PROJECT 1: DESIGNING AND DEPLOYING A SERVICE-BASED DISTRIBUTED SYSTEMS

1. Problem statement: Design a distributed web application that provides users enhanced information about a travel route. Typical "map" applications provide the route, towns/cities along the route, alternate routes, and may be gas station locations. We want more than that. We want weather not only at the starting point and the destination, but also at the towns/cities along the way. Let's call this application MyWaypoints. Simply put, when a user inputs "From" and "To" locations on MyWaypoints, you will have to figure out the weather predicted for a given route and display it in a user-friendly form.
We also want to save all the user requests (user input) for downstream data analytics as well as for quick, efficient and cost-effective response to popular (and repeated) requests.

2. Learning outcomes:

1. Explain the structure of a distributed system.
2. Integrate and render data from diverse sources using the APIs of services offered
3. Explore the components, core technologies, architecture and protocols that enable a services-based distributed system.
4. Implement a web application stack.

2. Preparation before lab

1. Read and understand fundamentals of a client/server distributed systems.
2. Read and explore the components of a distributed web application stack.
3. Examine and choose a typical web stack has web-user interface (Web UI using HTML5, JS or any frameworks), web server (e.g. lite server such as node.js), a data store (e.g. mysql DB).
4. You may work on your laptop and must be able to communicate with CSE machines through ssh for final submission. Make sure you follow the instructions for preparing the submission file.

4. Problem Description

Build a multi-tiered distributed system comprising two major sub-systems
1. A **data access** system to access distributed data and
2. A **web application** that processes and serves the data collected through a simple user interface to pull a request from the user and output the results

The two sub-systems are *loosely coupled* via a database. The block diagram in Figure 1 shows the design of the system you will implement. Always begin by analyzing the problem and designing a technology-independent design diagram. Then think about the techniques, hardware and software that will solve the problem. This is a top-down design starting with the UI. We will now explain the various components in the design. We have numbered them for ease of reference and explanation.

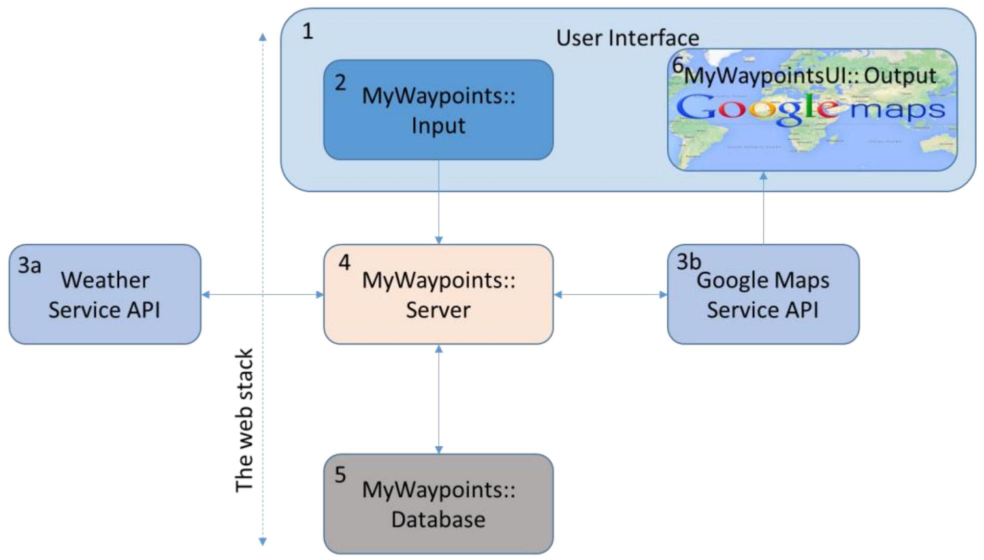

Figure 1: System Model of the MyWaypoints Distributed System

The User Interface (*box 1*) represents the complete UI (input request as well as results). Box 1 represents the input screen and Box 2 the display of the application output on a map.

Input user interface (*box 2*) is web interface to the user for obtaining the two ends of the route: starting and ending location information. You as a designer will have to decide the format of the input (visual, text, drop down, lat-long, zipcode..etc.). This is the landing page for your application. Make sure the user is drawn to your application by designing an intuitive and engaging user experience.

(*Box 3a, and 3b*) You are interested in the weather along the route or way you will be choosing. You will access the maps or routes API (*box 3b*) to obtain the 2 or more locations (cities or zip code along your route. Using this data, you will then access a weather services API to obtain the weather at these “waypoints”. The weather data should minimally have hi/lo temperature and sun conditions {sunny, cloudy, etc.} as it is usually shown by weather services. (Pause for few moments here: weather is just a metaphor for many other information you may (retrieve and) display such as “points of interest”, “historic district”, “scenic view”, etc.)

A web server (*box 4*) hosts the UI and receives the requests and initiates the calls to the map and weather services to obtain the necessary data, and processes them. Once the data for all the waypoints are ready, it sends them to be displayed by the interface (*box 6*). Maps API will have to be accessed to obtain to obtain the background for the displayed output. The route and the waypoint information are to be aesthetically placed on the map.

The persistent storage (*box 5*) stores all the data related user requests, the results from the API access to the maps API as well as weather API. Often these API accesses is not free. So you don’t want to hit the paid services APIs again and again for the same values. Moreover, this local database request-result may also help in quickly answering user requests thus improving efficiency and scalability. Also this may address connectivity issues to the remote APIs; in this case you will be accessing information from the database (cache).

Phase 1: (50%) Your task is to design and implement the complete services-based distributed system indicated by *boxes 1-6 except 5* in *Figure 1*, and explore the operation of this partial proof of concept (POC) system. Test it thoroughly. Call it version 1.

Phase 2: (25%) Let this be a separate fork. Save the code base of version 1. Make a new codebase, a copy of the version 1. Add the persistence layer in a database. You can choose a relational database such as MySQL or any other database you prefer. Test it thoroughly. Here repeated request should be answered from your data base and not by accessing the API. Call it version 2.

Phase 3: (5%) Compare the two implementations of versions 1 and 2. Assign cost functions c1, c2, and c3 to the API accesses and the DB access. Discuss the saving achieved by version 2. This can be done in the project report.

Good design: (10%) Discuss the choice of technologies and UI details in your report. Comment you code appropriately. Provide a detailed design representation like figure 1, customized to your design and technologies you used.

Directory structure and readme: (10%) Provide a readme file that provides details of how to run deploy and run your code. Feel free to include screen shots of a working application.

5. Project Implementation Details and Steps:

1. **Get used to building *client-server* systems**:
   When you implement a simple *client side* application program there are just two steps involved: compile and execute the code. In a *client-server* system, you will have to take care of the *server side* as well as the *client side*. On the *server side*, you will compile the code using special compilers, deploy the service, register and publicize the service for the clients to use. On the *client side* you will prepare the client code with appropriate keys and during execution lookup the service needed and use it.

2. **Working with APIs to distributed services:**
   This is an essential software development skill. Services of popular applications such as twitter are provisioned for use by external third-party application. While some of the services can be accessed without registration, a standard approach to retrieving the distributed data is by using APIs or application programming interfaces provided by the application providers themselves. You choose the service, register with them, and obtain the credentials that is usually in the form of a pair of cryptographically unique keys. These keys help programmatic access to the APIs offered by the services. We have list two representative services in the reference section, one for maps and other one for weather. You may decide to choose other services and APIs of your choice. Make sure you provide in the README the complete reference to the APIs you are using. Also keep the keys safe, do not send it to anybody, and remove it from the code you submit or store. It is like your identity card.

3. **Working with the *database:***
   In this project you will store the data in a relational table. Create a single table representation of the data. This application does NOT warrant a complex multi-table design; however, it does provide essential experience in designing a database if you are new to this area.

4. **Study, understand and choose the web application stack (indicated in figure 1):**
   One of the major design decisions is choosing your application stack. For example, is it LAMP (Linux, Apache (web server), MySQL, PHP) or MEAN (MongoDB, Express, Angular, Node,js), etc.? It does NOT have to be any one of these but a simple web server stack.

5. **Design, implement the distributed application in phases**
   You can choose any programming language of your choice. For the technologies, you may use anything appropriate (even R language suite will work!!!). Make sure you implement the Mywaypoints in phases and save each version. Do not overwrite. We want to see incremental development steps. This also lets to plan your time so that you do not wait till the last minute to mash something together to satisfy the requirements.
6. **Deploy the integrated system:**
   The various components listed above should have been deployed and tested individually. In this step you will run the entire integrated system. You can create a good test data that will completely test all the components of your system.

6. Project Deliverables:

- A standard directory structure for the two versions of the software developed.
- A readme files documenting all the details from APIs used to how to run the application and any special features. Don't forget to include screen shots of the application running.
- You may also submit other media such as a video of your application working (optional).

7. References:

1. Maps API: https://developers.google.com/maps/web-services/client-library
2. Weather API: https://openweathermap.org/api