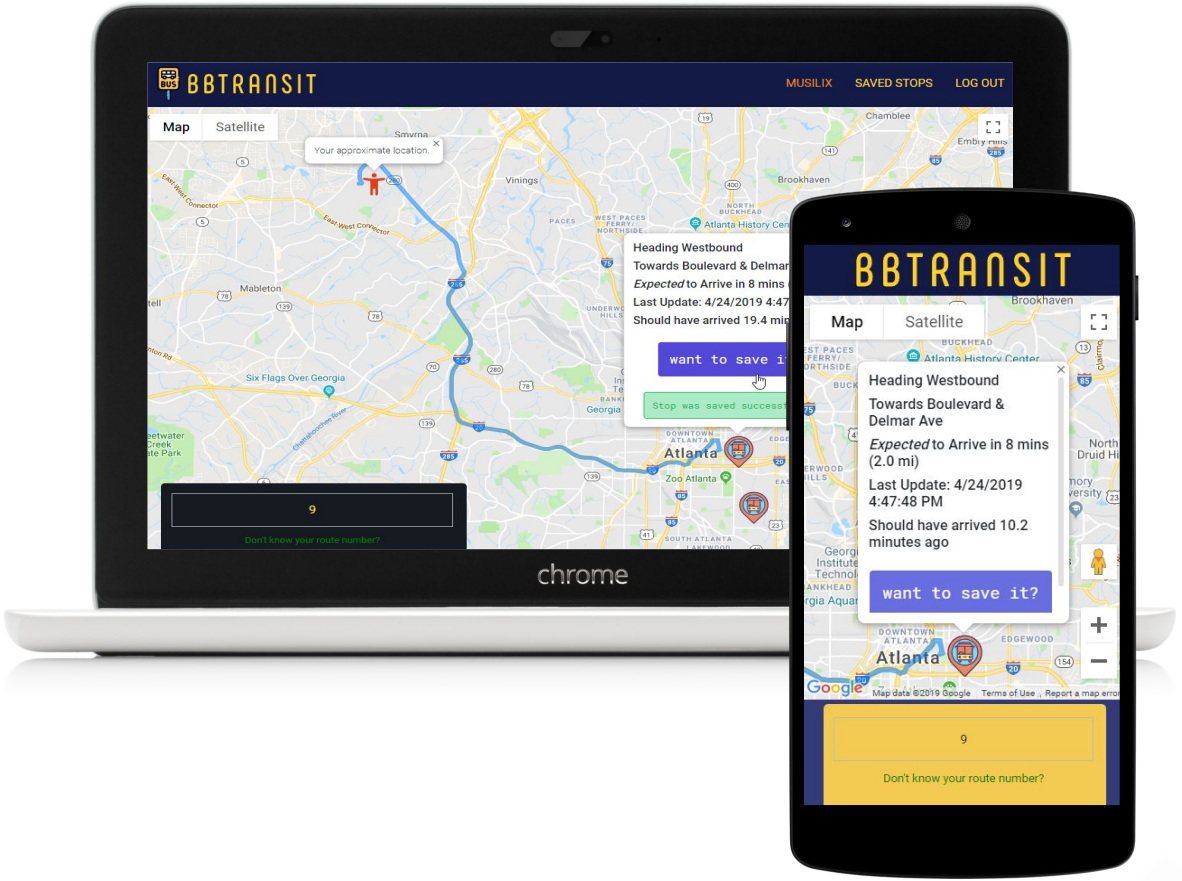
BBTransit

Team BusBoys:
Kareem Shehab
Forrest Lybarger
Pavan Namani
Earvin Bako

What we are and Why we are.

BBTransit is a web based transport alert app that aims to simplify the daily life of a commuter by offering access to simple yet meaningful info such as bus arrival times, destinations, locations, and their adherence to the schedule; all in real time.

Specifically focused on MARTA buses around Georgia, since MARTAs own apps for doing this are regarded to so poorly currently.



Acknowledging Competitors

As stated, *MARTA* itself has its own application. But it lacks fundamental qualities of a reliable application as it:

- Crashes frequently
- Relays inaccurate data - ETA
- Lacks simplistic UI

There is another; *One Bus Away*, which is outside of GA. Some of their functionalities were looked upon as reference in our system.

Planning & Scheduling

We established what each member would focus on early on: Forrest and Kareem would work more with the actual coding, while Earvin and Pavan would work more on the design, architecture, and testing.

Initially, we had no consistent meeting time, but as time went on every Friday or every other Friday would become our reserved day for meeting, filming, and discussing the project. Finally, it became as schedule to meet from 3-6 on Fridays.

Collaboration

Slack came to be a very helpful tool for structuring and centralizing all the needs for our project. Communication was facilitated, and it came to be a key in keeping us organized and on track.

Github added onto this, keeping our goals clear and set, all while providing a good repository setting to keep our systems code up to date and viewable by everyone on the team.

The Development Process

Like many others… we went with the waterfall model. Just kidding, we opted for the *incremental development method* as we knew the future was a bit uncertain.

Requirements in our system could be subject to change, and so, to combat that uncertainty, an incremental method could be employed to reduce the cost of any changes down the line of our system's creation.

For example, although many parts of our system work, we could always go back in the future and add more onto, since we went with this incremental method.

Requirements and Elicitation

User Requirements

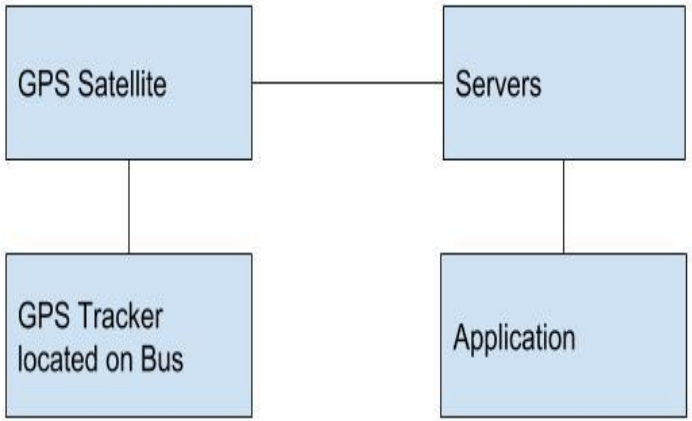
- Tracking app that will help people the public transit system estimate the arrival time of their buses to plan ahead and avoid missing their bus or wait for a elongated period of time.
- It aims to limit inconsistencies by providing them with bus timing, alerting users about possible time changes and schedule updates.

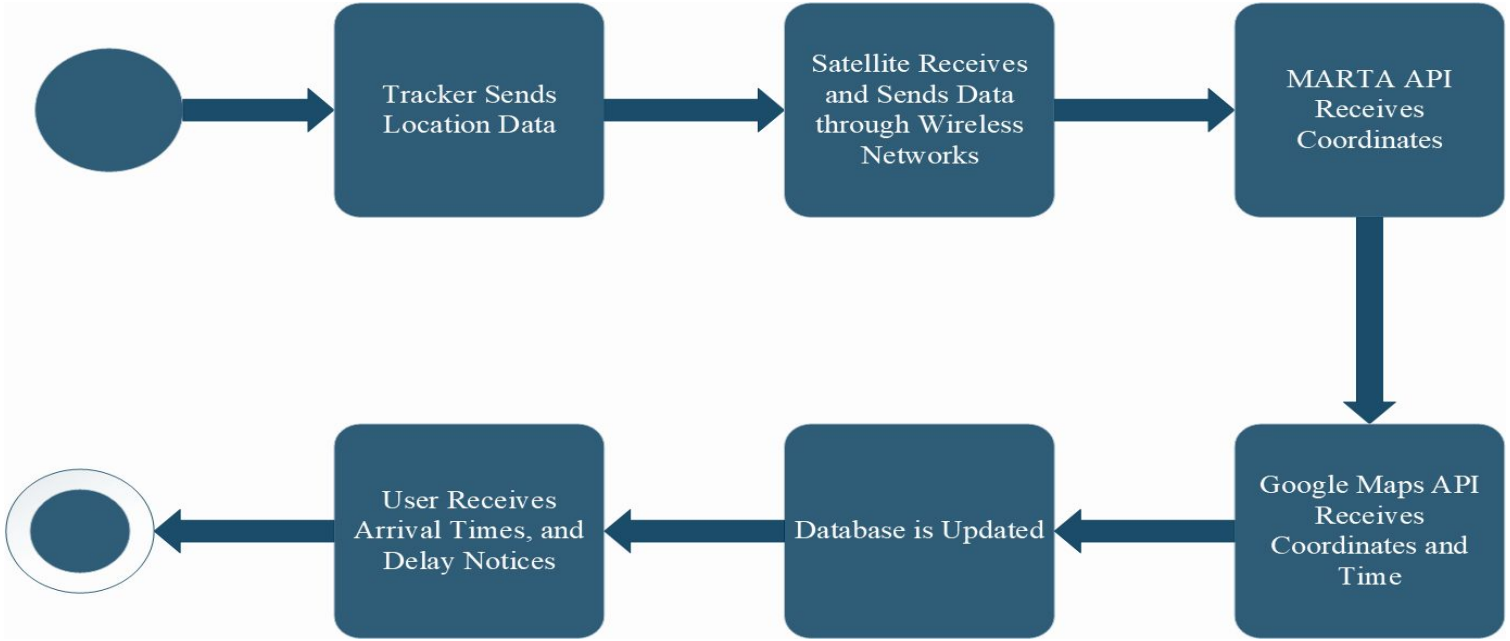
Requirements and Elicitation

System Requirements

- Provide user with a accurate location of their buses.
- Provide users with arrivals times of their buses.
- Help user map their route and plan ahead.
- Provide user with an alert system that will inform them on possible changes in arrivals time and delays.

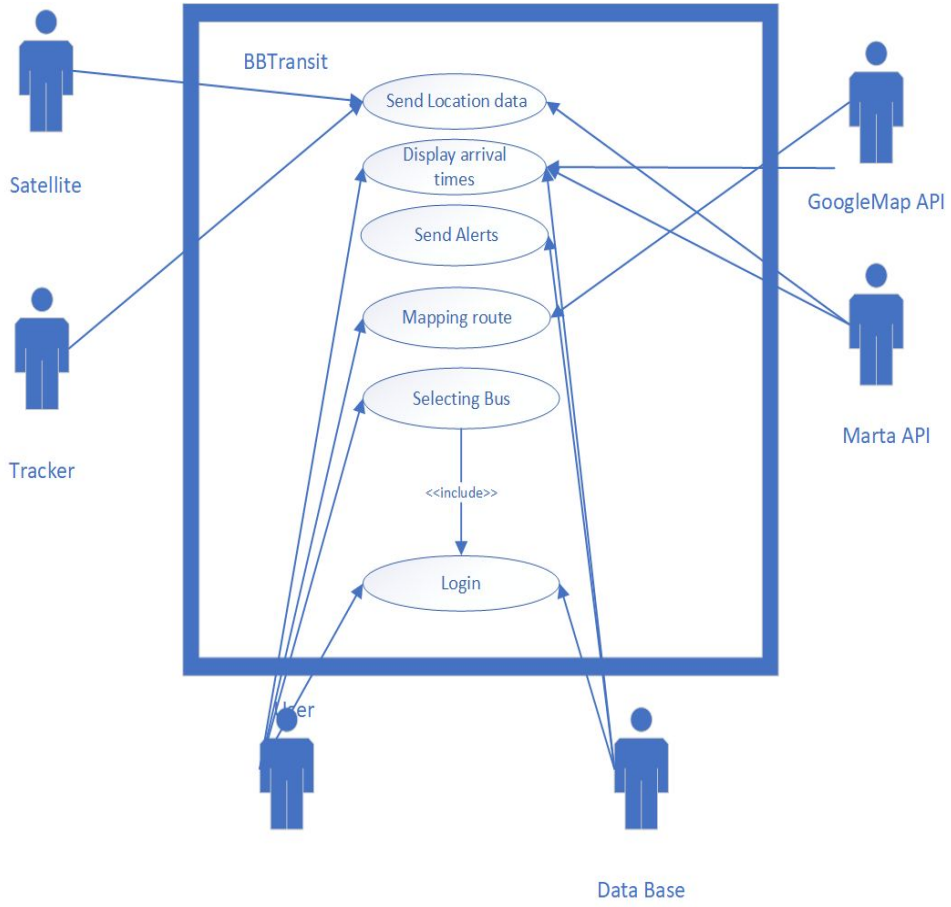
Context Diagram





Activity Diagram

Use case diagram



Use Case example

Use Case: Send Location Data

Actors: Tracker (initiator), Satellite, and MARTA API

Summary: Tracker sends information to satellite, which sends to MARTA API

Description:

1. The tracker located in the buses of MARTA send their locations(*longitude & latitude*) to a satellite.
2. The satellite redirects its input into MARTA API.
3. The tracker updates the bus coordinates repeatedly every few minutes.
4. Once those coordinates are in MARTA API, we extract them out into Google Maps API for arrival ETA.
5. Complete Use Case *Transmit Data*.

Exception Path: If the sensor/tracker is unable to send coordinates due to a mishap in step 1, execution jumps to step 4, and the coordinates will not be updated in MARTA API.
Pre-Condition: The tracker must be on, and work successfully.
Post-Condition: The bus's coordinates are updated.

Design Pattern - Observer

Coupling:

Our system has a **high** level of coupling because the code that is being shared through each of our components comes from Marta's API (the bus coordinates and id). However, to limit that, once we get the bus's coordinates and plug it into the Marta API to get the arrival times, we saved the one user selected in our Database, only to be changed when there is a increase in the traffic density given from Google map' API.

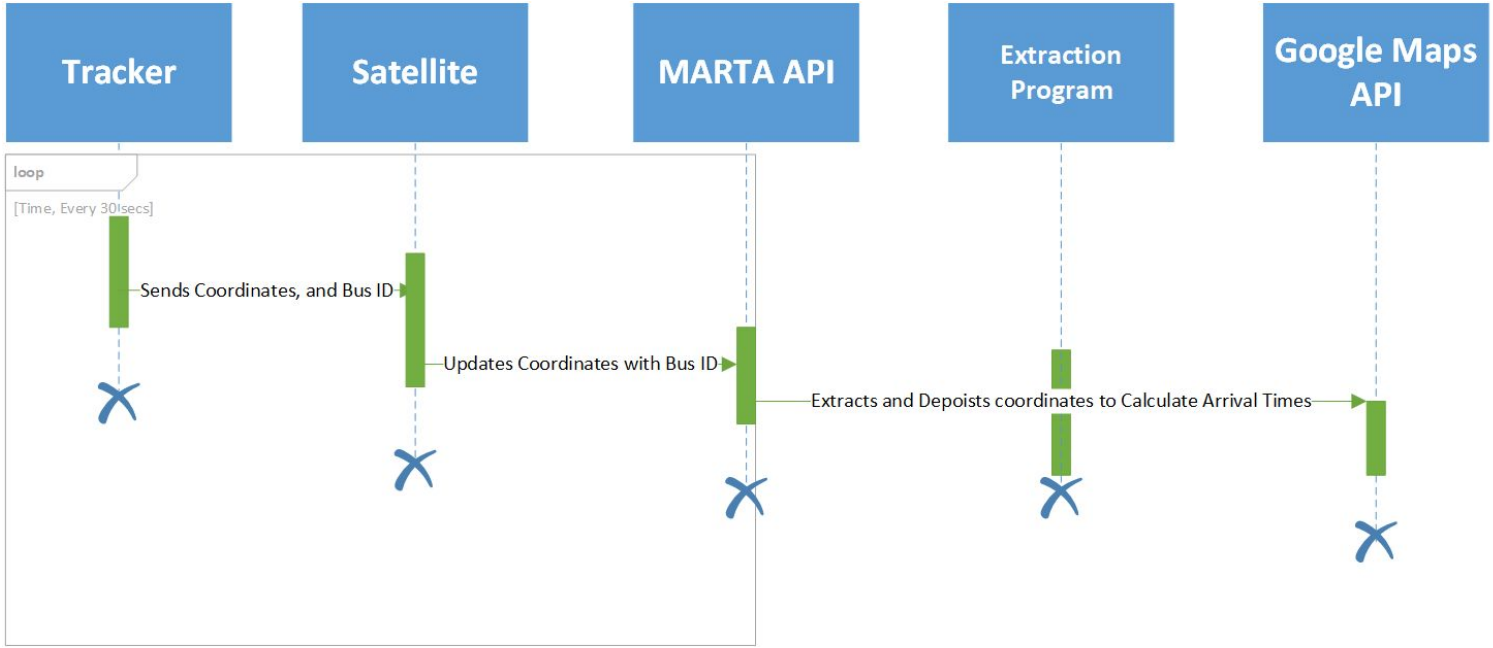
Cohesion:

Our system also has a **high** level of cohesion because the output of one component is often the input of the next, as is the case with the bus coordinates generated from the Marta's API becoming the input of the Google maps API. This is good because changes are localized to one single cohesive component.

Pattern Chosen:

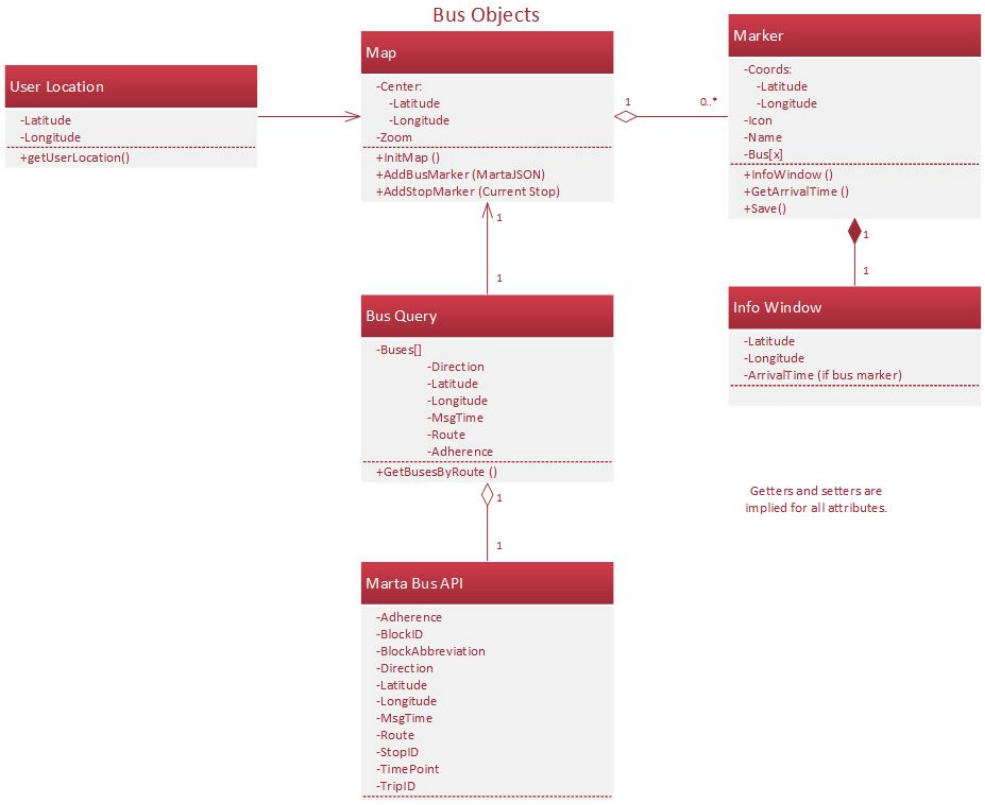
From our system features and the way data is shared across our system, we opted to go with an **Observer pattern** because Marta's API output is really the subject to all our other components. Our system relies primarily on the Marta API updates to provide all our functionalities.

Sequence Diagram from Design



-Retrieves Tracker Data from MARTA API using an Extraction Program as a pipe.

Class Diagram from Design

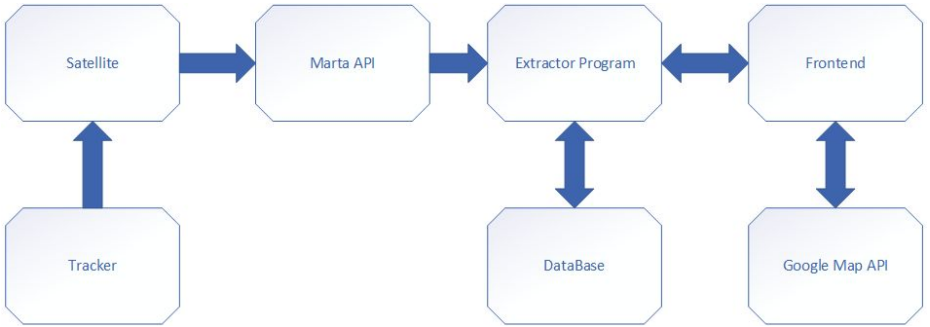


Architecture

The **Pipe and Filter** model came to be the architecture our system was based around, mostly due to the fact that rather than having a whole amalgamation of monolithic modules, we had a more of a stream of data from module to module; with each transforming the data in some manner.

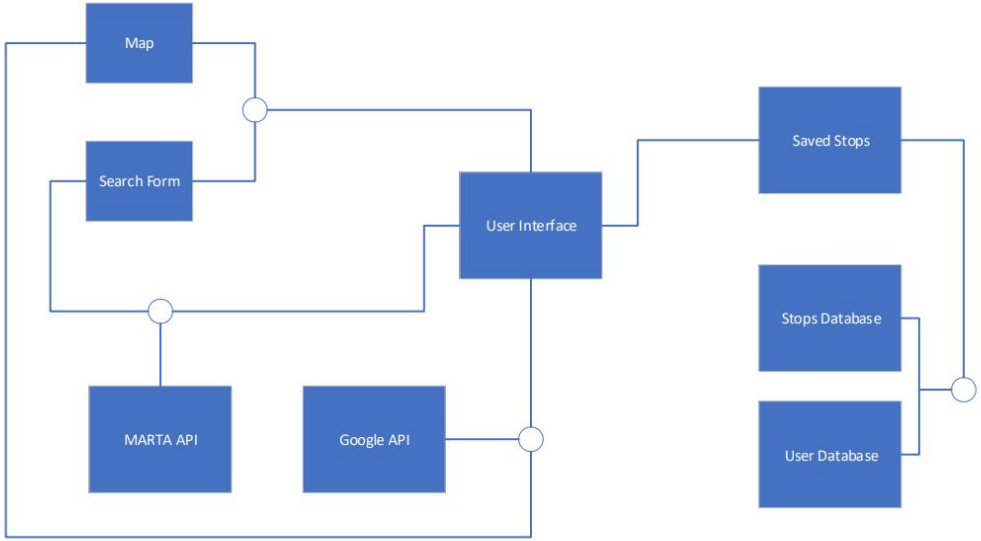
Promotes code reusability and modularity...

But also leads to high coupling.



4+1

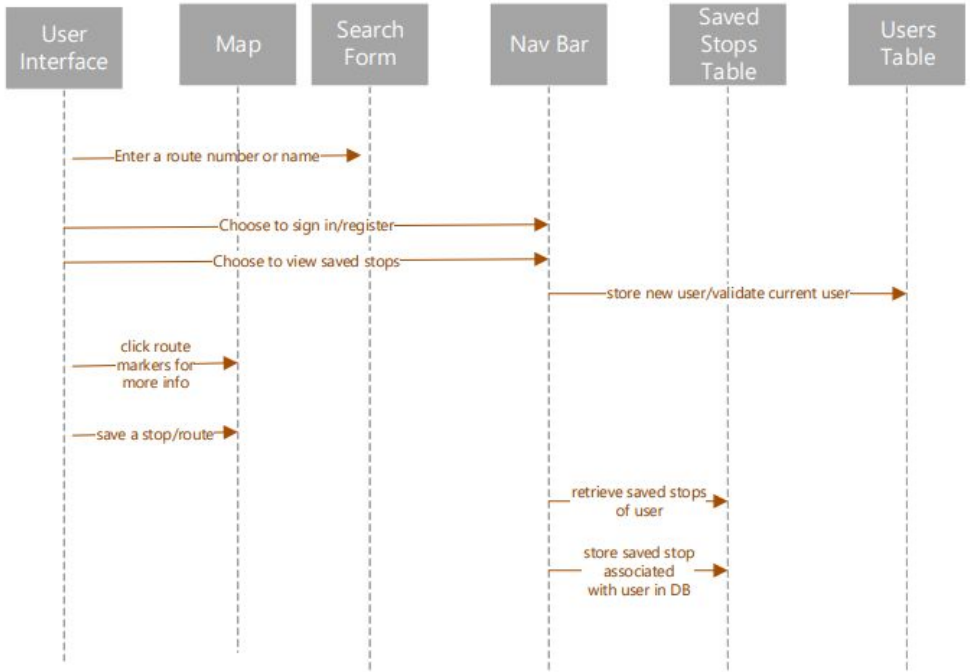
Logical View



We can map our functionality/use case scenarios onto specific components of our system.

4+1

Process View

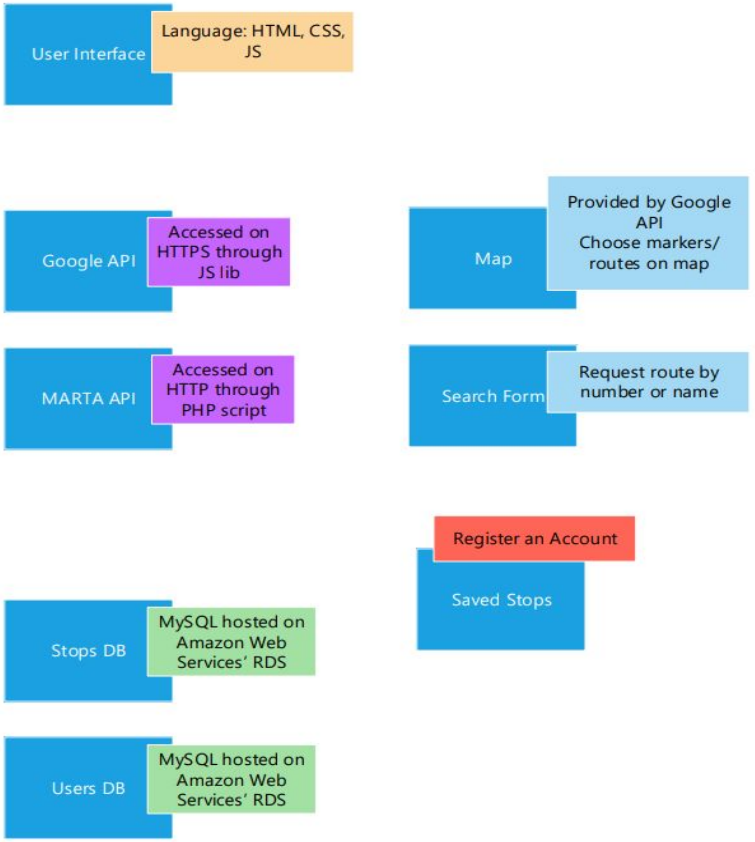


Specific processes and threads which will occur can be described.

4+1 = 3?

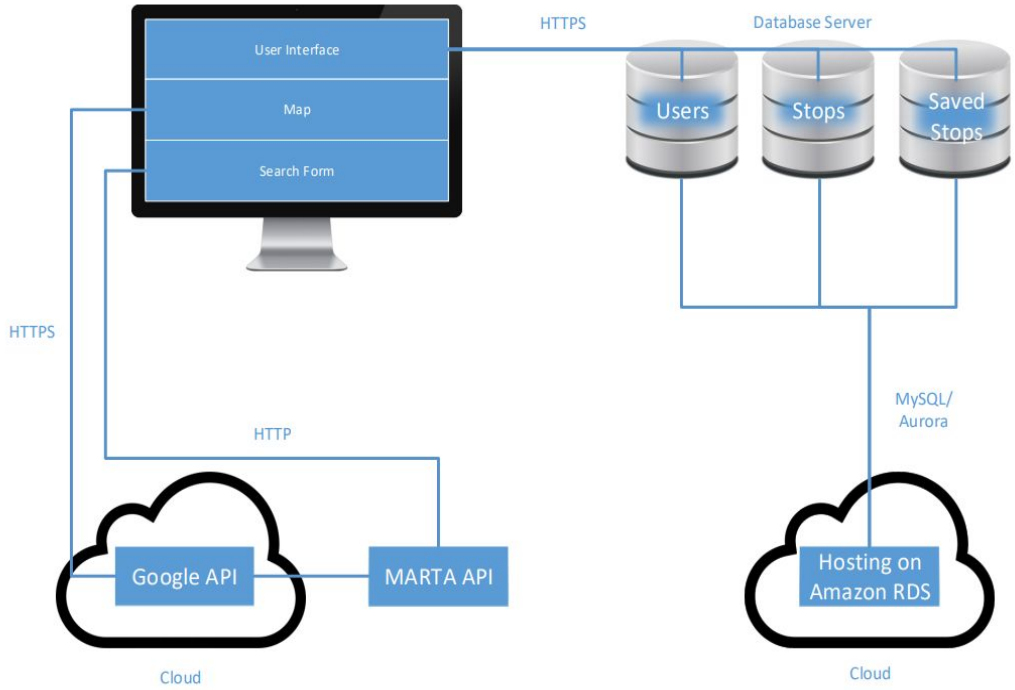
The technologies that are utilized to accomplish the tasks set forth by our system are as followed.

Development View



4+1

Physical View



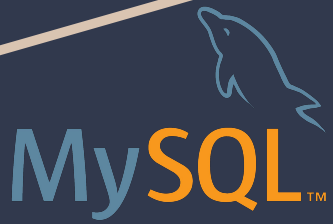
The manner in which the hardware we use coincides with the software components that are utilized.

Implementation

APIs/3rd Party Services

- Google Maps API
  - Components: Map, Direction, Geolocation
  - Front end mapping
  - ETA calculation
  - Location services
- Marta API
  - Real time bus location
  - Bus adherence
  - Route database
- Amazon RDS
  - Online Database Hosting
  - Login/Register
  - Stop storage
- Amazon EC2
  - Free web hosting
  - Allows dynamic websites (PHP)
- IP API
  - User location

Languages



- JavaScript
  - Interactive front end
  - Integration with Google Maps API
  - HTTP calls to PHP
- PHP
  - Connection with RDS
  - Query Marta API
  - Secure server-side authentication
- MySQL
  - Relational database
  - Integration with Amazon RDS

IDEs/Text Editors

- Atom
  - Lightweight
  - Helpful packages/modules
  - JavaScript and PHP
- Sublime Text
  - Lightweight
  - Helpful packages/modules
  - JavaScript and PHP
- MySQL Workbench
  - UI for working with database
  - Connected to Amazon RDS
  - MySQL

Geolocation and Route Estimation

```
function geocodeCurrLoc(element){
      let geocoder = new google.maps.Geocoder();
      let address = element['TIMEPOINT'] + " Atlanta, GA";
      let userGivenDest = [];

      geocoder.geocode({'address': address}, function(results, status){
          if(status === "OK"){
            let lng = results[0].geometry.location.lng();
            let lat = results[0].geometry.location.lat();

            userGivenDest.push(lat);
            userGivenDest.push(lng);

            estimateRoute(element, userGivenDest);
          }else{
            return "There was a geocoding error!";
          }
      });
}
```

```
function estimateRoute(element, dest){
        let origin = new google.maps.LatLng(element['LATITUDE'], element['LONGITUDE']);
        let destination = new google.maps.LatLng(dest[0], dest[1]);

        // must create a new directions sesrvice object in order to obtain route from origin to dest
        let directionsService = new google.maps.DirectionsService();

        // fill request var with our origin and dest objs, and then travelmode
        let request = {
            origin: origin, // LatLng|string
            destination: destination, // LatLng|string
            travelMode: google.maps.DirectionsTravelMode.DRIVING
        };


        // send those request details to the direction services  route method, and have callback fnct
        //log error if status isnt returned as OK; otherwise, print out the first routes legs duratio
        directionsService.route(request, function( response, status ) {
            if ( status === 'OK' ) {
                  let point = response.routes[0].legs[0];

                  let currETA = (point.duration.text + " (" + point.distance.text + ")");
                  addMarker(element, currETA);
                  // return point.duration.text + " (" + point.distance.text + ")";
            }else{
                console.log("FAIL");
                window.alert('Directions request failed due to ' + status);
                // return "route est. failed";
            }
        });

}
```

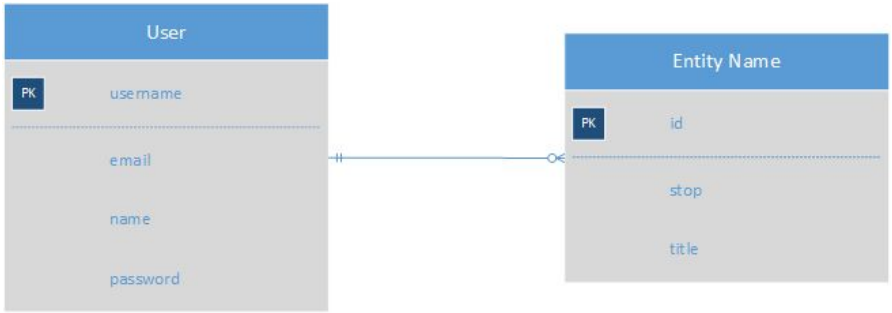
Placing Markers

```
//Add Marker function
function addMarker(element, currETA){
  let lat = parseFloat(element["LATITUDE"]);
  let lng = parseFloat(element["LONGITUDE"]);

  // console.log(lat + " " + lng);
  let dest_image = {
          url: './assets/png/bus-stop.svg',
          // This marker is 20 pixels wide by 32 pixels high.
          size: new google.maps.Size(50, 50),
          // The origin for this image is (0, 0).
          origin: new google.maps.Point(0, 0),
          // The anchor for this image is the base of the flagpole at (0, 32).
          anchor: new google.maps.Point(0, 25)
  };

  let marker = new google.maps.Marker({
    position: {lat, lng},
    map: map,
    icon: dest_image,
    optimized: false,
    draggable: false,
    animation: google.maps.Animation.DROP
  });

  let timeUser = element['MSGTIME'].split("  ")
  let googleETAMins = parseInt(currETA.charAt(0));

  // console.log("TIME: " + timeUser);

  let respectToAdherence = getRealTimeDiff(timeUser, googleETAMins);
  console.log(respectToAdherence);
  let etaResp = "";
  if(respectToAdherence < 0){
    etaResp = "Should have arrived " + Math.abs(respectToAdherence) + " minutes ago";
  }else{
    etaResp = "Will be arriving in " + + Math.abs(respectToAdherence) + " minutes";
  }

  let infoCon = {
    content: '<h6>Heading ' + element["DIRECTION"] + '</h6>' +
             '<h6>Towards ' + element["TIMEPOINT"] + '</h6>' +
             '<h6><i>Expected</i> to Arrive in ' + currETA + '</h6>' +
             '<h6>Last Update: ' + element["MSGTIME"] + '</h6>' +
             '<h6>' + etaResp + ' </h6>' +
             '<div class="center-button"><h5><form id="saveStopForm"><in
             '<div class=success></div>'
  };



  //Info Window Constructor
  let info = new google.maps.InfoWindow(infoCon);

  let busLoc = {
      lat: element['LATITUDE'],
      lon: element['LONGITUDE']
  };

  //Click-to-Open
  marker.addListener('click', function(){
    calcRoute(busLoc, userPosition);
    // added functionality to close any previously open
    // info window when a new one is open
    if(prevOpenWindow){
      prevOpenWindow.close();
    }

    info.open(map, marker);
    prevOpenWindow = info;
  });

  markers.push(marker);
}
```

Calculate Route

```
function calcRoute(busLoc, userLoc){
  let directionsService = new google.maps.DirectionsService();
  // console.log("BUS details: lat(" + busLoc.lat + ") and lon (" + busLoc.lon + ")");
  // console.log("user details: lat(" + userLoc.lat + ") and lon (" + userLoc.lon + ")");
  directionDisplay.setMap(null);
  directionDisplay.setMap(map);

  let user_image = {
          url: './assets/png/user.svg',
          // This marker is 20 pixels wide by 32 pixels high.
          size: new google.maps.Size(40, 40),
          // The origin for this image is (0, 0).
          origin: new google.maps.Point(0, 0),
          // The anchor for this image is the base of the flagpole at (0, 32).
          anchor: new google.maps.Point(0, 20)
  };

  let userMarker = new google.maps.Marker({
    position: new google.maps.LatLng(userLoc.lat, userLoc.lon),
    map: map,
    icon: user_image
  });

  let userInfo = new google.maps.InfoWindow({content: "Your approximate location."});
  userInfo.open(map, userMarker);

  let request = {
    origin: new google.maps.LatLng(userLoc.lat, userLoc.lon),
    destination: new google.maps.LatLng(busLoc.lat, busLoc.lon),
    travelMode: google.maps.DirectionsTravelMode.DRIVING
  };

  directionsService.route(request, function(response, status){
          if (status == google.maps.DirectionsStatus.OK) {
            directionDisplay.setDirections(response);
          }
  });

```

Database



- **GetBusByRoute** — returns those active buses with real-time data *for a given route*
  http://developer.itsmarta.com/BRDRestService/RestBusRealTimeService/GetBusByRoute/{ROUTE}

Sample Response


```
[ {
    "ADHERENCE": "4", "BLOCKID": "31",
    "BLOCK_ABBR": "110-4",
    "DIRECTION": "Northbound",
    "LATITUDE": "33.8346347",
    "LONGITUDE": "-84.3824637",
    "MSGTIME": "5\/14\/2013 12:14:04 AM",
    "ROUTE": "110",
    "STOPID": "900456",
    "TIMEPOINT": "Peachtree Hills & Peachtree",
    "TRIPID": "3719918",
    "VEHICLE": "2853" }, ... ]
```

Testing

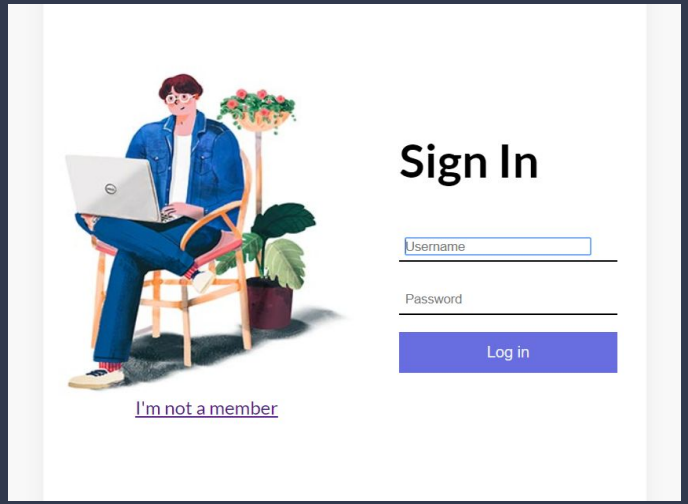
- Testing was done in HTMLUnit and PHPUnit as implementation was done in PHP, and JavaScript.
- We used Chai and Mocha Frameworks to complete Testing.
- Testing:
  - Correct Marker Location
  - Register User Successfully
  - Saving Bus Stops

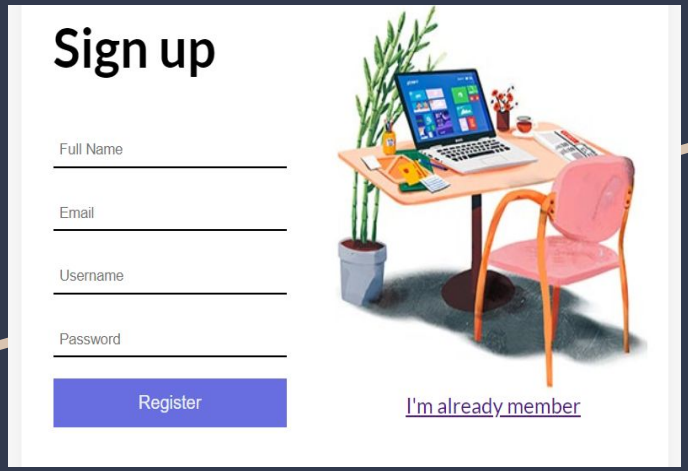
Correct Marker Location- Coding Test

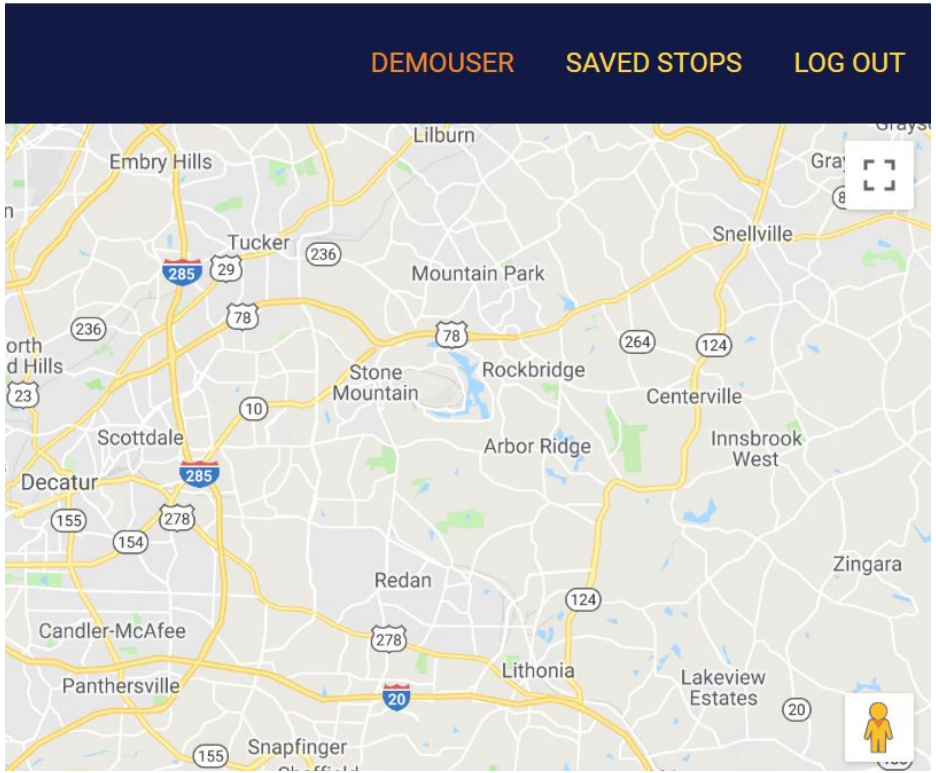
Chai Framework Unit Testing code ensuring that marker is placed on the correct location from the MARTA API.

```
//necessary dependencies
const mapper = new mapBus();
const assert = chai.assert();

// This testing element represents a bus/route object returned by the MARTA API we utilize. It contains
// the following attributes. For the case of testing addMarker(), we are concerned with latitude and longitude
test_element = {"ADHERENCE": "4", "BLOCKID": "31", "BLOCK_ABBR": "110-4", "DIRECTION": "Northbound",
                "LATITUDE": "33.8346347", "LONGITUDE": "-84.3824637", "MSGTIME": "5\/14\/2013 12:14:04 AM",
                "ROUTE": "110", "STOPID": "900456", "TIMEPOINT": "Peachtree Hills & Peachtree", "TRIPID": "3719918",
                "VEHICLE": "2853" };

// we save the response of the addMarker(element, currETA) method, passing it a dummy value for the currETA parameter, as we
// currently concerned with how it is working with the elements lat and long
addMarkerResponse = mapper.addMarker(test_element, 0);

// using mocha unit testing syntax, we do the following
describe('Testing Marker Creation', () => {

    //testing that the obj returned by addMarker() should have longitude corresponding to the element we sent in
    it('should have latitude corresponding to the element we are using', () => {
        assert.equal(test_element['LATITUDE'], addMarkerResponse.position.lat);
    });

    //testing that the obj returned by addMarker() should have latitude corresponding to the element we sent in
    it('should have longitude corresponding to the element we are using', () => {
        assert.equal(test_element['LATITUDE'], addMarkerResponse.position.lng);
    });
});
```

Testing: Register User

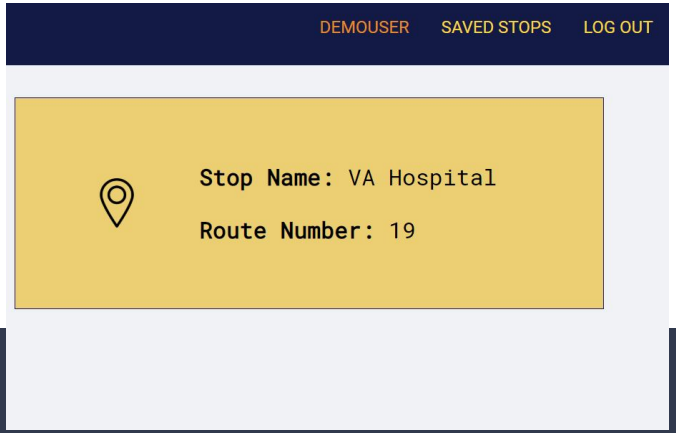






Testing: Saving Bus Stops– Test Case Script

Test ID: savingBusStop_Test
Purpose of Test: The purpose of this test is to check whether the savingBusStop program works correctly, and displays the selected bus stop(s) on Saved Stops page. The user must be able to click on them to ease their access to specific stops.
Test Environment: The code was made in PHP, thus we chose to use PHPUnit for this test case. The results and content should be visible on Saved Stops Page.
Test Input: A Bus & its location.
Test Expected Results: Saved Bus on Saved Stops Page.



Demo…here

Future Improvement

We could have used more:

- Computational Power
- Time & money

We planned to install:

- Sending alerts to User of incoming buses to Saved Stops.
- Mapping stops to routes

Outgoing Aspects:

- We definitely challenged ourselves.

Questions?