Enterprise Distributed Systems

Peer Reviewed Class Project – Kayak.com Simulation (Fall 2017)

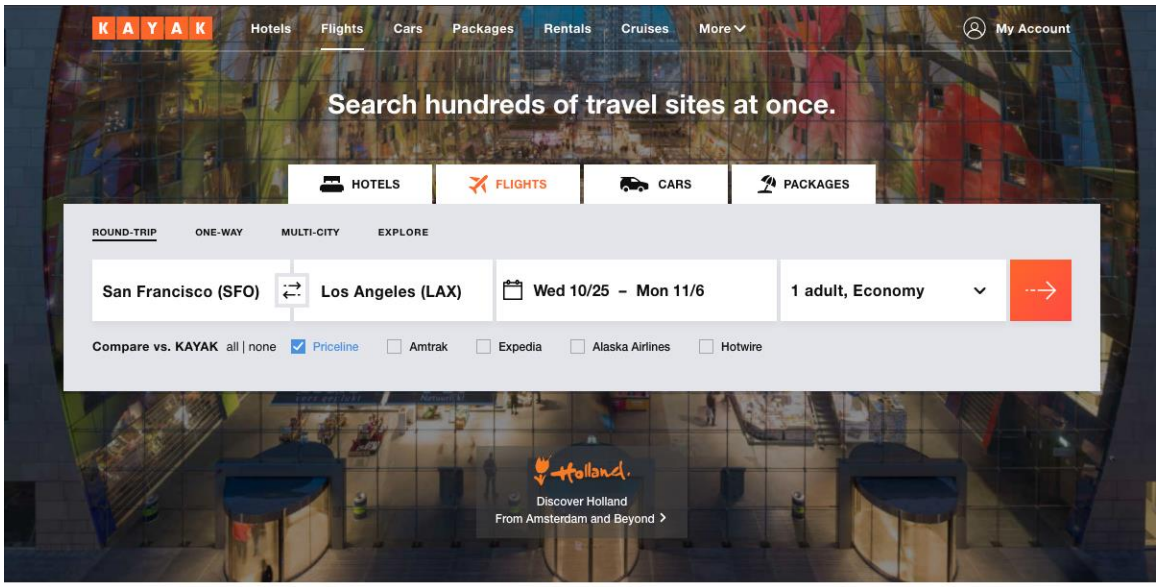
<u>Project Due Date</u>: 3rd December 2017
Demo: 12/4/2017
Presentation: 12/6, 12/11

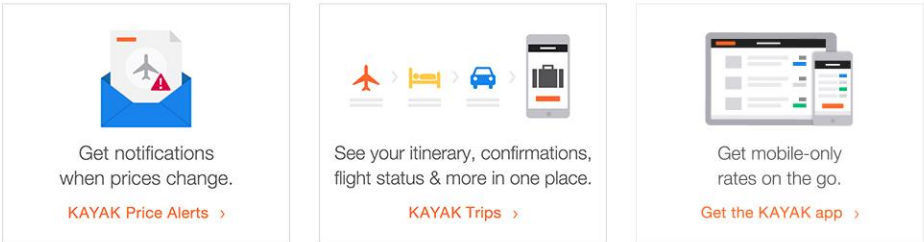
This is a group project up to 5 people in the team.
Turn in the following on or before November 8, 2017 on canvas. No late submissions will be accepted!

- An API design document of services provided by the application.

<u>Kayak</u>

KAYAK is a travel search engine. It looks across the web to find your options on flights, hotels and rental cars. It's all about giving traveler the information needed so they can choose the option that's right.





Project Overview

In this project, you will design a 3-tier application that will implement the functions of **Kayak** for different travel services. You will build on the techniques used in your lab assignments to create the system.

In Kayak prototype, you will manage and implement different types of objects:

- User
- Listings
- Billing
- Administrator

For each type of object, you will also need to implement an associated **database schema** that will be responsible for representing how a given object should be stored in a database.

Your system should be built upon some type of distributed architecture. You have to use message queues as the middleware technology. You will be given a great deal of "artistic liberty" in how you choose to implement the system. **These are the basic fields that needed. You can modify this schema according to you need and assign primary keys and foreign keys. Example: Create relation between User and Trips, Car, Flights, Hotels etc. Above mentioned are just examples.**

Your system must support the following types of entities:

- **User** – It represents information about an individual user registered on Kayak. You must manage the following information for this entity:
  - ⇒ User ID
  - ⇒ First Name
  - ⇒ Last Name
  - ⇒ Address
  - ⇒ City
  - ⇒ State
  - ⇒ Zip Code
  - ⇒ Phone number
  - ⇒ Email
  - ⇒ Trip ID
  - ⇒ Profile Image
  - ⇒ Credit Card details

- **Listings**

- **Hotels –** It represents the hotels listed by KAYAK.

⇒ Hotel ID
⇒ Name
⇒ Address
⇒ City
⇒ State
⇒ Zip Code
⇒ Stars
⇒ Rooms
  - Room Types
  - Prices

⇒ Ratings
⇒ Reviews

- **Flights –** It represents the Flights listed by KAYAK.

⇒ Flight ID (Flight Number e.g LX123)
⇒ Operator (Airline)
⇒ Departure/ Arrival Times
⇒ Origin
⇒ Destination
⇒ Classes (Eco, First, Business)
  - Prices

- **Cars –** It represents the Rental Cars listed by KAYAK.

⇒ Car ID
⇒ Car Type
⇒ Vehicle Specifications
⇒ Price (Daily Rental)

• **Billing** – It represents information about billing for bookings. You must manage the following state information for this entity:

⇒ Billing ID
⇒ Date
⇒ Booking Type (hotel/flight/car)
⇒ Total amount
⇒ User ID

• **Administrator** – It represents information about the administrator of the Kayak System. You must manage the following state information for this entity:

⇒ Admin ID
⇒ First Name
⇒ Last Name
⇒ Address

⇒ City
⇒ State
⇒ Zip Code
⇒ Phone number
⇒ Email

Project Requirements

Your project will consist of three tiers:
- The client tier, where the user will interact with your system
- The middle tier/middleware/messaging system, where the majority of the processing takes place
- The third tier, comprising a database to store the state of your entity objects

Tier 1 — Client Requirements

The client will be an application that allows a user to do the following:

- **User Module/Service:**
  - Create a new User (Reference Kayak websites for fields information)
  - Delete an existing User
  - Change a user's information (name, address, profile image etc) – *This function must support the ability to change ALL attributes*
  - Display information about a User.
  - Search listing for different categories
  - Filter listings
    - Filter hotels based on stars, price
    - Filter flights based on departure/arrival, price
    - Filter cars based on car type, price
  - Book a hotel/car/flight
  - Make Payment
  - View Past/Current/Future bookings

- **Admin Module/Service:**
  - Allow only the authorized (admin user) to access Admin Module
  - Add listing(hotel/flight/car) to the system.
  - Search for listing and edit it.
  - View/Modify user/customer account
  - Search for a Bill based on attributes (fetch all bills By date, By months)
  - Display information about a Bill.

- Sample Admin Analysis Report

Admin will have his dashboard. Which will have different types of graphs and charts for the statistics that needs to be displayed.
Graphs Criteria

1) Retrieve data from database and show first 10 properties with its revenue/year (Bar, Pie or any kind of graph)
2) City wise revenue/year (Bar, Pie or any kind of graph)
3) 10 hosts who sold maximum number of properties last month with its revenue

- Sample Host Analysis Report

Host will have his dashboard. Which will have different types of graphs and charts for the statistics that needs to be displayed. **Data will be fetched from logging.**
Graphs Criteria

1) Graph for clicks per page(Bar, Pie or any kind of graph)
2) Graph for Property click (Bar, Pie or any kind of graph)
3) Capture the area which is less seen.
4) Graph for reviews on properties (Data from database)
5) Trace diagram for tracking one user or a group of users (ex. Users from San Jose, CA)
6) Trace diagram for tracking bidding for an item

You may add any extra functionality you want (optional, not required), but if you do so, you must document and explain the additional features. You still must implement all the required features, however.

The client should have a pleasing look and be simple to understand. Error conditions and feedback to the user should be displayed in a status area on the user interface.

Ideally, you will use an IDE tool to create your GUI.

Tier 2 — Middleware

You will need to provide/design/enhance a middleware that can accomplish the above requirements. You have to implement it using REST based Web Services as Middleware technology. Next, decide on the Interface your service would be exposing. You should ensure that you appropriately handle error conditions/exceptions/failure states and return

meaningful information to your caller. Make sure you handle the specific failure cases described below.
ON OR BEFORE the date listed below, you must turn in a document describing the interface your server will use which precisely means that you have to give API request-response descriptions.

Your project should also include a model class that uses standard modules to access your database. Your entity objects will use the data object to select/insert/update the data in the relational database.

User Kafka as a messaging platform for communication between front-end channels with backend systems.

Tier 3 — Database Schema and Database Creation

You will need to design a database table that will store your relational data. Choose column names and types that are appropriate for the type of data you are using. You will also need a simple program that creates your database and its table. The MySQL Workbench is a very efficient and easy tool for it.

You will need to decide which data are stored in MongoDB and MySQL. (Hint: think about pros and cons of MongoDB vs MySQL)

Scalability, Performance and Reliability

The hallmark of any good object container is scalability. A highly scalable system will not experience degraded performance or excessive resource consumption when managing large numbers of objects, nor when processing large numbers of simultaneous requests. You need to make sure that your system can handle many listings, users and incoming requests.

Pay careful attention to how you manage “expensive” resources like database connections.

Your system should easily be able to manage 10,000 listings, 10,000 Users and 100,000 Reservation records. Consider these numbers as **minimum** requirements.

Further, for all operations defined above, you need to ensure that if a particular operation fails, your system is left in a consistent state. Consider this requirement carefully as you design your project as some operations may involve multiple database operations. You may need to make use of transactions to model these operations and roll back the database in case of failure.

Testing

To test the robustness of your system, you need to design a test harness that will exercise all the functions that a regular client would use. This test harness is typically a command-line program. You can use your test harness to evaluate scalability (described above) by having the test harness create thousands of listings and users. Use your test harness to debug problems in the server implementation before writing your GUI.

Other Project Details

Turn in the following on or before the due date. No late submissions will be accepted!
· A title page listing the members of your group
· A page listing how each member of the group contributed to the project (keep this short, one paragraph per group member is sufficient)
· A short (5 pages max) write-up describing:
  a) Your object management policy
  b) How you handle "heavyweight" resources
  c) The policy you used to decide when to write data into the database
· A screen capture of your client application, showing some actual data
· A code listing of your client application
· A code listing of your server implementations for the entity objects
· A code listing of your server implementation for the session object
· A code listing of your main server code
· A code listing of your database access class
· A code listing of your test class
· Any other code listing (utility classes, etc)
· Output from your test class (if applicable)
· A code listing of your database creation class (or script)
· A screen capture showing your database schema
· Observations and lessons learned (1 page max)

The possible project points are broken down as follows:
(One of grading criteria is subjective and relative strength and weakness of your project compared to other projects)
• **40% for basic operation** – Your server implementation will be tested for proper operation of some aspect of your project. Each passed test is worth some point(s) toward the total score, depending on the test. Each failed test receives 0 points.
• **15% for scalability and robustness** – Your project will be judged on its scalability and robustness. I will test your program with thousands of objects; it should not exhibit sluggish performance, and it definitely should not crash. In addition to performance improvement techniques cover in the class, you are required to implement **SQL caching** using Radis and show performance analysis.
• **10% for distributed services** – Divide client requirements into distributed services. Each service will be running on backend connected by Kafka and divide your data into MongoDB and MySQL and provide performance data with justification on results.

• **15% for Analysis report, web/user/item tracking** – Devise your own web pages/user/item tracking using logs and explain why it is effective to analyze a web site and user behavior.
• **10% for the client** – You will get more points if your GUI is similar to kayak site.
• **10%** for your test class and project write-up

Hints:

· Maintain a pool of DB connections – Remember that opening a database connection is a resource-intensive task. It would be advisable to pre-connect several database connections and make use of them throughout your database access class so you don't continually connect/disconnect when accessing your database.

· Cache entity lookups – To prevent a costly trip to the database, you can cache the state of entity objects in a local in-memory cache. If a request is made to retrieve the state of an object whose state is in the cache, you can reconstitute the object without checking the database. Be careful that you invalidate the contents of the cache when an object's state is changed. In particular, you are required to implement **SQL caching using Radis**.

· Don't write data to the database that didn't change since it was last read.

· Focus FIRST on implementing a complete project – remember that a complete implementation that passes all of the tests is worth as much as the performance and scalability part of the project.

· Do not over-optimize. Project groups in the past that have tried to implement complex Optimizations usually failed to complete their implementations.

Exceptions/Failure Modes

Your project MUST properly handle the following failure conditions
- Creating Duplicate User
- Addresses (for Person) that are not formed properly (see below)

For more failure conditions see Other Notes below

Other Notes

State abbreviation parameters

State abbreviations should be limited to valid US state abbreviations or full state names. A Google search will yield these valid values. Methods accepting state abbreviations as parameters are required to raise the 'malformed_state' exception (or equivalent) if the parameter does not match one of the accepted parameters. You do not need to handle US territories such as the Virgin Islands, Puerto Rico, Guam, etc.

Zip code parameters

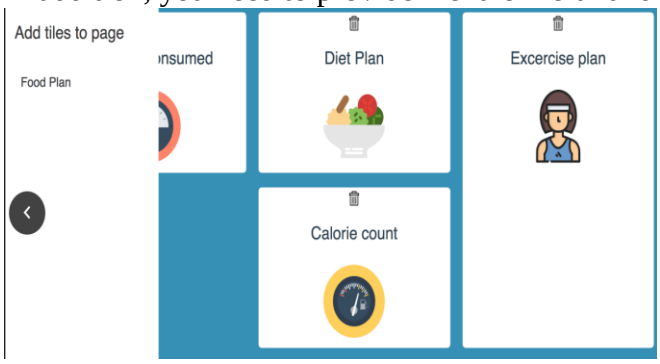
Zip codes should adhere to the following pattern:
[0-9][0-9][0-9][0-9][0-9]
or
[0-9][0-9][0-9][0-9][0-9]-[0-9][0-9][0-9][0-9]
Examples of valid Zip codes:
95123
95192
10293
90086-1929
Examples of invalid Zip codes:
1247
1829A
37849-392
2374-2384

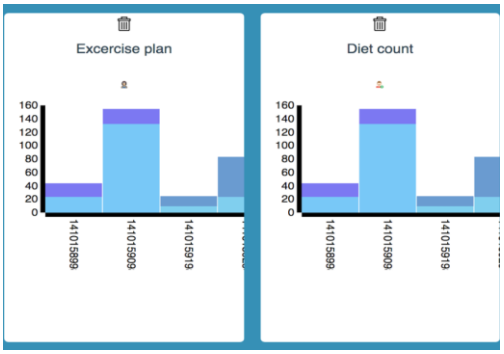
Peer Review

Each team member should write review about other team members except himself/herself. It is confidential which means that you should not share your comments with other members. Peer Review is submitted separately on Canvas.

Extra Credit (10pts)

There will be extra points allocated for creating tile based GUI for admin application.
https://strml.github.io/react-grid-layout/examples/0-showcase.html
In addition, you need to provide menu on left hand side and delete button for each tile.





There is no partial credit for Extra credit. You will get either full marks or none based on your mobile application quality.