson of Earlham College, started its journey as part of an independent study to help make it easier for college students to get to and from school. The premise of the project has vastly evolved as it now attempts to allow anyone to download the framework for free, install it on their servers, and open it up to a community of people who work together sharing resources for the common good. RideShare can be used in schools, companies, institutions, cities, states, and even countries as the interface is highly customizable for more specified purposes.

## About the User Interface (UI)

RideShare comes with a fully extensible UI based on Twitter's Bootstrap. It's equipped for easy customization and adaptation and provides many javascript tools such as pre validation of forms and searchable tables. The framework is completely cross platform compatible as it adapts well on any device it is rendered on.

The UI has many components built in to handle specific functions of RideShare on the user side of things. It has responsive tables, front end form validation, advanced ajax powered modals, and form format helper plugins.

The interface loads content by calling a URL and refreshing but keeps the tables in sync in case there are changes by using an AJAX call to a JSON file. In addition to this, forms status messages such as "Submitted Successfully" are received via AJAX.

A screenshot of the RideShare web application's login and sign-up interface. The form is white with a light gray border and is set against a light gray background. At the top, the word "RideShare" is displayed in a bold, black, sans-serif font. Below the title, there are two input fields: "Email" and "Password", both with light gray borders and placeholder text. Under the "Password" field, there is a checkbox labeled "Remember Me". At the bottom of the form, there are two buttons: a "Sign Up" button with a light gray background and a "Log In" button with a blue background and white text. The "Log In" button also features a small white user icon.

## Tour of RideShare's User Interface

You start off with RideShare with a login page. If you don't have an account, you can register for one and gain access to the driving community.

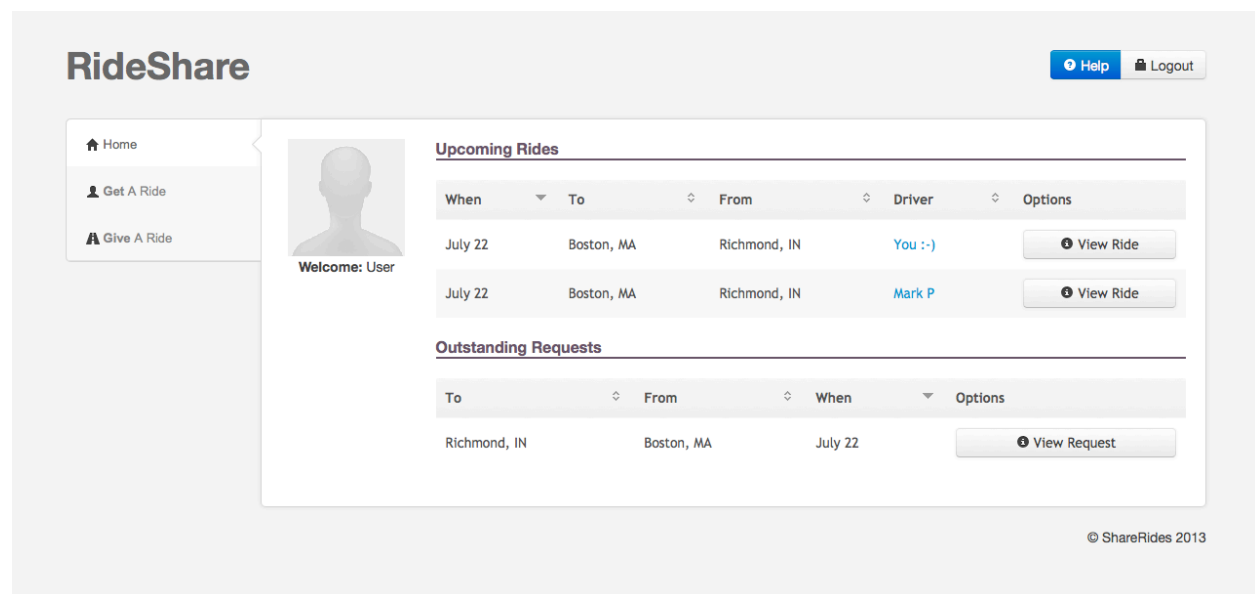
Once you are logged in, you will be directed to an application with three distinct tabs and a few small buttons. In the tabs, we have Home, Get A Ride, and Give A Ride. At the top of the application we have Help and Logout.

### Auxiliary Menu

There are two small buttons at the top of the screen; Help and Logout. If you need help with navigating and understanding how RideShare works, you can click on the help button. If at any point you want to log out, you can click the "Logout" button at the top of the screen.



### The Home Tab



The Home tab is where all the information involving anything relating to you are posted. You can view and edit trips that you are on, trips that you are driving, or trip requests that you made. All notifications involving RideShare will be displayed on this tab. You can also edit your user preferences by clicking on the "Preferences" button.

As a user, you have two distinct roles you can play in this application; you're looking to "Get a Ride" or you're looking to "Give a Ride".

## The Get a Ride Tab

The screenshot shows the 'Get a Ride' tab in the RideShare application. The interface includes a sidebar with navigation options: Home, Get A Ride (selected), and Give A Ride. The main content area features a 'Join Ride' button, a 'Request Ride' button, and a 'Refresh' button. A search bar labeled 'Search Rides' is positioned to the right. Below these controls is a table listing available rides. The table has columns for 'Select', 'When', 'To', 'From', 'Driver', 'Seats', and 'Options'. Five rides are listed, each with a radio button in the 'Select' column and a 'View Ride' button in the 'Options' column. The rides are: 1) April 19, Minneapolis, MN to Richmond, IN, driver Aaron T, 2 seats; 2) April 29, Orange City, IA to Richmond, IN, driver Emma W, 1 seat; 3) July 22, Boston, MA to Richmond, IN, driver Lelf D, 1 seat; 4) October 28, Richmond, IN to Memphis, TN, driver Mark P, 4 seats; 5) December 13, Richmond, IN to Los Angeles, CA, driver Mary C, 3 seats. A copyright notice '© ShareRides 2013' is visible at the bottom right of the interface.

Select	When	To	From	Driver	Seats	Options
<input type="radio"/>	April 19	Minneapolis, MN	Richmond, IN	Aaron T	2	<a href="#">View Ride</a>
<input type="radio"/>	April 29	Orange City, IA	Richmond, IN	Emma W	1	<a href="#">View Ride</a>
<input type="radio"/>	July 22	Boston, MA	Richmond, IN	Lelf D	1	<a href="#">View Ride</a>
<input type="radio"/>	October 28	Richmond, IN	Memphis, TN	Mark P	4	<a href="#">View Ride</a>
<input type="radio"/>	December 13	Richmond, IN	Los Angeles, CA	Mary C	3	<a href="#">View Ride</a>

If you are someone who is looking for a ride somewhere, the “Get a Ride” tab is where you want to be. You will be presented with a table that displays all of the available rides that drivers have posted. You can search and filter through rides on the table as well as view the full details of a particular trip.

If there is a ride you want to go on, select the item from the table and click on the “Join Ride” button to make a join request. Once the driver of the trip has approved, you will have the ability to see who else is on the trip and be able to chat with them with a wall like feature. You can take yourself off the ride at any point and can set up email reminders if you have a tendency to forget. The trip will instantly become available on your Home tab and trip page link will be emailed to you.

If you can’t find a trip you want to go on, you can request a ride by clicking the “Request Ride” button. This will bring you to a form that will allow you to fill out the details of your requested ride. Upon submitting the form, your request will become instantly available to drivers who might be able to add you on to a new or existing trip. Your request will also be available in the Home tab where you can delete it at any time. If a driver decides to take your request, you will get an email notification along with the ability to confirm your commitment. If a driver adds you to a trip, your role changes from a requestee to a passenger

## Give a Ride Tab

The screenshot shows the 'Give A Ride' tab in the RideShare application. The interface includes a sidebar with navigation options: Home, Get A Ride, and Give A Ride (selected). The main content area features a 'Drive' button, a '+ New Ride' button, and a 'Refresh' button. A search bar labeled 'Search Requests' is also present. Below these controls is a table displaying active ride requests. The table has columns for 'Select', 'Requestee', 'To', 'From', 'When', and 'Options'. Five requests are listed, each with a checkbox, the requestee's name, origin and destination cities, the date, and a 'View Request' button. The footer of the page indicates '© ShareRides 2013'.

Select	Requestee	To	From	When	Options
<input type="checkbox"/>	Aaron T	Minneapolis, MN	Richmond, IN	April 19	<a href="#">View Request</a>
<input type="checkbox"/>	Emma W	Orange City, IA	Richmond, IN	April 29	<a href="#">View Request</a>
<input type="checkbox"/>	Leif D	Boston, MA	Richmond, IN	July 22	<a href="#">View Request</a>
<input type="checkbox"/>	Mark P	Richmond, IN	Memphis, TN	October 28	<a href="#">View Request</a>
<input type="checkbox"/>	Mary C	Richmond, IN	Los Angeles, CA	December 13	<a href="#">View Request</a>

If you are someone with a car and willing to give people rides, the “Give a Ride” tab is where you want to be. You will be presented with a table that displays all the the active ride requests made by those who in the “Get a Ride” tab as a result of not finding a ride that matched their needs. You can search and filter through requests on the table as well as view the full details of a particular request.

If you are planning a trip and have a few extra seats, you can click on the “New Ride” button at the top of the tab container. This will bring you to the “Add a Ride Form” where you will be able to fill out all the details of your trip. You can also add passengers from active requests by selecting them in a list on the form.

If there is a ride request that fits well with your plans, you can select one or multiple requests listed on the table and proceed to click the “Drive” button. This will bring you to the “Add Ride Form” that will be partially filled with the details from the requestee(s). Once a requestee is added to your trip, they become a passenger.

To finish up, hit the submit button and the trip will instantly become available on your Home tab and a link to the trip page will be emailed to you and your passengers. You will be redirected to your “Trip Page” where you will be able to see the details of your journey, the passengers on your trip, respond to join requests, have discussions with the passengers using a wall like discussion board equipped with email features, and the trip control buttons. The trip control buttons allow you to edit the trip, remove passengers, and cancel the trip. Passengers will receive email notifications for any updates to the trip and can choose to receive email reminders. Your trip will also be available in the Home tab and you will have the ability to view it from there.

## How RideShare Works

RideShare's back end system is powered by a web application framework called CodeIgniter and is written in Hypertext Preprocessor (PHP). The framework is highly extensible, easily portable, blazing fast, and includes libraries of classes and helper functions that make coding web applications a breeze. CodeIgniter is capable of running on any web server system and can support most database solutions allowing an administrator to switch out hardware and software without infringing the integrity of the application.

CodeIgniter's architecture is based on concept called the Model-Viewer-Controller (MVC). This architecture allows developers to separate the application's data (the model), the display of that data (the view), and logic to bring everything together (the controller) The MVC architecture allows for amazing scalability, performance, and programability in web applications.

The Model represents the application's data structures and the functions it needs to create, retrieve, insert, and update information in its database.

### RideShare Database Tables

<b>users</b>	This table holds all user related data
<b>trips</b>	This table holds all trip related data
<b>requests</b>	This table holds all requests related data
<b>passengers</b>	This table holds passenger related data

### RideShare Database Schema

<b>users</b>	
user_id (unique num)	Unique identifier for the user
first_name (text)	User's first name
middle_name (text)	User's middle name
last_name (text)	User's last name
email_address (text)	User's email address
<b>trips</b>	

trip_id (unique num)	Unique identifier for the trip
to (text)	To destination. Format: city, state, country
to_lat (float)	Destination latitude
to_long (float)	Destination longitude
from (text)	From origin. Format: city, state, country
from_lat (float)	Origin latitude
from_long (float)	Origin longitude
date (date)	Date of journey
time (time)	Time of journey
driver_id (unique num) == user_id	This is the driver's unique identifier number which is equivalent to the user_id
seats (num)	Number of passenger seats available in the car
description (text)	Description of the trip
deleted (Bool)	Mark tuple for display/no display
requests	
request_id (unique num)	Unique identifier for the request
to (text)	To destination. Format: city, state, country
to_lat (float)	Destination latitude
to_long (float)	Destination longitude
from (text)	From origin. Format: city, state, country
from_lat (float)	Origin latitude
from_long (float)	Origin longitude
date (date)	Date of journey
time (time)	Time of journey
requestee_id (unique num) == user_id	This is the requestee's unique identifier number which is equivalent to the user_id

description (text)	Description of the request
deleted (Bool)	Mark tuple for display/no display
<b>Passengers Table</b>	
user_id (unique num)	Unique identifier for the user
trip_id (unique_id)	Unique identifier for the trip
deleted (Bool)	Mark tuple for display/no display
accepted (Bool)	Mark tuple as accepted/ not accepted
comments (text)	Comments for the passenger

## RideShare's Models

<b>Users_model</b>	Data interaction for the users
<b>Trips_model</b>	Data interaction for the trips
<b>Requests_model</b>	Data interaction for the requests
<b>Passengers_model</b>	Data interaction for the passengers

## RideShare Model Functions

<b>User_model</b>	
add_user()	Adds a user to the users table
remove_user()	Removes a user from the user table
update_user()	Updates a user in the user table
get_users()	Queries the users table for the index or a specific user_id
validate_user()	Validates users for login
get_prefs()	Queries the users table for the preferences
update_prefs()	Updates the users table for preferences
reset_prefs()	Reset the users table for preferences
<b>Trip_model</b>	



add_trip()	Adds a trip to the trips table
remove_trip()	Removes a trip from the trips table
update_trip()	Updates a trip from the trips table
get_trips()	Queries the trips table for the index or a specific trip_id
Request_model	
add_request()	Adds a request to the requests table
remove_request()	Removes a request from the requests table
get_requests()	Queries the requests table for the index or a specific request_id
Passenger_model	
add_passenger()	Adds a passenger to the passengers table
remove_passenger()	Removes a passenger from the passengers table
get_passengers()	Queries the passenger table for the index or a specific user_id or trip_id

The View is the information being displayed on the front end to the user in the form of a page, page fragment, or any other type of "page".

### RideShare's Views

<b>add_user</b>	New user registration form view
<b>login</b>	User login and registration view
<b>interface</b>	User dashboard, ride/request tables, & control buttons view
<b>add_trip</b>	User add trip form view
<b>trip_display</b>	Unique trip information and control buttons view
<b>add_request</b>	User add request form view
<b>request_display</b>	Unique request information and control buttons view
<b>user_preferences</b>	User preferences and settings dashboard view
<b>admin_preferences</b>	Admin preferences and CRUD privileges dashboard view

<b>help_pages</b>	This is the user help pages for RideShare view
-------------------	--

The Controller serves as an intermediary between the Model, the View, and any other resources needed to process the user's request and generate a web page.

## RideShare's Controllers

<b>Login</b>	Handles all login related functions such as authentication
<b>Interface</b>	Handles all the front end user interface functions
<b>Users</b>	Handles all users related functions such as update()
<b>Trips</b>	Handles all trip related functions such as add()
<b>Requests</b>	Handles all request related functions such delete()
<b>Passengers</b>	Handles all the passenger related functions such as add()

## RideShare Controller Functions

Login	
index()	Loads the login view
authenticate()	Validates user and redirects to the interface view
Interface	
is_validated()	Check if user is validated and creates a new session id. The function redirects to the login view if validation fails
index():	If is_validated() passes, this calls model functions for user, trip, and request information and loads the 'interface' view while passes \$data from the model
help()	This loads the help pages view
logout()	Ends session and redirects to the login view
Users	
sign_up()	Loads the add_user view
preferences()	Queries the model for user preferences and displays the user_preferences view

update()	After validation, controller loads model and runs the update_user() function and returns
add()	After validation, controller loads model and runs the add_user() function and returns
remove()	Loads the model function remove_user() and marks the tuple as deleted="true" and returns
<b>Trips</b>	
index()	Loads the available list of trips from the Trips_model and generates a JSON file
add()	After validation, controller loads model and runs the add_user() function and returns
remove()	Loads the model function remove_trip() and marks the tuple as deleted="true" and returns
update()	After validation, controller loads model and runs the update_trip() function and returns
view()	Loads the trips_display view after calling the Passengers_model and Trips_model
edit()	Gets the trips data from the model and loads the add_trip() form with updated and filled out data. Goes to update_trip() from there.
<b>Requests</b>	
index()	Loads the available list of requests from the Requests_model and generates a JSON file
add()	After validation, controller loads model and runs the add_request() function and returns
remove()	Loads the model function remove_request() and marks the tuple as deleted="true" and returns
update()	After validation, controller loads model and runs the update_request() function and returns
view()	Loads the requests_display view after calling the Trips_model
<b>Passengers</b>	

view()	Loads the available list of requests from the Passengers_model and generates a JSON file
add()	After validation, controller loads model and runs the add_passenger() function and returns
remove()	Loads the model function remove_passenger() and marks the tuple as deleted="true" and accepted="false" and returns
accept()	Loads the model function add_passenger() and marks the tuple as deleted="false" and accepted="true" and returns

## How the User Interacts with CodeIgniter

In CodeIgniter when a user makes a request, the framework reads the URL to figure out what the user wants it to do. This is an approach called segment-based matching. Lets take the URL <http://rideshare.org/view/trip/12> where after rideshare.org, each word in the URL has a specific meaning. CodeIgniter's controller reads the URL splits it up into segments so that it knows you want to "view" a "trip" with an id number of "12".

This is how CodeIgniter process the URL:

- *example.com/<controller\_call>/<controller\_function\_call>/<identifier>*

## RideShare URL Parsing Examples

Lets take <http://rideshare.org/users/remove/1345>.

This invokes the "Users" controller and it then knows that you want to "remove" a user with an id of "1345" using the remove\_user() function.

Lets take <http://rideshare.org/trips/update/1566>

This invokes the "Trips" controller and it then knows that you want to "update" a trip with an id of "1566" with data posted from the update form using the update\_trip() function.

Lets take <http://rideshare.org/passengers/accept/7890>

This invokes the "Passenger" controller and it then knows that you want to "accept" a user with an id of "7890" to be a passenger on your trip using the accept() function.

## Project's State

Currently, RideShare is in the planning phase of the project with a front end prototype set up for further development. The idea is to do prototyping and set up the project so that contributors in the Open Source community can start working on the implementation.

# Project Implementation Plan

Finalization of project plan

- This includes a careful analysis of CodeIgniter functions and logic

Database Setup

- Write all create statements and save them as rideshare\_setupdb.sql

Prototyping the RideShare framework

- Set up all controllers, views, and models with their respective function declarations. (No need for logic, just structure)

Set up Views

- Set the views up with sample data and test if the controller can reach them

Set up Logging and Error Handling

- This will make everyone's experience much better

Set up Development documentation and a project wiki

- This will help get the Open Source community to start work on the project
- Might be able to set up automatic documentation generation

Start Programming

- Start with these controllers and work down from there: Users, Trips, Passengers, and Requests

Get ready for v1 release!