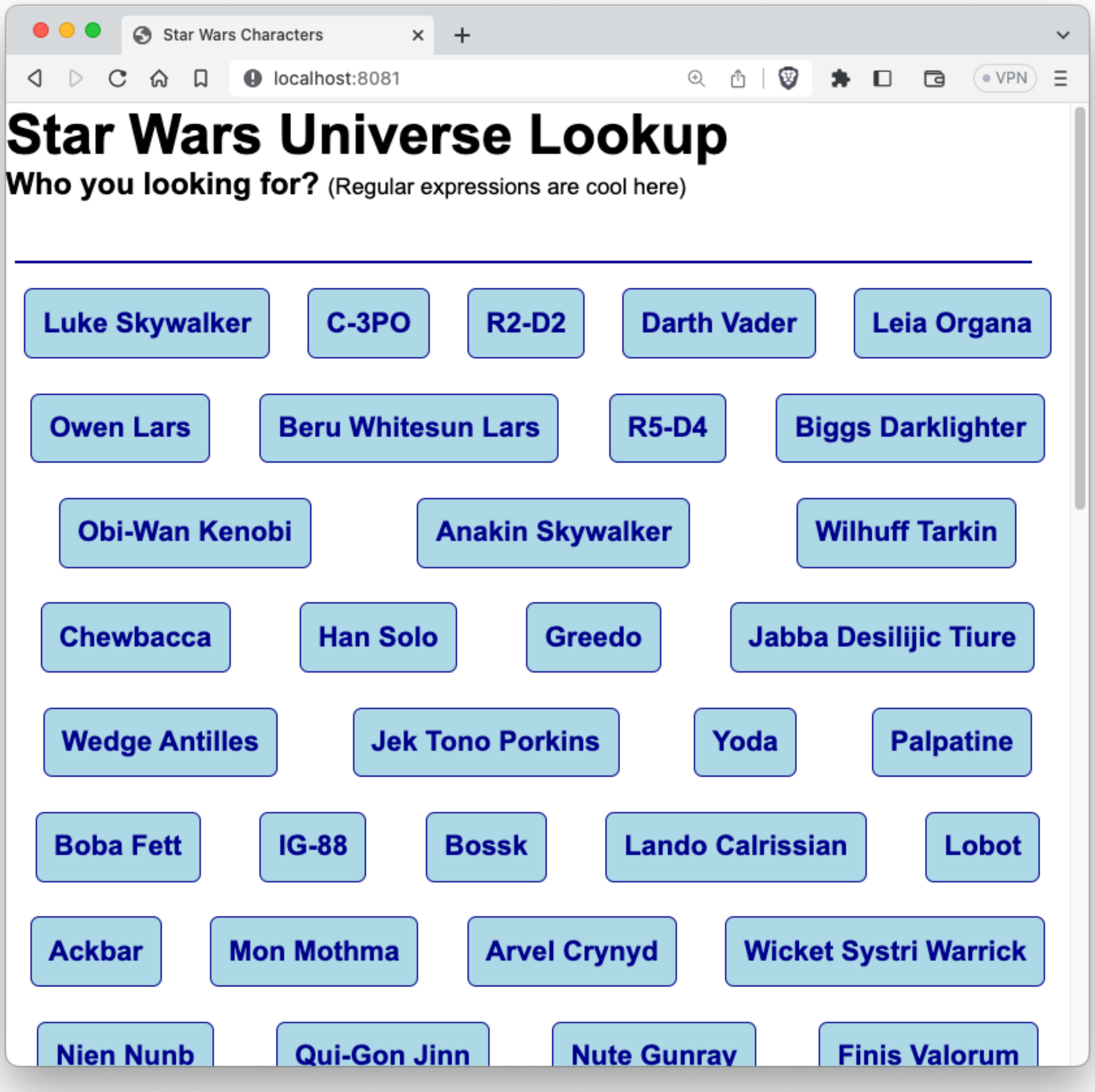
EDP Challenge Project 1

In this lab, we're going to exercise the HTML, CSS, JavaScript, and Git knowledge you've gained. We're going to build a front-end user interface (UI) that interacts with a Star Wars API (SWAPI). We'll do this in pairs in breakout rooms.

The Resulting Site

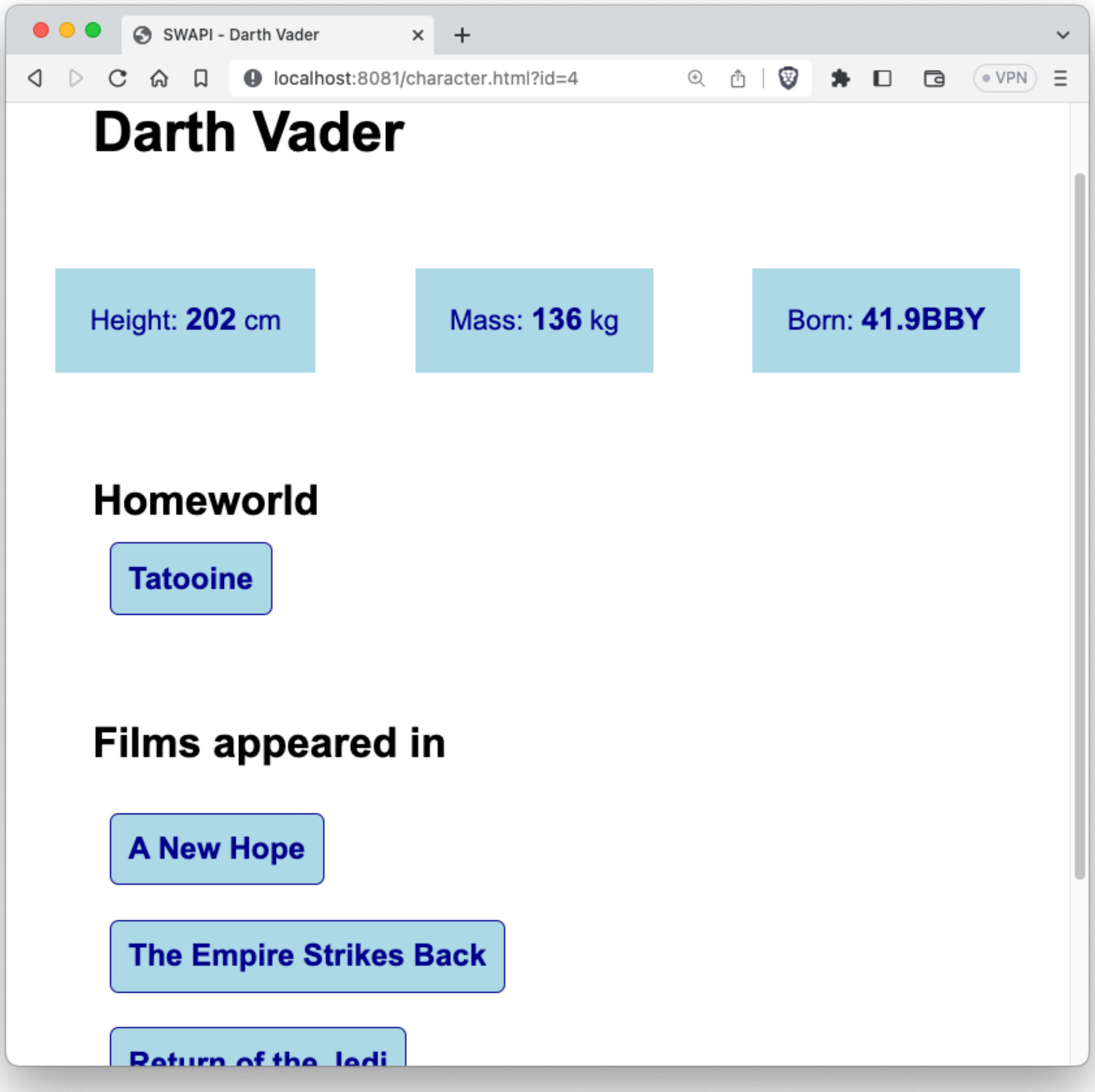
The result will be a web site that begins with a list of Star Wars characters.



Star Wars Universe Lookup

Who you looking for? (Regular expressions are cool here)

Clicking on a name will take you to the details page for that character. Each character has a home world and a list of films they've appeared in.



Darth Vader

Height: **202** cm

Mass: **136** kg

Born: **41.9BBY**

Homeworld

Tatooine

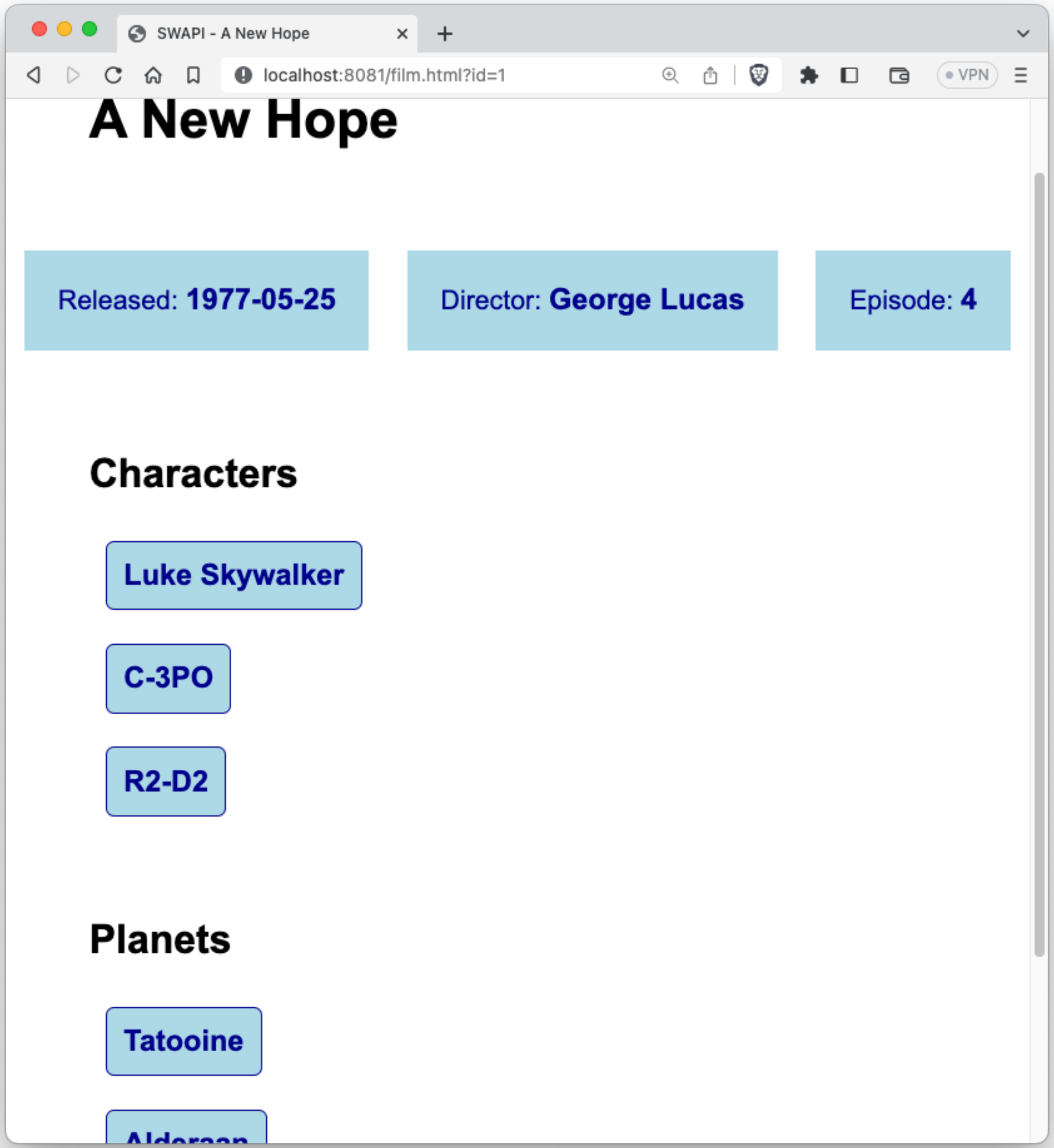
Films appeared in

A New Hope

The Empire Strikes Back

Return of the Jedi

You'll be able to click on film and navigate to the details page for that film. The film will have a list of characters and a list of planets. These also are clickable so the user can get details for them.



A New Hope

Released: **1977-05-25**

Director: **George Lucas**

Episode: **4**

Characters

Luke Skywalker

C-3PO

R2-D2

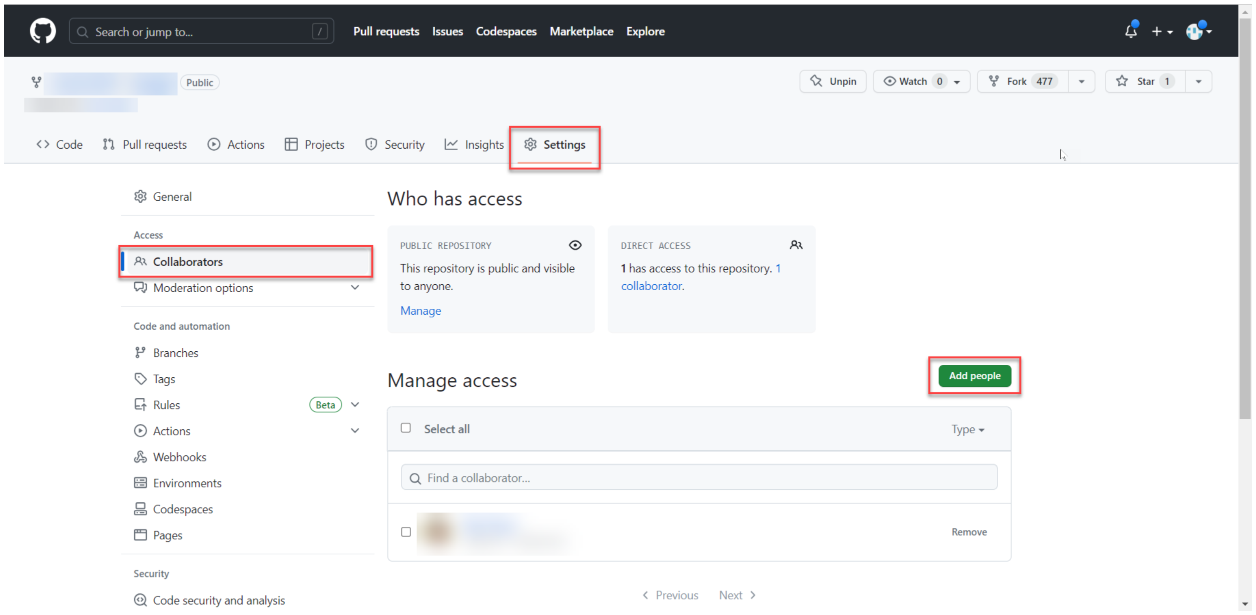
Planets

Tatooine

As you can see, we're allowing the user to navigate between people, planets, and films. You and your partner are going to create this scheme. Partner 1 will create the film page. Partner 2 will create the planet page. We've provided the characters page (index.html) as a starter example. See **guided_project_1_starter.zip** in Basecamp the starter example.

Using GitHub to collaborate with your team members

Your team will use one GitHub repository for collaboration. Partner 1 will set up a **new** GitHub repository using the same GitHub account that was used in the DevOps Foundations course. In GitHub, **invite** the other team member as a **collaborator** so that they can push.



This repository is public and visible to anyone.

1 has access to this repository. 1 collaborator.

Getting the starter

Partner 1 should get the starter files, **guided_project_1_starter.zip,** from Basecamp. They should then commit the starter files to their local repository and push to the GitHub repository. Partner 2 can then pull those into their local repository.

Both partners can now run the project by opening a terminal window in the folder where you have all the HTML, CSS, JS files and running:

```
$ npx browser-sync start --server --files "*.html,*.js,*.css"
```

You'll know it's working when you can browse to http://localhost:3000 and see the list of people.

Writing the new pages

Partner 1 will create the film page:

- Create a new development branch in git. Do all your development in this branch. Feel free to push your changes to the upstream repo at any time through this process.
- Create film.html and film.js.
- Read the film ID from the querystring (ie. get the 10 from "http://localhost:3000/film.html?id=10"). You can do that by using URLSearchParams like this: `const sp = new URLSearchParams(window.location.search);` and then `const id = sp.get('id');` YYY

- Using the fetch API, make a GET request from:
  ` https://swapi2.azurewebsites.net/api/films/${id}`
- There will be exactly one film in the response. Read it into a JavaScript object called film.
- Make another GET request from ` https://swapi2.azurewebsites.net/api/films/${id}/characters` to get the characters in that film.
- Make one last GET request from ` https://swapi2.azurewebsites.net/api/films/${id}/planets` to get the planets for that film.
- Display whatever details you like on the page. At minimum, you must display the characters and the planets. The character/planet name must be a clickable hyperlink that navigates the user to the details page for that entity.

While Partner 1 is writing the film page, Partner 2 will do the exact same thing but with planet. Planet must link to all films and all people.

Merging the branches

Once each partner's page is working properly, they should merge their changes to the main branch and push to the upstream repo. We are setting you up for conflicts. This is your opportunity to get experience in merging cleanly. Feel free to do this along with your partner so you can talk through the conflicts and how to merge them.

When you have a synchronized codebase in all three places (two local and one remote repository) and they're running as required, you are finished.

Bonus work

If you finish early, do these things for extra credit.

Make the pages look good with CSS. Try to write one CSS file for the entire site, not one for each HTML page. Feel free to change the layout.

Here's an advanced assignment: Store a few arrays in local storage; planets, films, and people. Each one should simply store the entity's ID and name. Populate the arrays as you make your GET requests. That way, when you're creating links, if you already have the entity name in the array, you won't have to fetch it again. You're just saving data locally, so you don't have to go the API server every time.

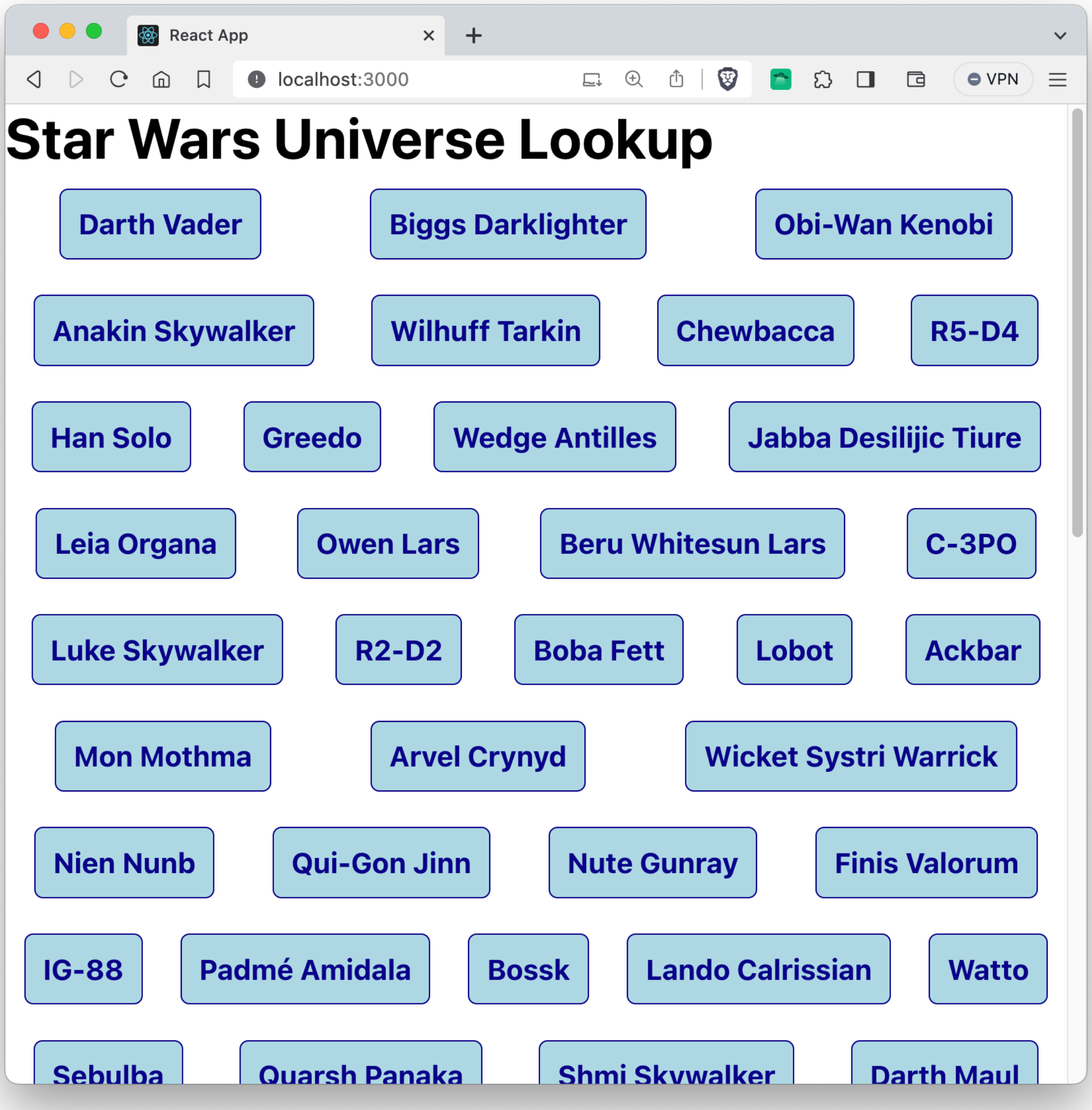
EDP Guided Project 2

In this project, we're going to exercise the React, Node, Express and MongoDB knowledge you've gained. You are going to recreate the SWAPI web site that you built during the first guided project with a MERN stack. We'll do this in pairs in breakout rooms.

You're encouraged to reuse as much of the HTML, CSS, and JavaScript from the prior project that you can. This will allow you to compare a vanilla JS project and a React project.

The Resulting Site

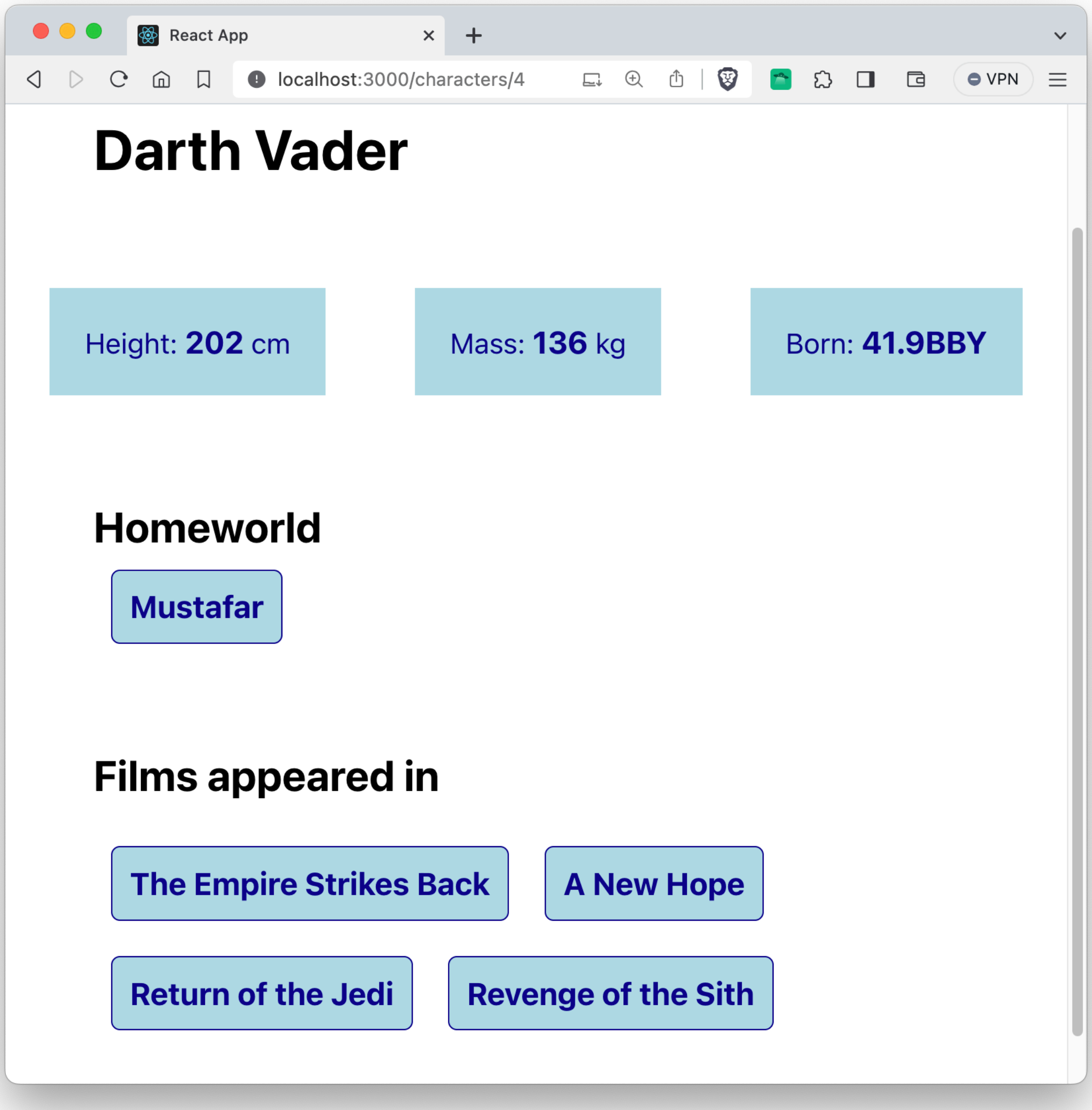
The result will still look and function like the previous website but use a different tech stack.



Star Wars Universe Lookup

Darth Vader Biggs Darklighter Obi-Wan Kenobi
Anakin Skywalker Wilhuff Tarkin Chewbacca R5-D4
Han Solo Greedo Wedge Antilles Jabba Desilijic Tiure
Leia Organa Owen Lars Beru Whitesun Lars C-3PO
Luke Skywalker R2-D2 Boba Fett Lobot Ackbar
Mon Mothma Arvel Crynyd Wicket Systri Warrick
Nien Nunb Qui-Gon Jinn Nute Gunray Finis Valorum
IG-88 Padmé Amidala Bossk Lando Calrissian Watto
Sebulba Quarsh Panaka Shmi Skywalker Darth Maul

Clicking on a name will route you to the details component for that character. (Note that you're using a route parameter here). Each character has a home world and a list of films they've appeared in.



Darth Vader

Height: **202** cm

Mass: **136** kg

Born: **41.9BBY**

Homeworld

Mustafar

Films appeared in

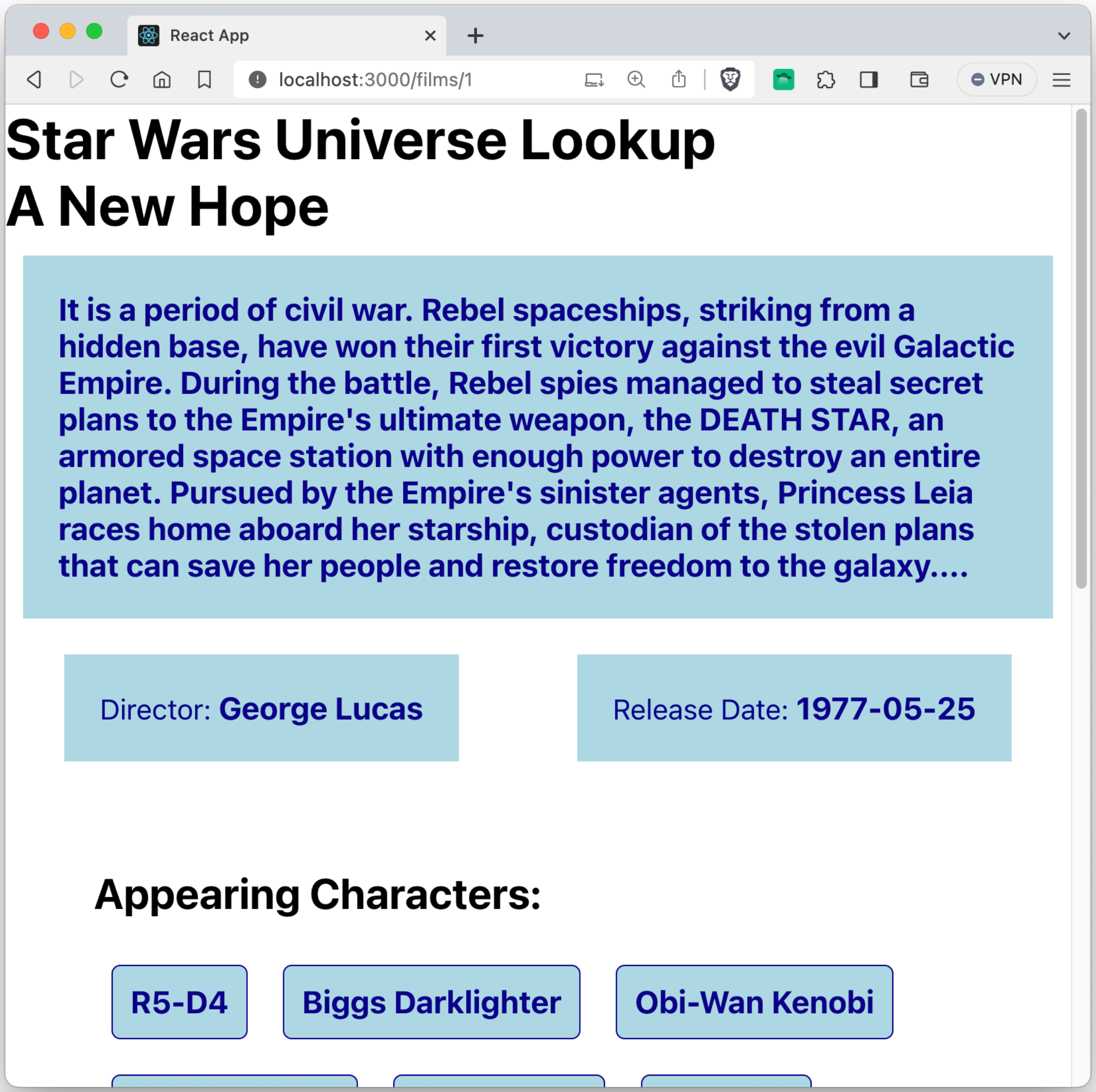
The Empire Strikes Back

A New Hope

Return of the Jedi

Revenge of the Sith

You'll be able to click on film and navigate to the details component for that film. The film will have a list of characters and a list of planets. These also are clickable so the user can get details for them.



Star Wars Universe Lookup
A New Hope

It is a period of civil war. Rebel spaceships, striking from a hidden base, have won their first victory against the evil Galactic Empire. During the battle, Rebel spies managed to steal secret plans to the Empire's ultimate weapon, the DEATH STAR, an armored space station with enough power to destroy an entire planet. Pursued by the Empire's sinister agents, Princess Leia races home aboard her starship, custodian of the stolen plans that can save her people and restore freedom to the galaxy....

Director: **George Lucas**

Release Date: **1977-05-25**

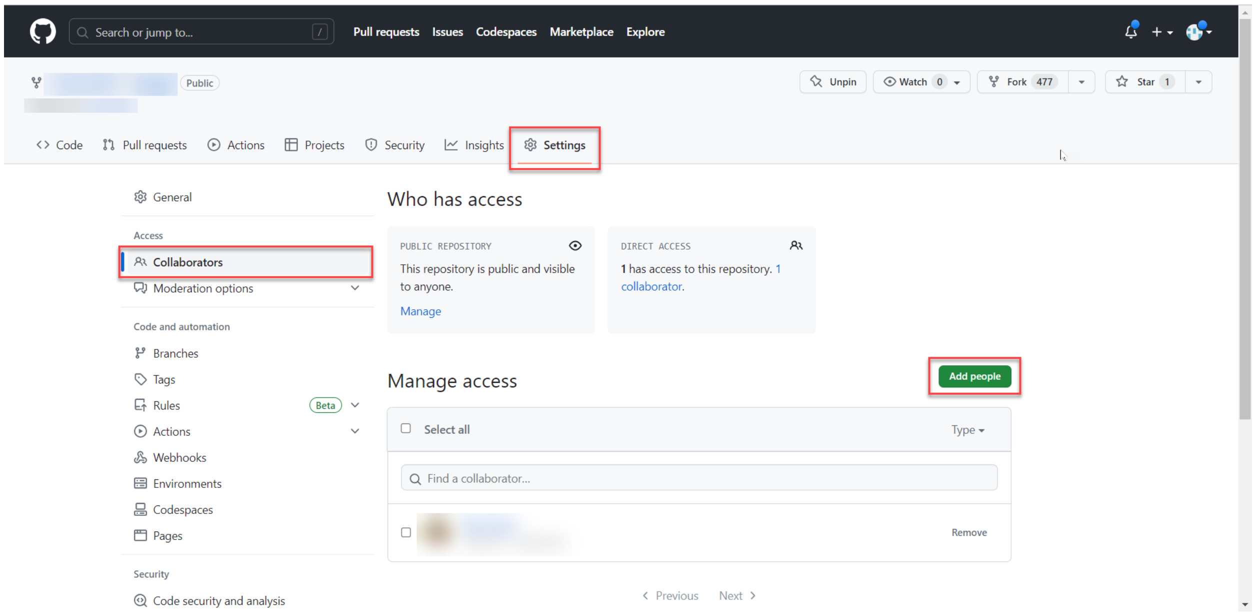
Appearing Characters:

R5-D4 | Biggs Darklighter | Obi-Wan Kenobi

As you can see, we're allowing the user to navigate between people, planets, and films. You and your partner are going to create this scheme. Partner 1 will create the films component. Partner 2 will create the planets component.

Using GitHub to collaborate with your team members

Your team will use one GitHub repository for collaboration. Partner 1 will set up a **new** GitHub repository using the same GitHub account that was used in the DevOps Foundations course. **Don't** reuse the GitHub repository from the Guided Project Part 1. In GitHub, **invite** the other team member as a **collaborator** so that they can push.



Part 1: Creating the Mongo Database

Tech involved: MongoDB

In this exercise, you will load the SWAPI data into MongoDB. There's no shared database; each team member must load data into their local MongoDB instance. Collaborate as a pair on how to do this. The json files containing the data can be found here https://github.com/olaekdahl/swapi/tree/master/json-data.

- Start Mongo with mongod.
- Load data into MongoDB. You'll load at minimum characters, films, planets, films_characters, and films_planets.
- NOTE: This is not a mongosh command. It's a CLI tool.
    - Example: **mongoimport --uri mongodb://localhost:27017/swapi --collection films --drop --file c:/swapi-data/films.json --jsonArray**
- Using mongosh, examine the data added and make sure it loaded as expected.

Part 2: Creating the Node/Express server

Tech involved: Node, Express, and git

Next, we'll need to create the Node/Express server.

- Create a new folder and cd into that folder.
- Use `npm init` to create a new package.json.
- Create a new file called server.js.
- Edit package.json. Add a "script" that runs your server.js file using node. It should run your server when you go `npm run start`
- npm install express

- In your server.js file, create the express app. Make sure it is listening on whatever port you choose.
- Create a GET route for /api/planets. It should just return a fake test object.
- Test it out by making a GET request to /api/planets. Make sure it is returning the data expected.
- git commit, merge, push, and pull to synchronize both partner's repo before moving on.

Part 3: Connecting Node to MongoDB

Tech involved: Node, Express, MongoDB, and git

Let's get our Node/Express server reading from the Mongo database. You'll create some routes. Consider dividing these between partners and merge them using git.

- Make sure your mongod process is still running.
- Edit your server.js. Connect it to your Mongo database. Use the techniques you learned in class with the mongo client, mongo database, and mongo collections.
- Create these routes:
    - /api/characters
    - /api/films
    - /api/planets
    - /api/characters/:id
    - /api/films/:id
    - /api/planets/:id
    - /api/films/:id/characters
    - /api/films/:id/planets
    - /api/characters/:id/films
    - /api/planets/:id/films
    - /api/planets/:id/characters
- Test each of these routes using a browser, curl, or a tool like Postman.

Part 4: Creating the web app with React

Tech involved: React, git, pair programming

Both partners pair on the creation of the base site using `npm create vite@latest <your project name> -- --template react`

As a pair:

- Create the initial component that fetches all characters and displays them. For now, don't worry about implementing the search/filter functionality until you've finished everything else.
- Use appropriate React hooks like useEffect() and useState() to make a fetch call from the `/api/characters` Express route.
- Iterate those characters using .map() and display one <div> for each character.
- Add a click event for each that will route to `/character/:id`

As a pair:

- Create the Character.js component.

- Read the characterId as a route parameter.
- Using useEffect() and useState(), make a GET fetch from `/api/characters/:id` (the Express API endpoint)
- Process the response.
- Make another GET request from `/api/characters/:id/planet` and one from `/api/characters/:id/films`
- Display their properties like the screen shot above.

Either as a pair or as separate development, create the Film component and the Planet component. Maybe partner 1 will create the Film component and partner 2 will create the Planet component.

Part 5: Putting it all together

Tech involved: React, Node, Express, and MongoDB

This last major step is the easiest provided that the prior steps went well. You're merely going to have your React app deployed to your own Node/Express server and have it consume its own API.

- In your Node/Express server, create a folder called public.
- Tell Express to serve static files from that folder using ` app.use(express.static('./public'))`
- Put a test.html file in there. Make sure you can browse to your text.html file in a web browser like Chrome or Firefox.
- Compile your React app by going `npm run build`.
- Copy/move the contents of the React project's build folder to your Node/Express public folder.
- You should be able to browse to your React app through your Node server's port.

When your React app is running from your Node/Express server and is consuming the data from your own MongoDB instance, you are finished.

EDP Guided Project 3

In this project, you will create a solution to help predict if a Star Wars character will join the Empire or the Resistance based on their home planet and unit type. You will start by reviewing a Python script that will help you generate sample data to train a model. Once the model has been built, you will use it to predict a 10 million record data set.

Good luck and may The Force be with you.

Part 1: Review a Python script to generate data

Review the Python script named **generate_data.py** (see Guided Project zip for file). The script generates data for 1000 Star Wars characters. Each record will have a:

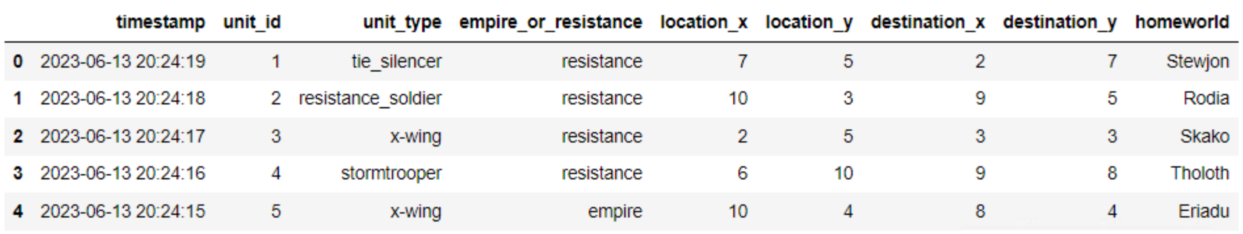
- timestamp
- unit_id (sequential integer)
- unit_type (random selection from sequence below)

unit_type=["stormtrooper", "tie_fighter", "at-st", "x-wing", "resistance_soldier", "at-at", "tie_silencer", "unknown"]

- empire_or_resistance (random selection of either empire or resistance)
- location_x
- location_y
- destination_x
- destination_y
- home_world (read data from a file named **home_worlds.json,** see Guided Project zip for file, and select a random homeworld)

The generated data is saved to a file named **troop_movements.csv**. The x and y co-ordinates simulate troops moving from one position to another in battle on an imaginary grid.

Here's an example of what the data should look like when read into a data frame.



	timestamp	unit_id	unit_type	empire_or_resistance	location_x	location_y	destination_x	destination_y	homeworld
0	2023-06-13 20:24:19	1	tie_silencer	resistance	7	5	2	7	Stewjon
1	2023-06-13 20:24:18	2	resistance_soldier	resistance	10	3	9	5	Rodia
2	2023-06-13 20:24:17	3	x-wing	resistance	2	5	3	3	Skako
3	2023-06-13 20:24:16	4	stormtrooper	resistance	6	10	9	8	Tholoth
4	2023-06-13 20:24:15	5	x-wing	empire	10	4	8	4	Eriadu

Since we are generating random data, there will be "bad" Empire characters ending up as Resistance and vice versa.

Part 2: Build a prediction model

Read the generated data from the csv file into a Pandas data frame and explore the data.

NOTE: All images below are just examples. Your numbers will be different.

- Create grouped data showing counts of empire vs resistance.

```
  empire_or_resistance  count
0               empire     24
1           resistance     26
```

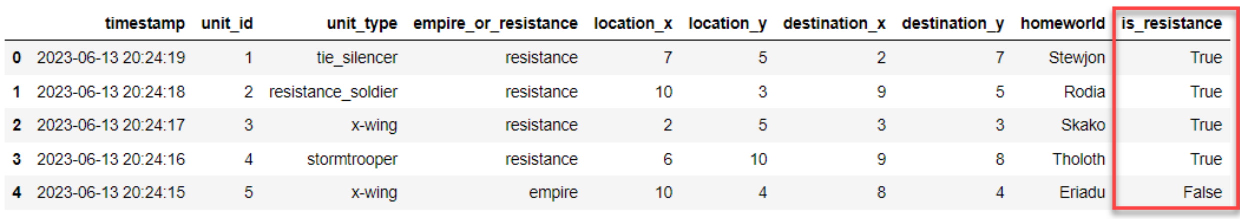
- Create grouped data showing counts of characters by homeworld.

```
        homeworld  count
0        Alderaan      1
1     Aleen Minor      1
2      Bestine IV      3
3           Cerea      2
4       Chandrila      1
5    Concord Dawn      1
6        Corellia      3
7         Dagobah      2
```

- Created grouped data showing counts of characters by unit_type.

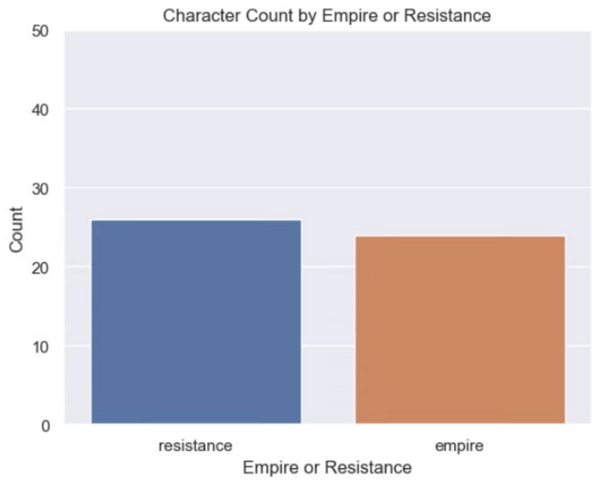
```
           unit_type  count
0              at-at      3
1              at-st      3
2  resistance_soldier     10
3        stormtrooper     11
4         tie_fighter      8
5        tie_silencer      9
6              x-wing      6
```

- Engineer a new feature called is_resistance with a True or False value based on empire_or_resiatance.



	timestamp	unit_id	unit_type	empire_or_resistance	location_x	location_y	destination_x	destination_y	homeworld	is_resistance
0	2023-06-13 20:24:19	1	tie_silencer	resistance	7	5	2	7	Stewjon	True
1	2023-06-13 20:24:18	2	resistance_soldier	resistance	10	3	9	5	Rodia	True
2	2023-06-13 20:24:17	3	x-wing	resistance	2	5	3	3	Skako	True
3	2023-06-13 20:24:16	4	stormtrooper	resistance	6	10	9	8	Tholoth	True
4	2023-06-13 20:24:15	5	x-wing	empire	10	4	8	4	Eriadu	False

- Create a bar plot using Seaborn showing Empire vs Resistance distribution.

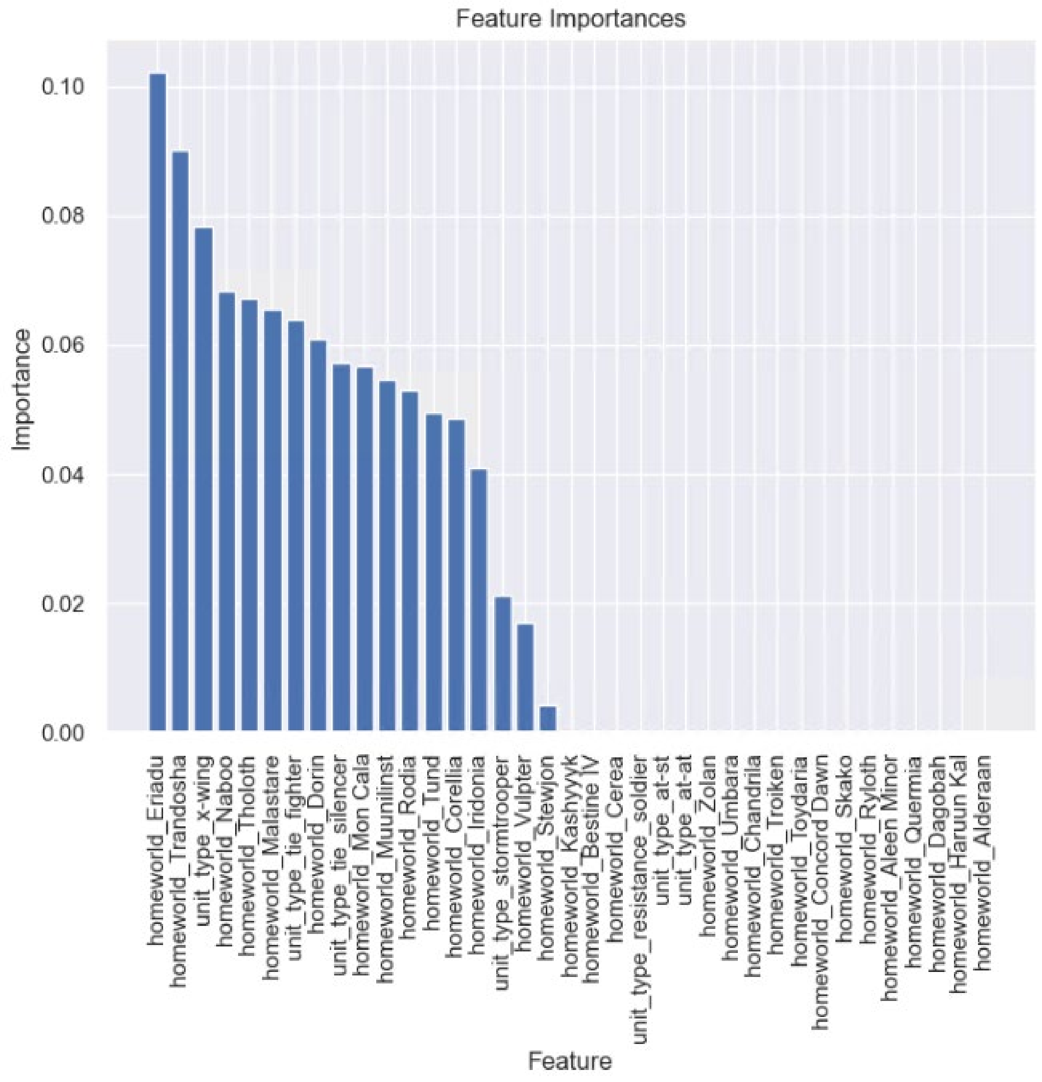


- Create a prediction model using **sklearn.tree.DecisionTreeClassifier** that predicts if a character is joining either the Empire or the Resistance based on their homeworld and unit_type.

- Convert categorical features to numeric using **pd.get_dummies**.
- Create a bar plot that shows feature importance.
- Example code to get feature importance:

```
# Get feature importances
importances = model.feature_importances_

# Create a DataFrame to hold the feature importances
feature_importances = pd.DataFrame({'Feature': X_encoded.columns,
'Importance': importances})
```



```
Most Influential Unit Type: homeworld_Eriadu
```

- Save to model as a pickle file named **trained_model.pkl**.

Part 3: Use the trained model with “real” data

Load data from **troop_movements10m.csv** (see Guided Project zip for file). This file contains 10 million records to be predicted.

This data must be cleaned up a bit before it can be used:

- Some unit_type records have a value of invalid_unit. Replace that with **unknown**.
- Some location_x and location_Y values are missing. Use the **ffill** method to fill.
- Save the clean data into a Parquet file named troop_movements10m.parquet.
    - You need to install pyarrow and fastparquet to support saving to a Parquet file.

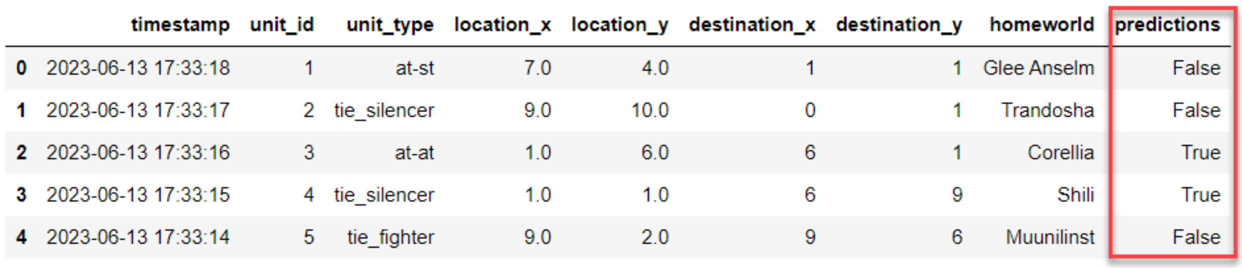
```
pip install pyarrow
```

```
pip install fastparquet
```

Load the pickled model and load the data from the Parquet file into a data frame. Run the data through the model.

Add the predicted values to the data frame. The finished result should look like this:

NOTE: Your values will be different.



	timestamp	unit_id	unit_type	location_x	location_y	destination_x	destination_y	homeworld	predictions
0	2023-06-13 17:33:18	1	at-st	7.0	4.0	1	1	Glee Anselm	False
1	2023-06-13 17:33:17	2	tie_silencer	9.0	10.0	0	1	Trandosha	False
2	2023-06-13 17:33:16	3	at-at	1.0	6.0	6	1	Corellia	True
3	2023-06-13 17:33:15	4	tie_silencer	1.0	1.0	6	9	Shili	True
4	2023-06-13 17:33:14	5	tie_fighter	9.0	2.0	9	6	Muunilinst	False

BONUS

- Expose the model as a RESTful API using Node or Flask.
- Create a React component that can input Homeworld and Unit Type and display the prediction.
- Display the Feature Importance chart in a React component.