# 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.

Graphical user interface

Description automatically generated

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.

A screenshot of a computer

Description automatically generated

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.

A screenshot of a computer

Description automatically generated

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 you 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.

A screenshot of a computer

Description automatically generated

# 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,
  + 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 `npx create-react-app <your-project-name>`.

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`
* 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.