Lunch With Friends: Complete Documentation

By: Amanda Zhang, Amir Hegazy, Brenna Chen, Jaemyung Choi, Soumika Guduru

Table of Contents

Project Proposal

For our final project, we plan on implementing a web application called Lunch with Friends that is similar to Tinder but matches users to certain events and restaurants they would like to visit. The main purpose of the app is for users to express their interest in visiting a certain restaurant or event and then be matched with other users with the same preferences. This way, users can meet other individuals interested in trying out a particular restaurant or going to a certain event. In terms of location, we will be focusing on restaurants and locations within range of USC to narrow our scope.

For the assignment specifications, authenticated users will be able to view other users and mark their interest in locations. Guests will be able to view different locations but will not be able to mark their interest or be matched with other users. Network functionality and multithreading will come into play when 1) matching users to each other based on profile and location and 2) as the number of users interested in a certain restaurant/event is updated globally for other users to see.

~~Some other miscellaneous features include a visual indicator for users who have responded yes and an expiration date/time for suggested connections. We will also allow for user profile pictures, general user location information, friending features, and privacy functions if users wish to remain anonymous to guests similarly to other popular social media platforms.~~ In terms of gathering information on the restaurants and event locations, we expect to use Yelp or Google API.

Technical Specifications

Front-end

- Landing page
- Create login ~~screen~~ button and pop-up
    - Create Account / Log In / Browse as Guest
        - Username, password, ~~profile pic~~
        - ~~Enter user bio, interests, etc.~~
        - Prevent duplicate accounts
    - ~~Location tracking accept button (if enough time)~~
- Create main screen (mimics Yelp)
    - Map on right
    - Settings / Account / Chat buttons top right
        - ~~Settings features~~
            - ~~Change name/email/password/location~~
            - ~~Log out~~
    - Search bar top center
        - Displays list of near-by restaurants based on search - connects to back-end
        - ~~Also displays how many people are interested in going to the restaurant (updated live/globally)~~
    - ~~Notifications on left side - matchings made (Yes / No options)~~
- ~~Connect to back-end using Spring framework~~

Back-end

- Get list of restaurants based on search result
    - Yelp API call
- Create map with pins on different restaurants nearby using restaurant locations
        - Google Maps API
        - Google Geolocation API
        - Google Geocoding API
- Figure out log-in functionality
    - Central server to authenticate
    - Database for user registration information

- User suggestions
    - Group option and single option
    - Groups get access to a group chat to plan on visiting a certain restaurant together
    - Single users get suggested other user profiles, can say yes or no
- Chat feature for people who are suggested to each other
    - Group chat option as well
- ~~Profile Info~~
    - ~~Location data~~
    - ~~Store info using spring mvc form handler~~
    - ~~Groups of two or more preferred?~~
    - ~~Name + Profile Pic~~
    - ~~Bio/interests~~
- Multithreading
    - ~~While chatting + ? - update # of people waiting to be matched for restaurants in the background - multithreading?~~
    - Apache Tomcat
    - JSP + servlets
    - Synchronized Java Collections
- Networking
    - Matching users to each other simultaneously
    - Apache Tomcat
    - Server Endpoint + Servlets

Resources:

- Yelp API:
    - https://rapidapi.com/blog/yelp-fusion-api-profile-pull-local-business-data/
- Google Maps API:
    - https://developers.google.com/maps/documentation/geocoding/overview?hl=en
    - https://developers.google.com/maps/documentation/maps-static/overview?hl=iw&csw=1

Detailed Design Document

Overview:

- Hardware
    - PC / Laptop
- Software
    - Eclipse, VSCode, etc. for programmers to write the code
    - MySQL to store user and event data
- Languages
    - Java to communicate with the SQL server
    - SQL to write instructions for the MySQL server
    - HTML, CSS, Javascript, and JSP for the front-end

Task Breakdown

Amanda, Brenna - display experience, restaurant data storage, connecting to servlets

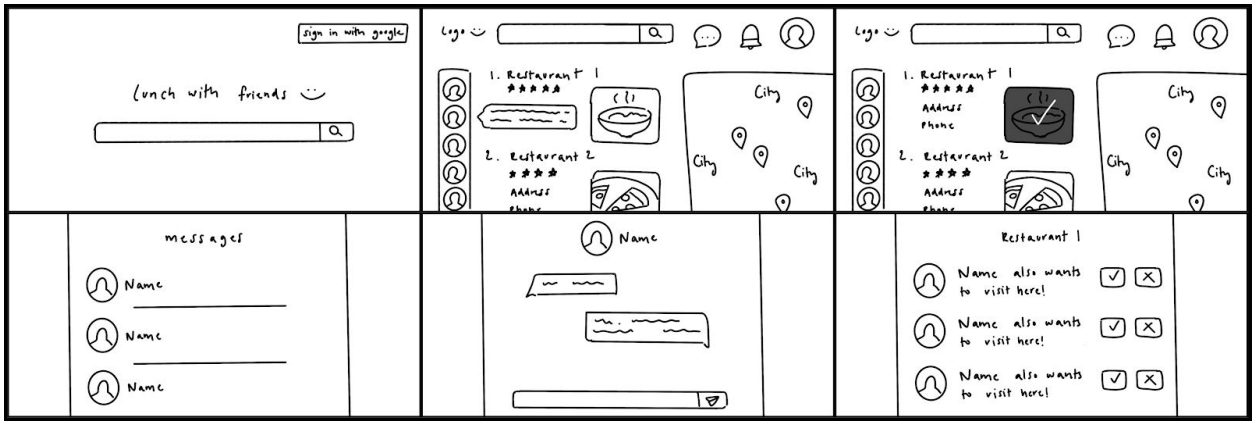
Amir, Jaemyung - matching/chat features, communication between parts

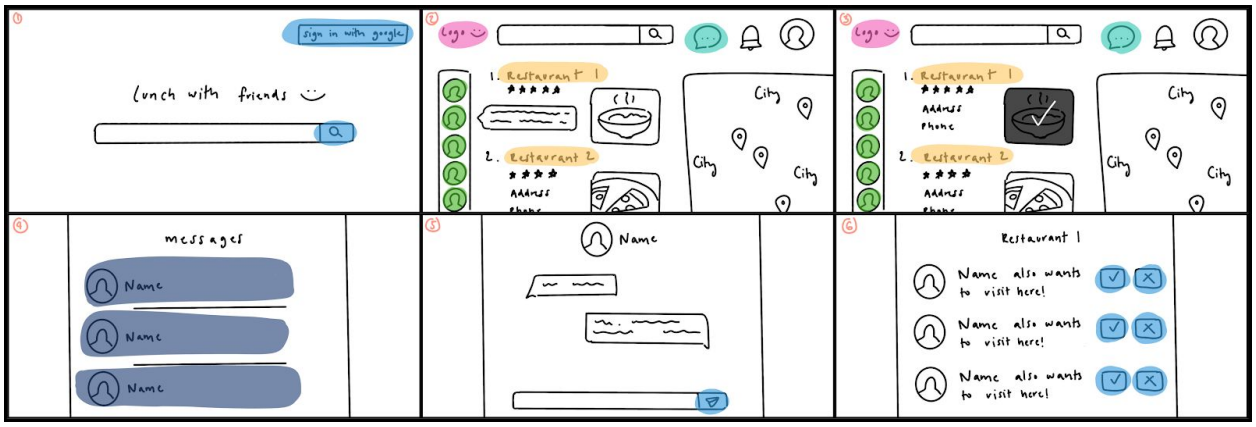
Mika - user login/data storage, differentiate registered/non-registered users

- User Interface & Front-end (30 hours)
    - Landing/search page
    - Restaurants page
    - Map
    - Login with Google API, also integration with back-end and database info (8 hours)
    - Guest functionality is limited to just searching restaurants
- Back-end Logic in Java (40 hours)
    - Implement function for matching users
    - Implement function for searching restaurants
        - Parsing JSON from Yelp API
    - Integration from back-end to front-end to send notifications to users (5 hours)
    - Chat feature, also integrates with front-end (8 hours)
- Database (30 hours)
    - Stores users and basic user info

- Stores restaurants and interested users
- Pipeline to access database info from front-end or back-end (5 hours)

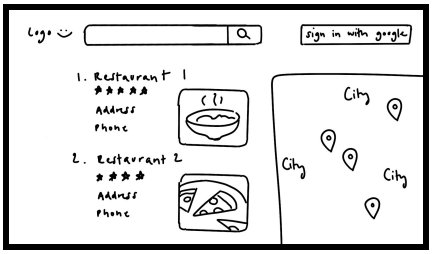
GUI Mockup

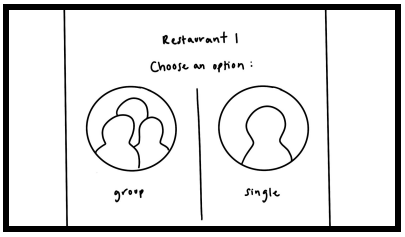




Clarifications:

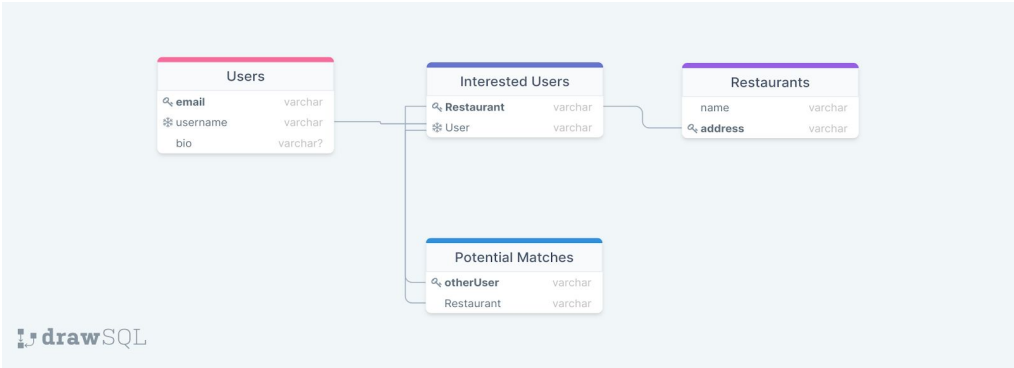
- Colored areas are clickable buttons. Notifications button will likely show that the user has been matched with someone.
- Guests will only be able to see the first two screens, the landing/search page and the list of restaurants (and probably a number or rating of how popular the restaurant is). They will not be able to have notifications, messages, or be able to express interest in a restaurant. The second screen will be slightly different, depicted here:

- 

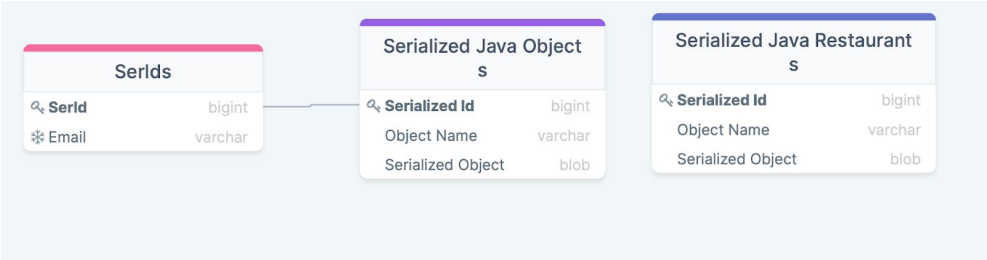
- Login will be completely handled by Google API.
- Clicking the logo will direct the user back to the landing/search screen.
- Clicking the restaurant name will lead to an options page (depicted below) and then the matches page.
  - 

Database Schema

Old:



Current:



Class Details

Class	Data Members	Methods	Overview
User	- string username - string email: will verify if email is suitable through Google API - HashMap<Business, List<User>> potentialMatches - HashMap<Business, List<User>> potentialInterestedMatches	- getters and setters for each var - addToPotentialMatches(User newUser) - addToPotentialInterestedMatches(User otherUser) - May open chat window if User is also in otherUser's potentialInterestedMatches	Contains basic information of user HashMap potentialMatches links each list of possible matches (Y/N option) to each restaurant the User is interested in HashMap potentialInterestedMatches links the restaurant to each list of users the User has said yes to for that restaurant - If the User is also in the other user's potentialInterestedMatches for that restaurant, opens a chat window
Login	- Sign in with Google	- Authenticate (username, password): uses Google sign-in to sign-in the user - creates User upon first login	Needed for login page - authenticates user - Database of users to check for first time logins

Business	- Double rating - String price - Boolean is_closed - String name - String url - Coordinates coordinates - Address location - ArrayList<User> interestedUsers; - ArrayList<User> groupUsers;	- Constructor - Getter & setter methods - likesRestaurant(User, boolean): adds User to interestedUsers or groupUsers - hasDecidedOnRestaurant(User): returns if user has decided on restaurant	Contains information for business - Contains ArrayList interestedUsers, which we'll use to keep track of all users that show interested for the restaurant who don't want a group - Contains List groupUsers, which we'll use to keep track of all users who want a group
YelpAPIParser		- List<Business> getBusiness(String term, Location location): gets list of businesses from Yelp API	
UserDBAccess		- Long serializeJavaObjectToDB(Connection connection, Object objectToSerialize) - Object deserializeJavaObjectFromDB(Connection connection, long serialized_id)	- Serializes and deserializes Java objects to/from database

SearchRestaurantDisplay JDBC_Access JDBCUser_Access businessDBAccess DisplayInterestedUsers ChatServlet		- doGet(HttpServletRequest request, HttpServletResponse response)	- Servlet implementation classes to access database for businesses & users & chat
ChatRoomServer	- SynchronizedMap <String, Set<Session>>	- open(Session session, String userID, String room) - receivedMessage(String message, Session session) - connectionClosed(Session session) - String makeText(String userID, String msg) - Set<Session> getChat(String roomName)	- Creates chat room, sends/receives messages between users

Clarifications:

- If a user chooses the group option, then they get access to a group chat of everyone else who chose the group option (not split into smaller groups).

- Each user has a hashmap of the restaurants and the users they'd be interested in going with. To check for matches, we have to check if the current user is on the other user's hashmap.
    - All users (matched or not) who are interested in the restaurant are located in the ArrayList.

Testing Document

Log-in

1. Purpose of test: Makes sure account information is stored once a new email logs in
   Input: User inputs email address, username, and password
   Output: User is created, with correct info stored
2. Purpose of test: Check that logging in takes user to the correct account
   Input: User inputs email address and password
   Output: User is logged to their account
3. Purpose of test: Verify that there are no duplicate accounts
   Input: Existing user logs in
   Output: User only appears once in database

Yelp API & Google Maps API

1. Purpose of test: Makes sure list of restaurants is pulled from Yelp API
   Input: N/A
   Output: Restaurants successfully added to SQL database w/o duplicates
2. Purpose of test: List of restaurants changes based on coordinates
   Input: Manually inputted coordinates
   Output: Different restaurants added to database based on coordinates
3. Purpose of test: Makes sure Google Maps API displays map based on user location
   Input: User location
   Output: Map displayed centered on user's location

Matching process (single)

1. Purpose of test: Makes sure there is no time when two mutually interested users are not matched
   Input: One user A says yes to user B who has already said yes to A
   Output: Chat is created
2. Purpose of test: Makes sure no chat is created between non-mutually interested users
   Input: One user A says yes to user B who has said no to B
   Output: Chat is not created

Matching process (group)

1. Purpose of test: Check if newUser gets added to List<User> groupUsers in Restaurant class
   Input: newUser clicks Join → Group
   Output: successful addition of newUser to List<User> groupUsers
2. Purpose of test: Check if newUser is added to the group chat
   Input: newUser clicks Join → Group
   Output: newUser added to chat

Database

1. Purpose of test: Businesses stored in database properly
   Input: Business class
   Output: MySQL server correctly receives and stores business
2. Purpose of test: Users stored in database correctly
   Input: User class
   Output: MySQL server correctly receives and stores users
3. Purpose of test: Businesses can be read from database properly
   Input: Business class
   Output: correct data/variables from Business (name, location, etc)
4. Purpose of test: Users can be read from database correctly
   Input: User class
   Output: correct User information

User

1. Purpose of test: Verify that User business interests are updated correctly
   Input: User clicks interested for a business
   Output: boolean isInterestedinBusiness returns true, User is added to list of users
2. Purpose of test:Verify that User decisions are accurately processed
   Input: User decides on a match for a certain restaurant
   Output: hasDecidedOn returns the correct boolean
3. Purpose of test: Verify that Interested Users the User dislikes are removed from their potential matches
   Input: User selects no for an Interested User
   Output: Interested User is removed from potentialMatches

4. Purpose of test: Verify that User's preference for group or single is stored correctly
   Input: User selects group or single for a particular restaurant
   Output: The corresponding boolean in the HashMap groupOrSingle is updated accordingly

Chat Room Server

1. Purpose of test: Verify that chatroom connects correct pair of users
   Input: Match is made between two users
   Output: Chatroom is created with the correct two users
2. Purpose of test: Verify that message is received by intended user when sent
   Input: User sends message to match
   Output: Match receives message

Deployment Document / Instructions on How to Run

1) Import the project into Eclipse

2) Right click Dynamic Web Project → Run Configurations → Tomcat

Environment: New Environment Variable

Name: GOOGLE_APPLICATION_CREDENTIALS

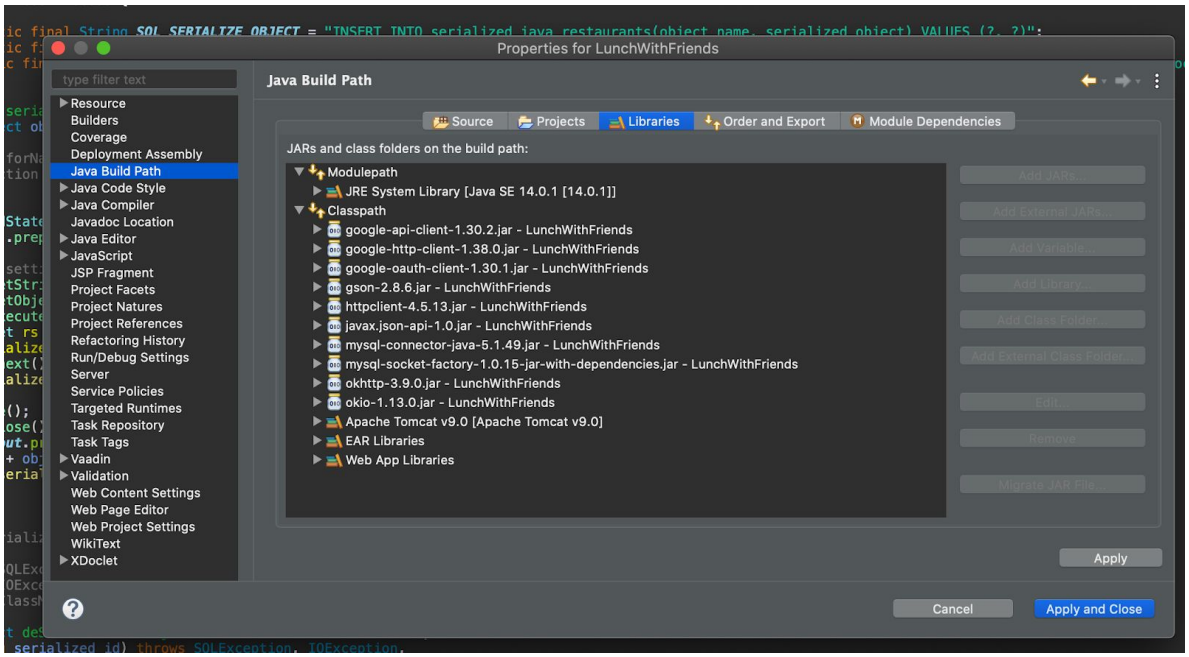
Value: Hardcoded - path to json file (Drag

"CSCI201-LunchWithFriends-7cac0ee5ef7f.json" file into terminal and copy statement into value

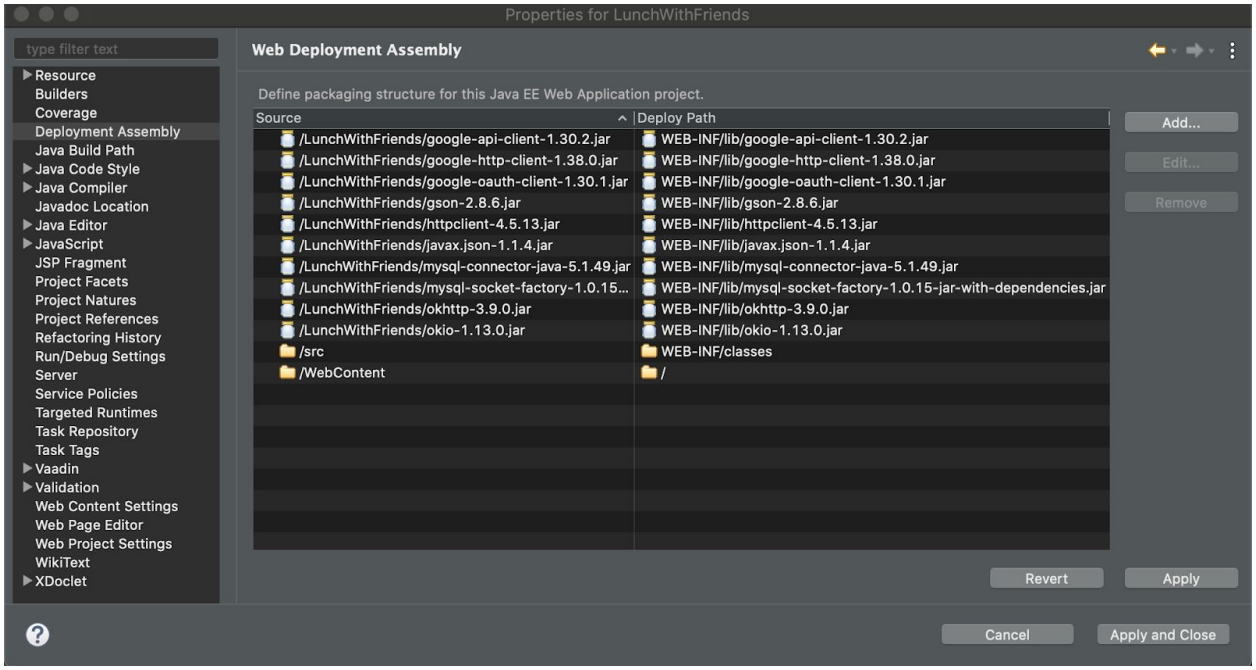
3) With Apache Tomcat, launch the web application by running index.jsp on the server.

4) If it does not run:

1. Right click project "LunchWithFriends" then click properties
2. Java Build Path: Although the files should already be there, Make sure selected jars are added in build path then Press "Apply" (to add press "Add Jar" on right side)

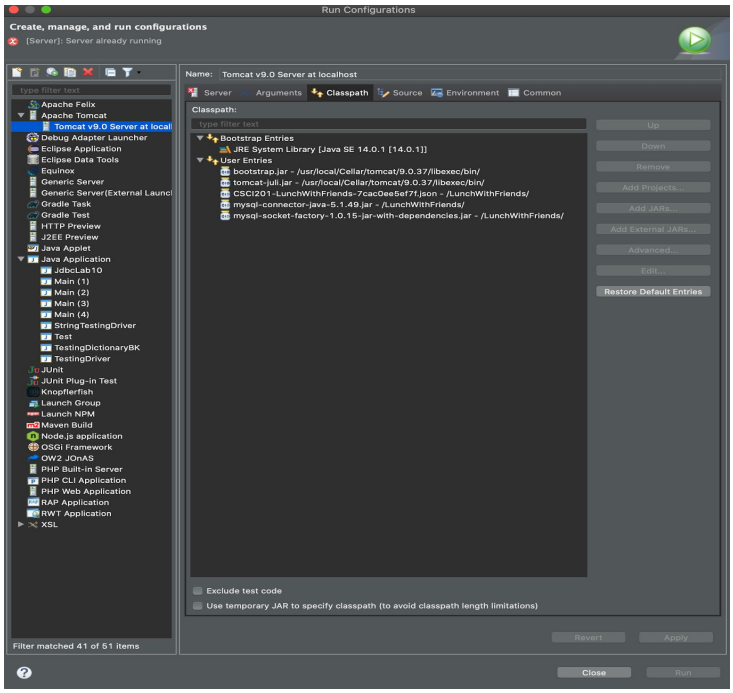


3. Web Deployment Assembly: Make sure selected jars are added in development assembly then Press "Apply and Close"



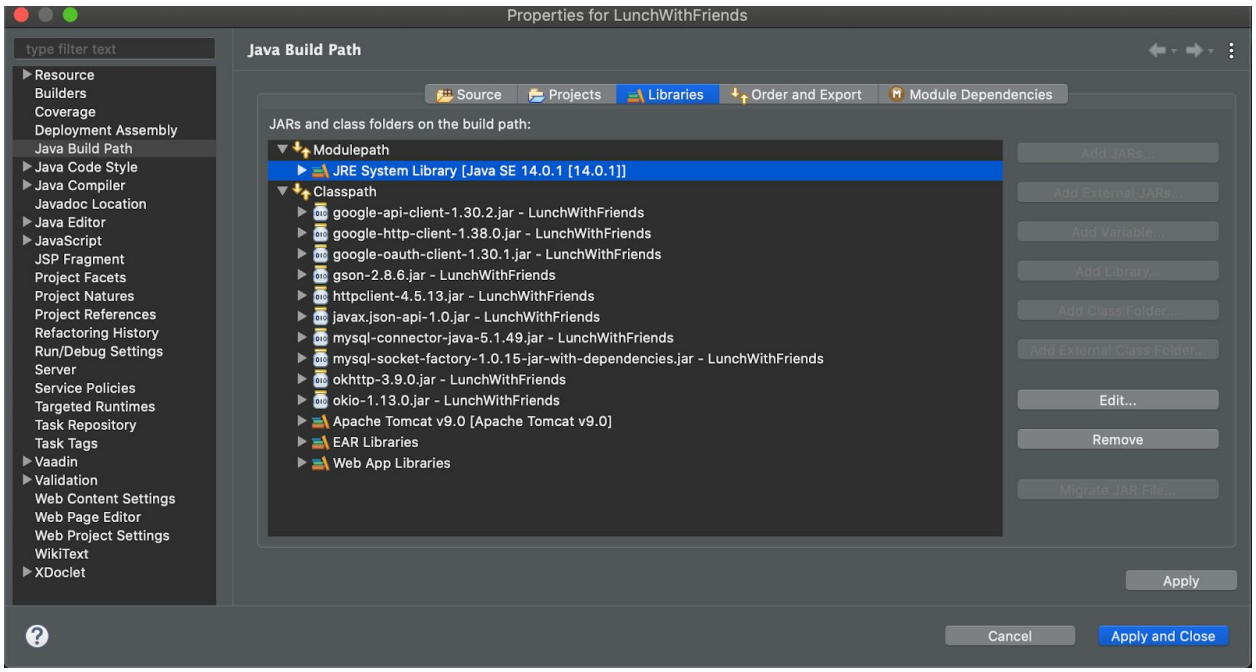
4. Press “App

5. Right click project “LunchWithFriends” then hover over “run as” and then click “run configurations” then “Classpath” and add the bottom three jars files with “Add jars” on the right



6. Extra step:

If server is unbounded Right click project “LunchWithFriends” then hover over “build path” and then click “build path configurations” and edit the blue highlighted button. Selected alternate JRE and click any of the ones from the dropdown..



7) run index.jsp !!!