- Frontend - Trivia API
    - Getting Setup
        - Installing Dependencies
    - Required Tasks
        - Running Your Frontend in Dev Mode
        - Request Formatting
        - Optional: Updating Endpoints and API behavior
        - Optional: Styling
        - Optional: Game Play Mechanics
    - DO NOT PROCEED: ENDPOINT SPOILERS
        - Expected endpoints and behaviors

Frontend - Trivia API

Getting Setup

> *tip*: this frontend is designed to work with Flask-based Backend so it will not load successfully if the backend is not working or not connected. We recommend that you **stand up the backend first**, test using Postman or curl, update the endpoints in the frontend, and then the frontend should integrate smoothly.

Installing Dependencies

1. **Installing Node and NPM** This project depends on Nodejs and Node Package Manager (NPM). Before continuing, you must download and install Node (the download includes NPM) from https://nodejs.com/en/download.

2. **Installing project dependencies** This project uses NPM to manage software dependencies. NPM Relies on the package.json file located in the `frontend` directory of this repository. After cloning, open your terminal and run:

```
npm install
```

> *tip*: `npm i`is shorthand for `npm install`

Required Tasks

Running Your Frontend in Dev Mode

The frontend app was built using create-react-app. In order to run the app in development mode use `npm start`. You can change the script in the `package.json` file.

Open http://localhost:3000 to view it in the browser. The page will reload if you make edits.

```
npm start
```

Request Formatting

The frontend should be fairly straightforward and disgestible. You'll primarily work within the `components` folder in order to understand, and if you so choose edit, the endpoints utilized by the components. While working on your backend request handling and response formatting, you can reference the frontend to view how it parses the responses.

After you complete your endpoints, ensure you return to the frontend to confirm your API handles requests and responses appropriately:

- Endpoints defined as expected by the frontend
- Response body provided as expected by the frontend

Optional: Updating Endpoints and API behavior

Would you rather the API had different behavior - different endpoints, return the response body in a different format? Go for it! Make the updates to your API and the corresponding updates to the frontend so it works with your API seamlessly.

Optional: Styling

In addition, you may want to customize and style the frontend by editing the CSS in the `stylesheets` folder.

Optional: Game Play Mechanics

Currently, when a user plays the game they play up to five questions of the chosen category. If there are fewer than five questions in a category, the game will end when there are no more questions in that category.

You can optionally update this game play to increase the number of questions or whatever other game mechanics you decide. Make sure to specify the new mechanics of the game in the README of the repo you submit so the reviewers are aware that the behavior is correct.

> **Spoiler Alert:** If needed, there are details below regarding the expected endpoints and behavior. But, ONLY look at them if necessary. Give yourself the opportunity to practice understanding code first!

DO NOT PROCEED: ENDPOINT SPOILERS

> Only read the below to confirm your notes regarding the expected API endpoint behavior based on reading the frontend codebase.

Expected endpoints and behaviors

`GET '/categories'`

- Fetches a dictionary of categories in which the keys are the ids and the value is the corresponding string of the category
- Request Arguments: None
- Returns: An object with a single key, categories, that contains an object of id: category_string key:value pairs.

```
{
  "categories": {
    "1": "Science",
    "2": "Art",
    "3": "Geography",
    "4": "History",
    "5": "Entertainment",
    "6": "Sports"
  }
}
```

`GET '/questions?page=${integer}'`

- Fetches a paginated set of questions, a total number of questions, all categories and current category string.
- Request Arguments: `page` - integer
- Returns: An object with 10 paginated questions, total questions, object including all categories, and current category string

```
{
  "questions": [
    {
      "id": 1,
      "question": "This is a question",
      "answer": "This is an answer",
      "difficulty": 5,
      "category": 2
    }
  ],
  "totalQuestions": 100,
  "categories": {
    "1": "Science",
    "2": "Art",
    "3": "Geography",
    "4": "History",
    "5": "Entertainment",
    "6": "Sports"
  },
  "currentCategory": "History"
}
```

`GET '/categories/${id}/questions'`

- Fetches questions for a cateogry specified by id request argument
- Request Arguments: `id` - integer

- Returns: An object with questions for the specified category, total questions, and current category string

```
{
  "questions": [
    {
      "id": 1,
      "question": "This is a question",
      "answer": "This is an answer",
      "difficulty": 5,
      "category": 4
    }
  ],
  "totalQuestions": 100,
  "currentCategory": "History"
}
```

`DELETE '/questions/${id}'`

- Deletes a specified question using the id of the question
- Request Arguments: `id` - integer
- Returns: Does not need to return anything besides the appropriate HTTP status code. Optionally can return the id of the question. If you are able to modify the frontend, you can have it remove the question using the id instead of refetching the questions.

`POST '/quizzes'`

- Sends a post request in order to get the next question
- Request Body:

```
{
    'previous_questions': [1, 4, 20, 15]
    quiz_category': 'current category'
 }
```

- Returns: a single new question object

```
{
  "question": {
    "id": 1,
    "question": "This is a question",
    "answer": "This is an answer",
    "difficulty": 5,
    "category": 4
  }
}
```

`POST '/questions'`

- Sends a post request in order to add a new question
- Request Body:

```
{
  "question": "Heres a new question string",
  "answer": "Heres a new answer string",
  "difficulty": 1,
  "category": 3
}
```

- Returns: Does not return any new data

`POST '/questions'`

- Sends a post request in order to search for a specific question by search term
- Request Body:

```
{
  "searchTerm": "this is the term the user is looking for"
}
```

- Returns: any array of questions, a number of totalQuestions that met the search term and the current category string

```
{
  "questions": [
    {
      "id": 1,
      "question": "This is a question",
      "answer": "This is an answer",
      "difficulty": 5,
      "category": 5
    }
  ],
  "totalQuestions": 100,
  "currentCategory": "Entertainment"
}
```