Project Part 3: Bringing it all Together
Due: May 10th, 2019
150 Points

Purpose:
We are going to start building out a front end framework that will be grown upon in future labs and projects. We are going to base our project on an old game called “Oregon Trail” that was released in the 70’s. Here is a link to play the game online:
https://classicreload.com/oregon-trail.html

The last part of the project will now bring together the first 2 parts, applying our knowledge of asynchronous client-server communication. We will not only attach the front end UI to the back end game controls we created, we will also build out the setup screens and controller as an example of how you can build SPA (Single Page Application) using asynchronous client server communication. We will also add a database connection to our application and learn how to map our in memory data objects to be saved in persistence. You will need to add a customization to the game, a major function of your own choosing, and finally review the application for any missed or broken pieces.

Preparatory:
You will need everything working from our previous labs, and also you should have finished the first 2 parts of the project.

Submission:
1. Submit the iLearn lab assignment with a link to your git repository. You do not need to create a release version unless you plan on continuing to work on the code after the due date. In that case, create a tag / release for the project.
2. In ilearn tell me about your customization so that i know what to look for when grading it.

Summary of **Major** Pieces:

1. Build out the setup piece of the game, in which you define the player names, choose your profession and pick the month you wish to start
2. Create asynchronous calls between the client and server sides of your application for the following:
    a. Your game APIs (update, reset, setPace)
    b. Your game setup (get screen, save player names, save profession, saveStartMonth)
3. Create a database and connect your node server side application to it.
    a. Create a table in your database to save your top ten list.

    b. Write logic to both convert your in-memory top ten list into your database table and to retrieve the data from the table to populate your in-memory array of top ten scores.
4. A customization to the software that is all your own
5. Clean up, over all polish.

Part 1: Buildout your Game Setup

The setup controller on the server side should already provide methods to accomplish the following goals but you may need to add some data, particularly the 5 snippets of HTML that will be sent to the client side to show to the user

1. Choose your profession: Choosing a profession affects the amount of money your user starts with, you can set the amounts, I used what is in () below next to the profession. Neither the profession or starting money has any game effect.
    a. Banker (2000)
    b. Carpenter (1800)
    c. Farmer (1500)
2. Choose your wagon leaders name
3. Choose names for the 4 other members of the wagon party
4. Choose your start month (no game effect)
    a. March
    b. April
    c. May
    d. June
    e. July
5. Summary screen that shows the results

Your setup.js file on the client side will need to have **fetch api** calls to accomplish the following:

1. Retrieve the HTML for the setup page the user is on (call getSetupScreen api)

    Here is an example of what the 1st element of the setupScreens array might look like, containing the HTML for the choose profession screen.

```
var startGame1 = "<p>Many kinds of people made the trip trip to Oregon.</p>"
  + "<p>You may:</p>"
  + "<ol id=\"setupQuestions1\" >"
  + "<li id=\"bankerMenuItem\">Be a banker from Boston</li>"
  + "<li id=\"carpenterMenuItem\">Be a carpenter from Ohio</li>"
  + "<li id=\"farmerMenuItem\">Be a farmer from Illinois</li>"
  + "<li id=\"differencesMenuItem\">Find out the differences between the
choices</li>"
  + "</ol>"
  + "<div id=\"selectedOption\">What is your choice?</div>";
```

For this piece you want to create a RESTful API that allows you to request one of the snippets from the front end and get it back as HTML not JSON! Make sure that the MIME type for the RESTful API is returning HTML!

Once the UI receives back the response with the snippet, you will use DOM manipulation to "insert" the snippet onto the page.

2. Save chosen player names to your gameData object (call savePlayerName api)
3. Save chosen profession to your gameData object (call saveProfession api)
4. Save chosen start month to your gameData object (call saveStartMonth api)

All of this information should be stored in your gameData object for use during the game. All of the information sent from the save APIs should be sent in a POST request and the data to save should be sent as JSON. Note: You may choose to reference id numbers with the URL of the API call, but the value to save must be in the post. For example, for savePlayerName, you may do the following:
GET /api/savePlayerName/:id
With a JSON string like the following {"playerName": "Brian"}

Or
GET /api/savePlayerName
With a JSON string like the following {"playerId": 0, "playerName": "Brian"}

Please see the "Modern Ajax" slides for an example and code to start you for both getting a html snippet for the page and saving a value (profession) back to the server side.

Part 2: Create Asynchronous Calls for Game Play

Now you need to tie together the trail.js (or game.js) client side with the RESTful APIs for the game. This will be the last major part to get your game up and working!

You should have backend APIs at a minimum to handle the following requests from the front end, you may have created more for your application:

1. getGameData
2. updateGame
3. resetGame
4. setPace

Create fetch API methods in your front end game.js file to make asynchronous requests to your RESTful APIs and then update your game screen with the relevant information, using DOM manipulation. All data sent should be as JSON.

Part 3: Database

Your game will save your top 10 list to a mysql database. It will also read the top 10 list from the database and update it when there are changes. We will be discussion setting up local database installations and connecting via node in class.

Your topTen list will now be saved to a mysql database. To do this you will need to:

- Install mysql on your local machine
- Create a database and table to store the topten list in
- Connect to your database using node
    - Link: https://www.w3schools.com/nodejs/nodejs_mysql.asp
- Create queries to get all topscores from the database and save new ones.
    - Note: You table can store all top scores regardless of whether or not they are currently in the “top 10”. The controller logic must only return the 10 highest to the UI when the /game/topten GET api is queried, but all values can persist in the database.
- Make sure to include the following sql statements in your project submission:
    - Create database
    - Create table

Part 4: Customization

You must add your own significant feature to your game. Document the basics of what is and how it works and submit this information when you turn the project in within iLearn.
Examples:

- You add food to the game, uses consume some amount of food everyday (you should be able to set rations), running out of food has detrimental effect on health, extra large rations have a positive effect on health.
- You add random landmarks along the path which allow the player to interact (talk to other people, cross a river, etc)

Part 5: Overall Polish

- Find any loose ends on either the front end or back end and fix them!

- Remember to review for comments and clarity of code
- Test everything!

Grading breakdown:

- Setup.js client side buildout
- Fetch API / Promise (ajax) calls!
- Your game should work!
- Your setup for the game should work!
- Does the database store / retrieve the topTen list?
- Customization works and complexity / value to game
- Code it like you are proud of it! Coding style will count towards grade!