Nomad

Use case Requirements:

- Best route from source to destination
- Provides weather information at particular times on the way to destination
- Stores the history of journeys suggested to the user
- After reaching destination, should suggest the nearby places to visit

Architecture:

We choose Web mash-up style architecture to design our application. As this best suites our requirements.

Why Mash-up?

- A Mashup is a web application that combines data from more than one source into a single integrated tool.
- Content used in Mashup is typically sourced from a third party via a public interface or so called API.
- Mash-up have genres and we are using Mapping mashups
- **Mapping mashups :** Google Maps, Yahoo Maps, Microsoft Virtual Earth

Sources for Mash-up:

- API/content providers
- Web Protocols: REST, Web Services, RSS/ATOM
- Screen Scraping

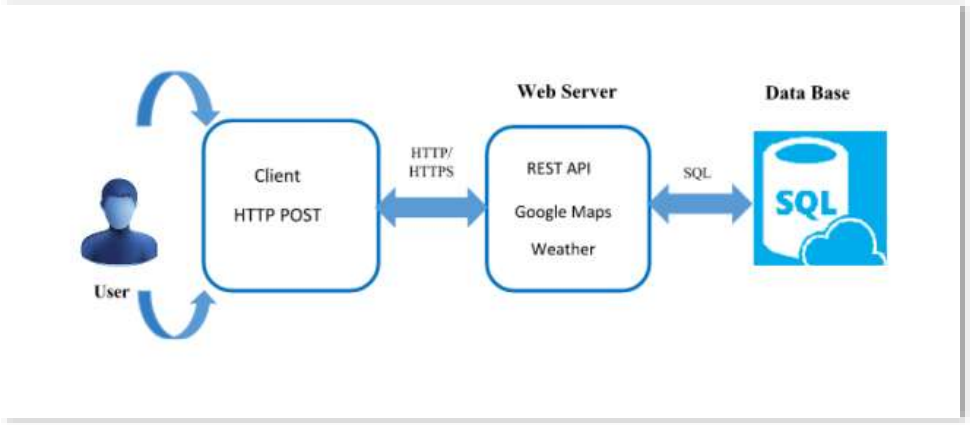
The mashup site:

- This is where the mashup logic resides, it is not necessarily where it is executed
- Server-side: Dynamic content aggregation
- Client-side: Client side scripting

The client's Web browser

- This is where the application is rendered graphically and where user interaction takes place

Server-side Mash-up:



Reuse of existing applications:

We are reusing the existing application like Google Maps, Weather API's. So that we need not rewrite the code. This reduces the effort of coding and also the existing applications are tested and working fine, it reduces the defects.

Rapidly develop applications:

We can rapidly develop any application using mashups with API's as they are already existing.

Using Mash-up does not require the user to have extensive IT or programming skills

Cost of application development is reduced greatly.

Design Considerations in Server-Side Mashups:

There are various things to consider when you do a server-side mashup:

- **Security:**

  Some of the security-related issues you need to address when you do a server-side mashup are the following:

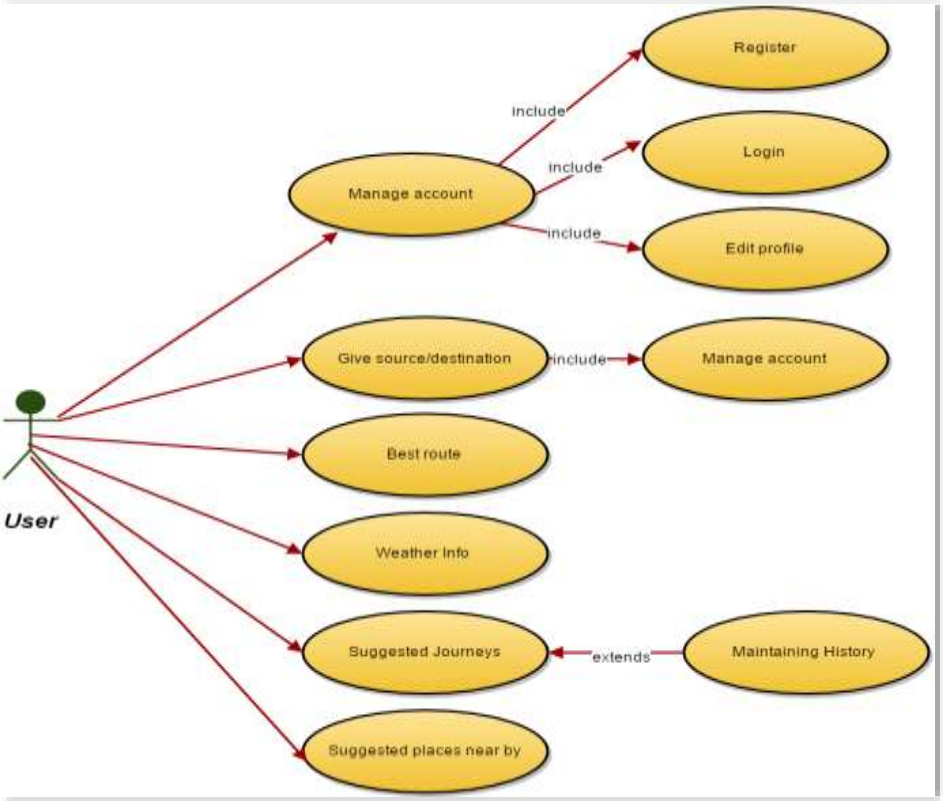
- Does the exposed service require the user to log in, is the service certificate-based, or is there an authentication mechanism?
- When you expose the service, are you opening a tunneling mechanism that a malicious developer might abuse? For example, can another developer use the service to initiate a denial-of-service attack on the service you are accessing?
- Can the service be used to hide the identity of a malicious client that is illegally using the service?
- Can a malicious user use the service to avoid paying fees?

- **The format of the response from the service:**
  - When you do a server-side mashup, you need to determine the format of the response that the service returns. A service can return a response in many different formats including XML, JSON, HTML, plain text, RSS/ATOM, and GData. In the Pet Store mashup with the Yahoo Maps Geocoding service, the client doesn't display the data returned by the service. It simply uses the data as input to the Google Maps service.
  - Other applications might need to convert the data returned by a service to another format such as JSON. Data in formats such as JSON can be easier for the browser to handle. By using the server as a proxy to a service, the server does the work of parsing an XML document and possibly converting it to a format such as JSON. In that case, your client JavaScript code can be simpler because it doesn't have to parse and convert the XML.

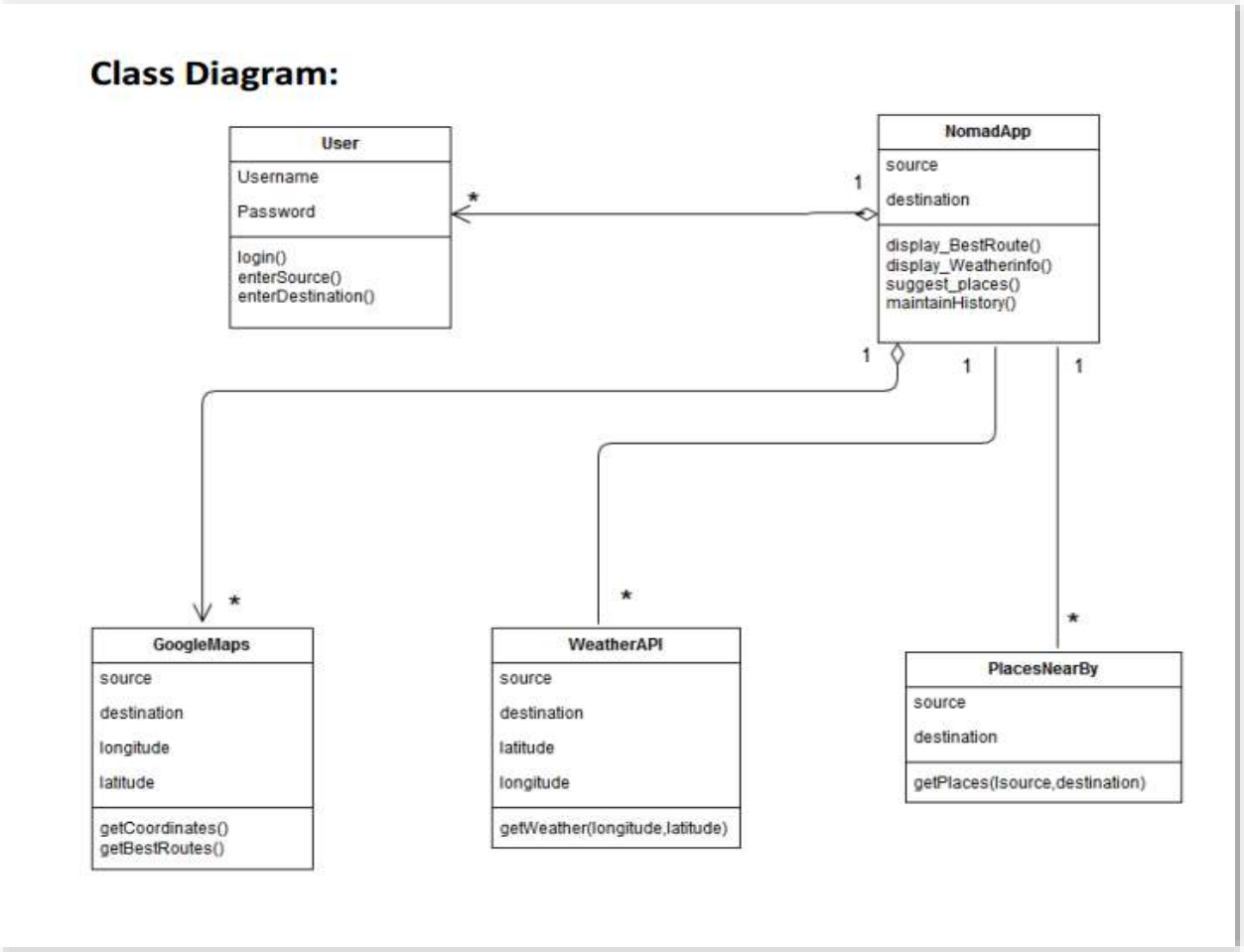
- **Caching of results:**
  - Will the response from the service be used by the client, or will the data be stored on the server? In Pet Store's mashup with the Yahoo Maps Geocoding service, the data provided by the service is not sent back to the client. If it did not store the results, it would have to make a request to the Geocoding service for each client request for a map and for each point shown on the map. This is a good reason for doing the mashup with the geocoding service as a server-side mashup rather than a client-side mashup.
  - Map data can change. For example, street addresses and street names change. So you might need a mechanism for updating data that might go out of date. It's relatively easy to write code on the server to reaccess the mashup service if you need to update the data cached from a service.
- The RSS feed in Pet Store is another example of data that has been cached from a service for reuse. After data is retrieved from the live RSS feed on the Java Blueprints project site, the server side of the Pet Store application parses the RSS XML document and converts it to JSON. It then caches the data. The cached data in JSON format is used to fulfill Ajax requests from all Pet Store clients for display in the news bar.

- **The need to modify the response:**
    - You must determine whether the server side of the application will need to modify the response from the service before the client can use it or whether the application can directly pass it to the client.
- **Ways to handle exceptions and errors:**
    - The GeoCoder proxy class handles exceptions that could occur when accessing the Yahoo Maps Geocoding service. In general, it's good practice to handle exceptions that could occur when accessing a service. Additionally, it's good practice to validate the input data to a service. For example, GeoCoder validates the address input string before sending it to the geocoding service. Doing this can help uncover error conditions before the call to another web site and it could help you respond more quickly to the user. For instance, you could prompt the user to correct the entry.
    - Also, web sites and public APIs used in mashups have very different mechanisms for responding to exception conditions. So your error-handling code can get convoluted. By using the server as a proxy to a service, you can shield the client JavaScript code from many of these details and keep it more loosely coupled. In this way, you can lessen the impact on your client code if the mashup service changes its API.

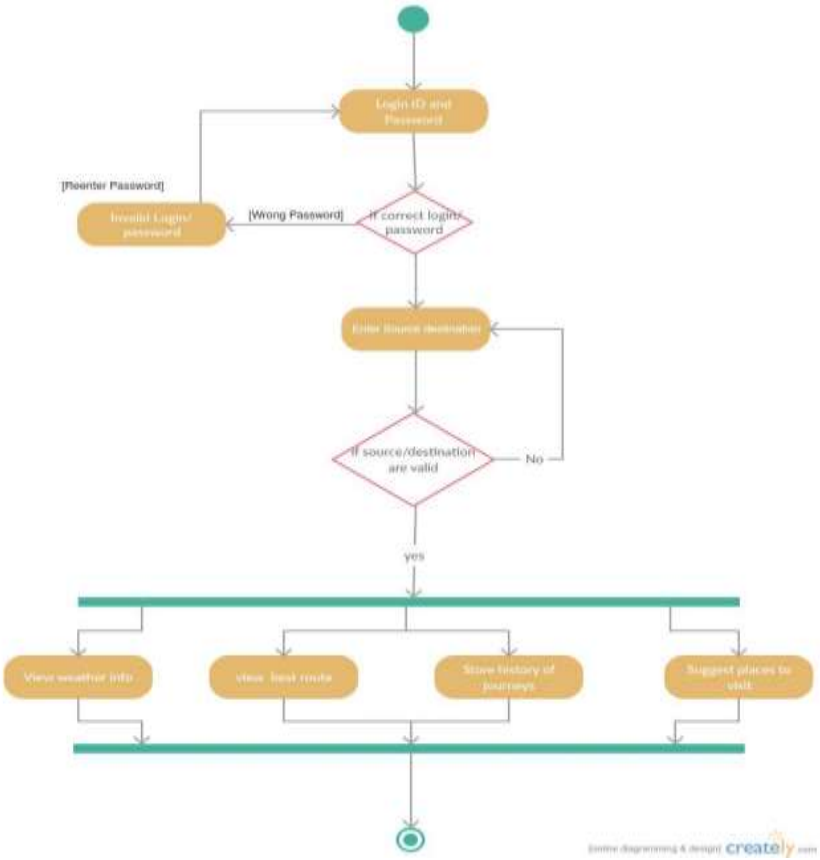
Use Case Diagram:



Class Diagram:

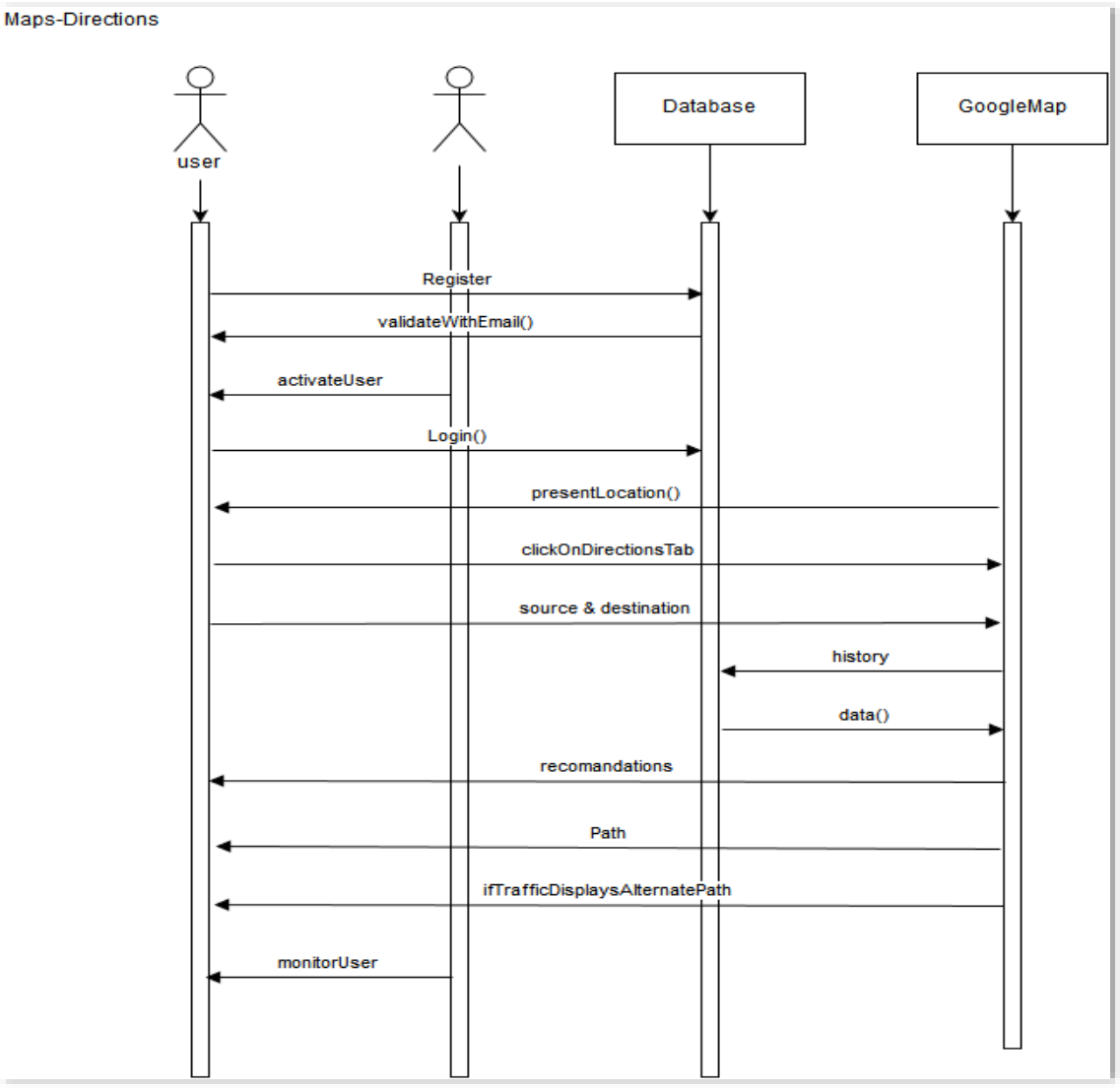


Activity Diagram:

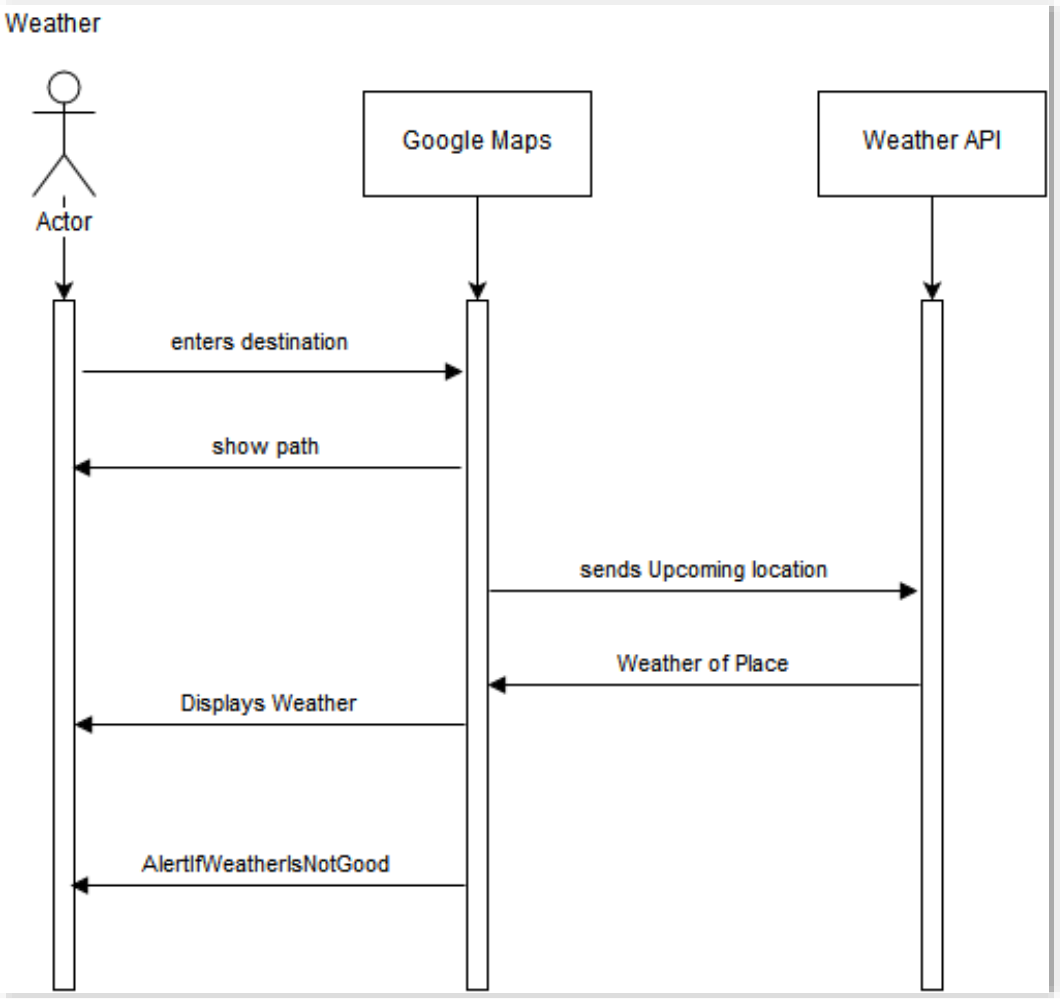


Sequence Diagram:

Best Route:



Weather:


Weather
Actor
Google Maps
Weather API
enters destination
show path
sends Upcoming location
Weather of Place
Displays Weather
AlertIfWeatherIsNotGood

Suggest Places nearby: