P5: Schedule Mapper Revised Version

Revise your project using the feedback from reviewers. Prepare a 500 word narrative summarizing the revisions for suggested improvements from reviewers. Place the revised presentation and report on your web-site and ensure that the course web-site has a link to your material.

See Schedule Mapper at: http://moorespatial.com/csci5980/ It will require access to a Google Calendar.

Goals
The larger context of our goal is that we are interested in adaptive user interfaces and presenting the most relevant information to the user through adaptive maps.

To this end, our goal is to provide the User with a map and directions for their day based on their schedule, and current position. Use the Google Calendar API to discover the User's scheduled events and appointments, then use the Google Maps API to geocode (where needed) the events into location coordinates, and plot the User's future travels on a map.

On the map we will use coloring and shading to provide contextual clues as to which locations are the User's next destinations.

By using the Google APIs instead of doing a proof-of-concept project with UMN schedule and map data we will be able to create a map and project which will be useful to a larger audience.

Work Report
We used an agile development method of development. As we began planning the implementation of our project it became obvious that we had more ideas than time, and agile development methodology gave us the opportunity to continually re-evaluate which features were most important and which would best compliment the features we had already implement.

Development Setup
Server: We used an Apache2 server with the mod_php module installed.

Programming: All programming was done in VIM. Testing: Firefox 16 was used, but any browser which supports HTML5/CSS3 should work.

Debugging: The Firebug and Web Developer Toolbar was essential for debugging errors in our JavaScript code.

Other: We had to configure the Google API Console to get a developer key and a secret OAuth token.
Project Milestones

Milestone 1 Our first milestone was to implement oAuth authentication with Google. oAuth allows us to access Google services with the current user's credentials without the user entering their password on our site.

Getting oAuth working involved installing the Google PHP Client libraries on our server, and creating a config file with the appropriate key and token.
Milestone 2 Next we needed to fetch the user's calendar data. We started by fetching and dumping the raw data (in JSON format) to see what information we had access to. Once we understood the format we printed it in a static list, sorted by event start time.

Limitations: As we looked at the list of events we realized that many users would have repeating events. Since repeating events would be in the same location, it would likely be difficult to display more than 24 hours of a user's schedule on a map. We decided to always display the next 24 hours.

Milestone 3 Once we had their schedule we next wanted to print a route between all events in their schedule. For milestone 3 we decided to ignore any events without locations and any events without a time (ie. All-day events). We also drew a single route to the map, using waypoints for all intermediate events. Limitations: We discovered that there was no way to apply different colors to each branch of the route and no way to apply a color gradient to a polyline on Google Maps. This meant that for milestone 4 we would need to work out a different solution.

Milestone 4 For milestone 4 we broke the user's route into multiple routes so that between each event is a single route with no waypoints. This enabled us to implement color coded route segments. With milestone 4 complete the user's path progresses from red to blue as they go through the day.

Limitation: Routes on Google Maps display alphabetic markers by default. Since we had multiple routes with 2 nodes each, we ended up with multiple A and B markers, many of which overlapped (since the end of one route is typically the start of another route). We had to hide the default markers and place our own.

Milestone 5 Most users don't enter location information for all of their calendar events, making it impossible to route their day on the map. With milestone 5 we were ready to resolve this issue. We first stubbed out the function calls with a simple JavaScript pop-up. Although the pop-ups were intrusive, they allowed us to implement the route updating and user-input handling code.

Milestones 6

As we neared the project deadline we started working on user-interface issues to make our project more appealing and useful. We replaced the JavaScript pop-ups with on-page field forms and JavaScript change detection triggers. We also implemented automatic zooming which made sure that all points in the user's day were visible initially. We implemented the HTML5 geolocation feature so that the user's
starting point was determined by their current location, regardless of where their first scheduled event was.

Milestone 7 The final milestone was to make the site more visually appealing and to do final debugging. During this milestone we wrote the CSS and added project information on the site.

**Codified Documents**

The codified documents we used includes:

- Google OAuth2 Documentation: https://developers.google.com/accounts/docs/OAuth2
- Google Calendar API: https://developers.google.com/google-apps/calendar/
- Google Geocoding API: https://developers.google.com/maps/articles/geocodestrat
- W3C's JavaScript geolocation Documentation: http://dev.w3.org/geo/api/spec-source.html
- W3C's CSS3 calc() function http://www.w3.org/TR/css3-values/
- Google Directions API: https://developers.google.com/maps/documentation/directions/
- Google Maps API: https://developers.google.com/maps/documentation/
- Google Styled Maps Documentation: https://developers.google.com/maps/documentation/javascript/styling

**Goals For Learning**

We had two goals for learning from the codified documentation. First, we needed to learn enough to implement the needed features from the APIs. Second, we wanted to be familiar with the current state of the APIs so that we could have a discussion about future directions currently possible, as well as to make suggestions for future API versions.

**About The Codified Documentation**

Google OAuth2 Documentation In order to access a user's private Google data, including their calendar they must give us access to their data. The key piece of information we need from the OAuth exchange is their Google Account ID. We will use their account ID in subsequent API calls to fetch their personalized calendar data.

Google Calendar API The Google Calendar API is available through several different methods. The most basic is through the RESTful API. In this case we, the developers, would be responsible for implementing the calls to the API endpoints either through JavaScript AJAX calls, or server side.
The Calendar API is also available via client libraries in several languages including PHP and Python. The Python library is the only production ready client library. The PHP and other language client libraries are all in beta or alpha status. The use of the label Beta by Google usually indicates code that is nearly production-ready so the PHP libraries are likely still robust enough for our use. We have not yet decided which language to use in our project.

We will be using the Google Calendar API to fetch the user's calendar and to update individual events with geocoded location information.

Geocoding with the Google Maps API Geocoding will be an important task for our project. Most users will not enter locations for their events and if they do, their locations will not be in the form of coordinates which we can use for mapping. Google's Geocoding API offers two main options. Client-side geocoding is done from the browser with JavaScript AJAX requests. Server-side geocoding is done from the server with http requests. While server-side geocoding would make it easier to cache geocode results in our database, there is a limit of 2500 server-side geocode requests per IP address per day. Due to this quota limit we will need to implement our geocoding process on the client side.

The geocoding results includes both point information and geometry information for the building at the results location, where applicable.

W3C's JavaScript Geolocation While our project is a webmapping project, not specifically a mobile project, we will be using W3C's geolocation specification to make use of the user's current location. Many mobile and non-mobile browsers alike support this JavaScript API, although devices which are not GPS enabled may rely on IP Address geocoding to determine their location. IP Address geocoding may not be accurate enough to provide good directions.

Google Directions API With the geocoded results we can use Google's directions API to get routing information between events in their schedule. We will use the user's current location as the starting point, regardless of where their schedule says they should be. If no current location can be found, we will fall back to their previous scheduled appointment. The user will have an opportunity to change their current location so that we can update the routing to their next location. The Directions API allows us to specify the mode of transportation as one of driving, walking, bicycling or transit. The major gap in this list is air travel. We won't be attempting to find flights for the user but we will need to not route them between events for those trips either.

Another limitation of the Directions API is that there's not a way to specify mixed-mode transportation. In real life the user might take his car to the park-and-ride, then take the train or bus downtown, rent a Nice Ride bike to the bike station near his work, and then walk the rest of the way.
Google Maps API The Google Maps API will be used to display the locations of the user's upcoming events. After gathering the information we need with the calendar and geocoding APIs above we will draw that information on the map. For locations which are geocoded with a polygon we will draw the object's polygon. For objects without a polygon we will draw a marker object. Routing will be done with Google's routing API and we won't have to directly draw the routes.

In order to put emphasis on soonest events we will be using opacity and color. We will set the polygons opacity as they are created. Markers do not have an opacity setting so we will have to use JavaScript to set it directly.

Google Styled Maps Documentation Google provides an option to create styled maps. Unfortunately map styling is by feature class only, so we cannot use it to emphasize user destinations. We will however be able to use it to disable feature classes which we don't need, and to make the map look less busy.

Other Documentation Besides the several APIs which we will be using we will also be using JavaScript and jQuery to interact with the user. Our JavaScript and jQuery usage is incidental to the project and shouldn't present any surprises or limitations that we will need to be aware of.

Learning, Future Directions

Current Limitations

Some limitations are in the API, others are in the user-facing Google Products the APIs interact with.

• In Google Calendar there is a location field, but it is not spatially aware in any way. You can store any string you want in the field.

• It is difficult to get exact locations out of Google Maps. A user might want to do this so they can enter it into Google Calendar. To find a location you can search (and it is geocoded), or click on the map and create a marker. To get the coordinates out of Google Maps you need to look at the “Share this map” URL and extract the coordinates from there.

• You can't save geocoded locations back to Google Calendar. You could save a string with coordinates in it, but that's not very user-friendly for the user to view in their calendar entry.

• There's no way to make separate segments in a single trip different colors. Since we wanted to color code the trip path segments we were forced to make an additional API request for each path segment we need to create. Since there is a limit of requests for free usage, and since billing in a commercial environment is based on the number of requests, this isn't cost effective.

• The routing API doesn't return polygons for the waypoints. If we wanted to highlight specific buildings, parks or other destinations we would need to do a separate geocoding API call.

Currently Possible Directions

Some things which we would do if we had time:
• Color code map markers
• Enable Bus/Car/Walking choices
• Save entered locations back to Google Calendar (as strings, see above)
• Handle overlapping events
  ◦ We currently only detect scheduling conflicts with the same start times
• Detect when routing (car/bus/walking) will not allow enough time for the user to get where they're going
• Offset overlapping markers on the map

**Future Directions**

Besides the Currently Possibly (but unimplemented) features above, a smartphone with schedule mapping could automatically detect if the user is going to be late based on their mode of transportation and current location. An application like Schedule Mapper would be useful mostly for users who are in unfamiliar locations on an unfamiliar schedule. Use cases such as self-guided tourism and conferences and conventions come to mind. There may be recreational purposes too – a sort of cross between The Amazing Race and geocaching.

As per the suggestions of various classmates, we could also implement a smart schedule suggestion feature which would attempt to optimize the user's day.

We could also integrate with 3rd party data sources such as metro transit's bus scheduled.

**Screenshot**

![Schedule Mapper Screenshot](image-url)