CS 441 Project Requirements Specification

iGo

Group Members
Chetana Vedantam
Harry Cordewener
Kanishka Garg
Lokesh Gunda
Shouvik Sayef
Sudhanshu Malhotra

Project Domain

Our project will gather data related to travel, and our application functions will attempt to provide informative content based on the the user's travel destination. Our initial ideas for functionality include informing the user of relevant friends, restaurants, and hotels based on a location the user wants to learn more about using data from social media APIs. After we have a functional program with basic functions using dummy/randomly generated data, we can continue to add additional travel related data and functionality.

The central domain of the project will be creating a data-serving API based on simple variable argument requests, generating representational formats such as JSON to serve as an in-between for location-based content visualization.

Data Generation

In order to limit the size of our initial database we will generate random data for major cities in the U.S. fitting the functions described above, the database can be supplemented with data provided by social media APIs, we are currently planning to integrate Facebook and Yelp APIs.

Sample Object Types & DB Schema

NOTE: These will be modified as we add functionality to the program.
A unique 'id' is assumed for each object.

Object	Attributes
User	Name, Current Location
Friendship	User From, User To
Location to Visit	User, Location
Location	GPS Coordinates, City Name, State, Zip Code, Company
Company	Name, Rating, Type

Databases

SQL - MySQL
noSQL- MongoDB

Development Environment

Eclipse Juno + BPEL plugin for Apache ODE

JUnit Testing

Software Components

The project will consist of three main software components: the DB loader, the server, and the client. The DB loader will be run a single time to set up the database schema and populate the tables (for a fresh deployment of the full project). The server will handle remote object functionality, including storing objects, registration, and stubs, as well as communication with the database to store/generate objects. The client will act as the user interface, all functionality will make remote calls to the server and display returned data to the user.

Database Loader

The purpose of the loader will be to store the hard-coded schema of the databases for our application, and when given a connection to a database, either local or remote (to both MySQL or noSQL) it will first create tables according to our schema and populate them using our data source's API and/or randomized data. The idea is that the loader will be an independent program that is able to recreate our data on any system. A script will take up this task.

Input

- User will provide database info (URL, credentials, etc), and credentials for social media API (Facebook, Yelp, etc.)
- NOTE: social media credentials may be provided by client program so that loader will not require user input.

Functions

- Compatibility with both SQL and NoSQL must be considered
- Check the database connection
- Check to see if tables already exist in the database
- Create tables according to hardcoded schema if they are not already present
- Check for existence of data (using expected row count?)
- Populate tables in the specified database using source APIs
- Populate the tables in the database with randomized data which fit our usage cases

Output

- Confirmation of table creations and row insertions returned from the database
- Display any db errors that occur

Server

The server will receive function calls from the client requesting objects represented within the database. The server will act as a backend for the client; it will interact with the databases as

needed to gather the data, create the objects, register them within the server application, run methods remotely called by the client, and finally return the data to the client for display. The server will also take in modified/created objects from the client and modify the database, and register the objects as needed. It will be in charge of the translation of queries between SQL and NoSQL as well.

Input

- Connection information (i.e. port/address/connection method for the client to connect to.) to launch server.
- Credentials and function calls will be received from client.

Functions

- Query the database on behalf of the client's requests
- Process database query and create objects
- Register requested objects in Server
- Run any object functions as requested by client
- Modify database and register new objects if sent from client

Output

- Output database responses to console (Debug only)
- Inform the client of any networking/db errors if they occur
- Return references to objects and results of remote function calls to client as requested

Client

The purpose of the client is to act as the frontend where the user will input commands which correspond to functions, which are then sent to the server. The client will also have 2 options to indicate to the server which type to database to utilize (SQL vs noSQL). However, the server is tasked with using this switching argument to grab data. We may also implement some kind of metric to measure and compare client performance based on Mininet settings and database type. The client will handle formatting the data for display on the frontend.

Input

- Connection info for server.
- DB connection info to provide to server.
- DB type to use/ credentials.
- User will input functions to send server via frontend.
- Some functions will use user input to create objects to send to the server, these functions will trigger the server to modify the database.

Functions

- Communicate with server, server will be responsible for all processing/method calls of remote objects.
- After receiving remote objects, the client will convert into a standardized format for graphical display (i.e. JSON).

Output

- Display errors/confirmation/connection info from server.
- Display function results for user.
- Metrics and performance summary.
- Frontend will allow user to input needed data and send function calls to server.

Assumptions

- Existing databases accessible by server. (To be created/configured prior to loading).
- A reasonable 1:1 relation between NoSQL and MySQL schema, or a viable Adapter model that can handle the relationship.