UI/UX DESIGN

A summer internship report submitted in partial fulfillment of the requirements for the award of degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

BY

Lakshmi Sanjana D

HU21CSEN0101118

Under the Guidance of

Dr. Yugandhar Garapati

Assistant Professor

Department of Computer Science and Engineering

GITAM School of Technology

GITAM Deemed to be University Hyderabad

Campus -502329

August- 2024

GANDHI INSTITUTE OF TECHNOLOGY AND MANAGEMENT
(GITAM)
(Declared as Deemed-to-be-University u/s 3 of UGC Act 1956)
HYDERABAD CAMPUS

Declaration

I hereby declare that the summer internship report entitled "**UI/UX DESIGN**" is an original work done in the Department of Computer Science and Engineering, GITAM School of Technology, GITAM (Deemed to be University) submitted in partial fulfillment of the requirements for the award of the degree of "Bachelor of Technology" in Computer Science and Engineering. The work had not been submitted to any other college or university for the award of any degree or diploma.

Place - Hyderabad

Date:13-August-2024

Lakshmi Sanjana D
HU21CSEN0101118

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GITAM SCHOOL OF TECHNOLOGY

GITAM (DEEMED TO BE UNIVERSITY)

HYDERABAD CAMPUS

Certificate

This is to certify that the Internship report entitled "**UI/UX DESIGN**" is a bonafide record of work carried out by **Lakshmi Sanjana D (HU21CSEN0101118)** submitted in partial fulfillment of the requirement for the award of the degree of Bachelors of Technology in Computer Science and Engineering.

Dr. Yugandhar Garapati
Assistant Professor
Dept. Of CSE

Dr.Mahaboob Basha Shaik
Professor and HOD

CERTIFICATE

REGD. OFF : 5/484,Kunnukad, Thenkurissi,
Palakkad- 678671 CIN: U72900KL2016PTC047519
CORP. OFF : #4, 2nd Floor, Athulya Annexe Block,
Infopark SEZ, Kakkanad, Ernakulam- 682042

Vaiśeṣika

VAISESIKA CONSULTING PRIVATE LIMITED

TO WHOMSOVER IT MAY CONCERN
Internship & Experience Letter

It gives me an immense pleasure to write a recommendation for Ms. Lakshmi Sanjana D,who is pursuing her Bachelor of Technology from GITAM, HYD with Computer Science stream. Sanjana has completed her 45 days(6th May 2024 to 21st June 2024) internship program with us and her contribution towards JavaScript, HTML, CSS and UI/UX design is highly appreciated.

Vaisesika is a fast-growing technology solutions and consulting services company, enabling the foundations of digital for clients using data, cloud and DevOps approach, to accelerate enterprise digital transformation. We offer a highly motivated & skilled team in Software Development, Quality Engineering, Computer System Validation, ERP services, Cloud Migration & DevOps, RPA, Data Analytics & Business Intelligence.

During the 45 days tenure of the entire internship program at Vaisesika Consulting, Sanjana has contributed towards designing & creating the interactive Forms, Various website and reported in time. It helps in Gap Analysis and meeting the timelines with high accuracy & quality deliverables. Her contribution is highly appreciated by her coworkers and leadership team.

Sanjana is a fast learner and in no time, she has proved herself as one of the top performers in the team. She is a good team player with positive attitude. She has good programming skills in JavaScript, HTML and CSS.

We found Sanjana technical, Sincere and hardworking analytical/research-oriented attitude. I am sure that after completion of her B.Tech, she will be an asset for an organization she joins.

We wish her all the best and great success in her personal and professional career.

With kind regards,
Yours sincerely,

For Vaisesika Consulting Private Limited

Aparna Warrier
Head- HR and Operations
www.vaisesikaconsulting.com

Contact us :
Phone no: +91 -9495650000 (India),+1- 9206508469 (USA)
Email: vaisesika@vaisesikaconsulting.com Website: www.vaisesikaconsulting.com

ACKNOWLEDGEMENT

Apart from my effort, the success of this Internship largely depends on the encouragement and guidance of many others. I take this opportunity to express my gratitude to the people who have helped me in the successful competition of this Internship.

I would like to thank respected **Prof. D. Sambasiva Rao**, Pro Vice Chancellor, GITAM Hyderabad, and **Dr. N. Seetharamaiah**, Associate Director, GITAM Hyderabad.

I would like to thank respected **Dr. Mahaboob Shaik Basha,** Head of the Computer Science and Engineering for giving me such a wonderful opportunity to expand my knowledge for my own branch and giving me guidelines to present an Internship report. It helped me a lot to realize of what we study for.

I would like to thank the respected faculties **Dr. Yugandhar Garapati** who helped me to make this Internship a successful accomplishment.

I would also like to thank my friends who helped me to make my work more organized and well-stacked till the end.

Name: Lakshmi Sanjana D
Pin: HU21CSEN0101118

Content

QTripStatic

1.1 Overview

Objective

In this module, We will be working on the Landing Page of the QTrip website.

- You will understand how to come up with a layout for a given mock-up and start adding each element of this layout one by one.
- You will start off with a basic HTML file for this page and start constructing the page on top of it.

Primary goals

1. Understand how to deconstruct a layout you want to implement.
2. Add a Navigation bar with Brand, Links and Toggle functionalities.
3. Add a Hero section to the page with description and text input sections and align them.
4. Create a grid of cities on the page.

Prerequisites

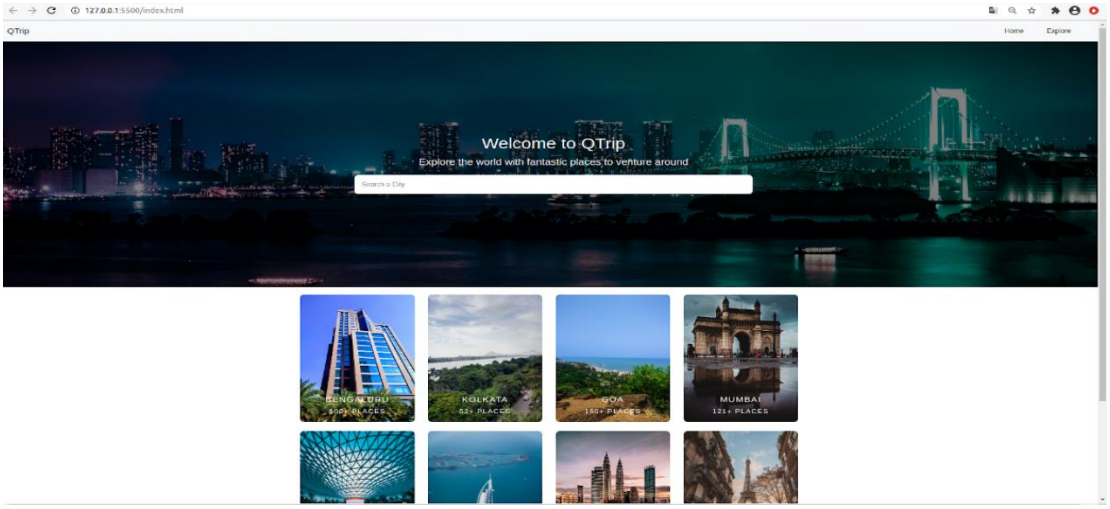
Basic working knowledge of HTML, CSS and Bootstrap.

You are going to embark on a journey of building a travel website where you will learn how to apply HTML and CSS fundamentals. We will learn to

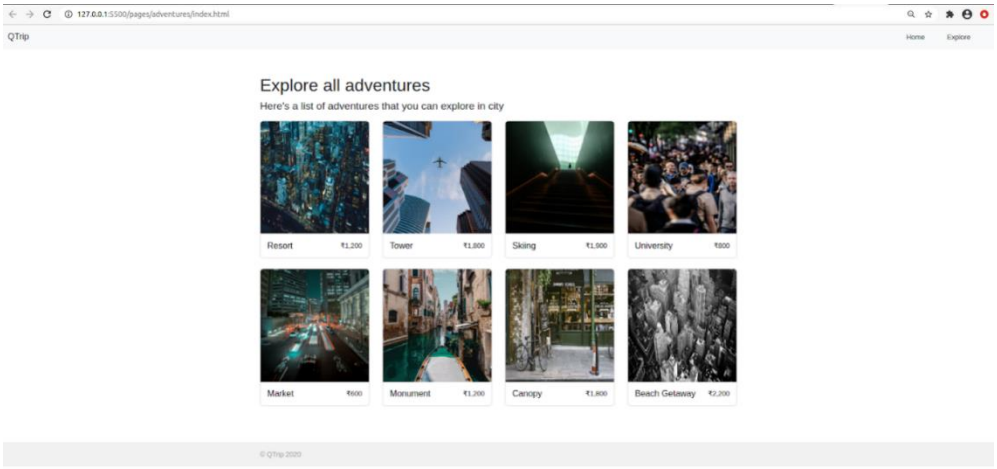
- Construct web pages from the ground up
- Deploy them on a server for the world to see.

QTrip website will have these 3 pages

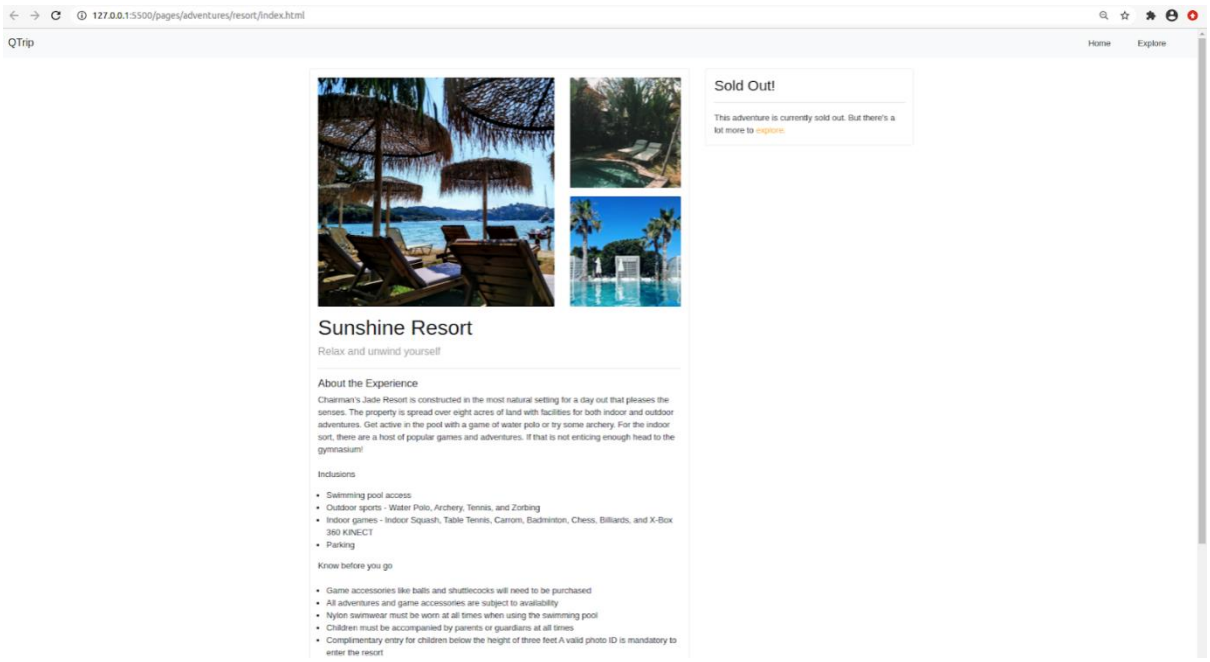
- **Landing Page** - This is the starting page for the website. It lists the various cities for which you can explore adventures



- **Adventures Grid Page** - This is the page that you'll get to by clicking on one of the city cards on the Landing Page. It lists the Adventures available for that city.



- **Adventure Details Page** - This is the page that you'll get to by clicking on one of the Adventures on the Adventures Grid Page. It lists details about that particular Adventure.



1.2 Background

The focus of this Micro-Experience (ME) is to use HTML, CSS and Bootstrap skills to create a Frontend website. QTrip has this travel website where users can browse the various Adventures on offer, based on their places of interest and book an Adventure.

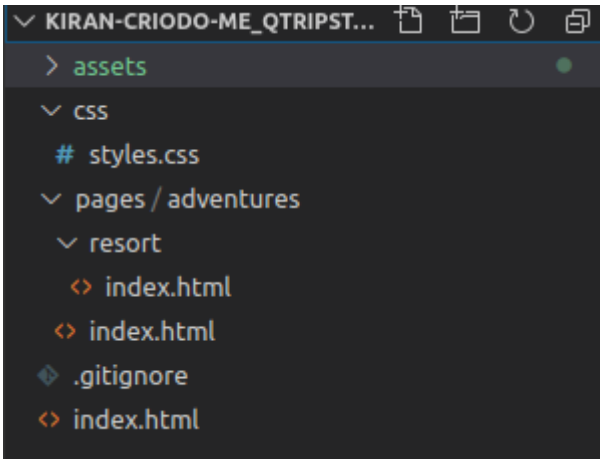
- They can navigate between the pages by clicking on the cards and using the navigation bar.
- The website can also be viewed on other devices such as smartphones and tablets.

You are building static web pages in this. Some of the more dynamic features will get added in the next QTrip ME.

Milestone 1 : Deconstructing the layout

Code Overview

Under the QTrip folder in your workspace, you will see this structure



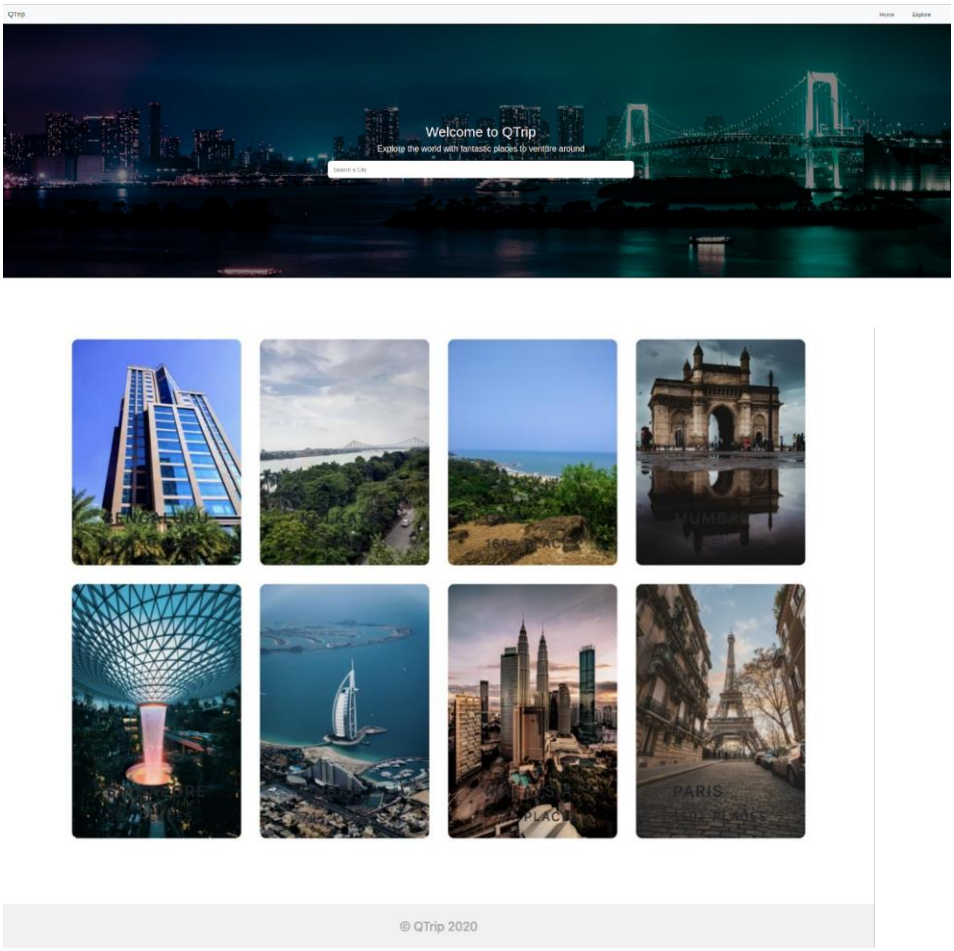
These are the files and directories you'll be working with in this ME

1. HTML files to define the structure of

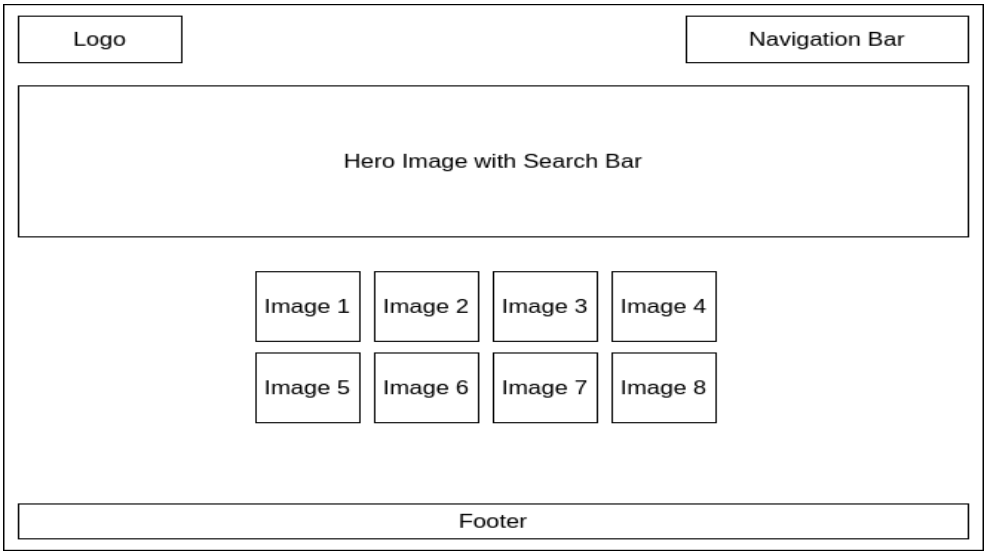
1. Landing page - index.html
2. Adventures Grid Page - pages/adventures/index.html
3. Adventure Details Page - pages/adventures/resort/index.html

2. css/styles.css is the main CSS stylesheet you'll be using for all the 3 web pages.
3. assets folder contains the images needed for the web pages. You will not be modifying this folder. You'll only be linking these images in the pages.

1.3 Introduction to Landing Page layout



After taking a good look at the website we have come up with this format

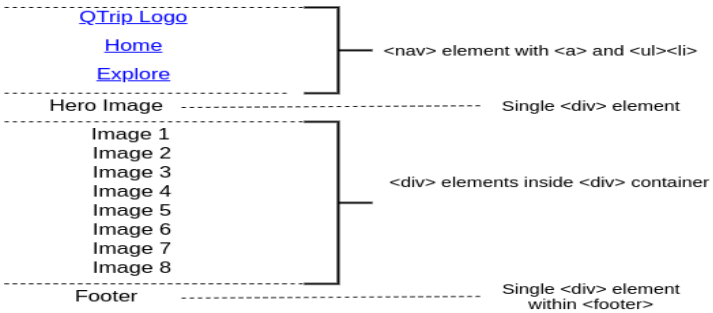


Writing HTML

One way to structure the elements will be to use a

- <nav> element to specify a set of navigation links
- element to specify divisions in the layout, this can be used to form the containers encapsulating the
- Hero section elements
- Grid of images
- Footer section elements

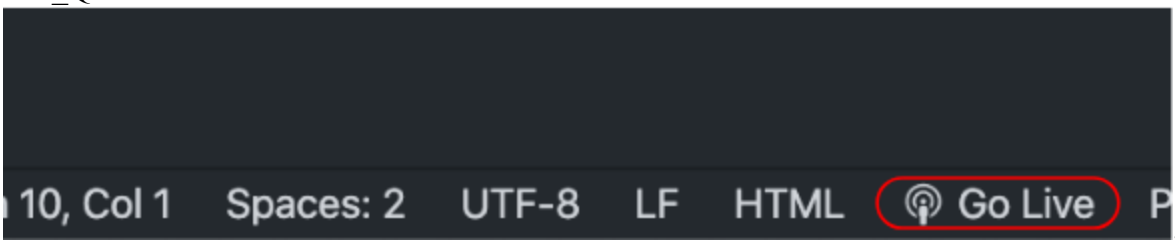
The output with indicators on what the elements are is shown below.

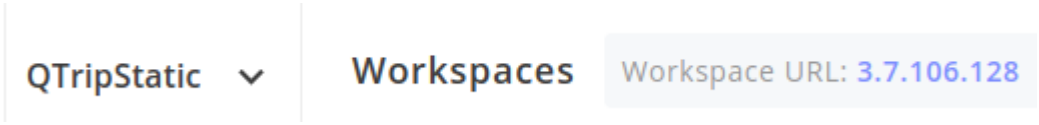


1.4 Navigation Bar

Use Live Server to view the Landing Page

Open the index.html file at /home/crio-user/workspace/damuluru-sanjana-ME_QTRIPSTATIC/ in VSCode and click the “Go Live” button at the bottom of the screen.





- Open a new tab and visit <your-workspace-ip>:8081 to see the rendered content of index.html.

Navigation Bar

The <nav> element should contain links to other HTML documents or internal links within the webpage.

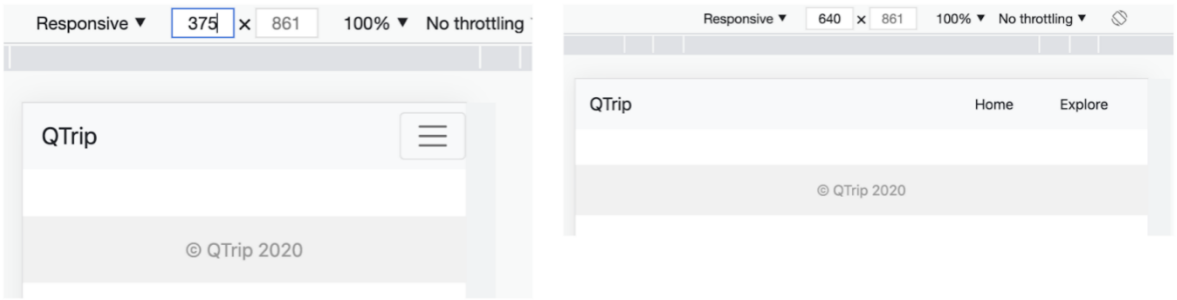
- In general, one can commonly find the navigation bar at the top of the page.
- Note that you can have several <nav> blocks within the page, not only at the top.
- One of the other advantages of using <nav> element is that screen readers can use it to help users determine where to navigate quickly.

Add Navigation Bar using Bootstrap

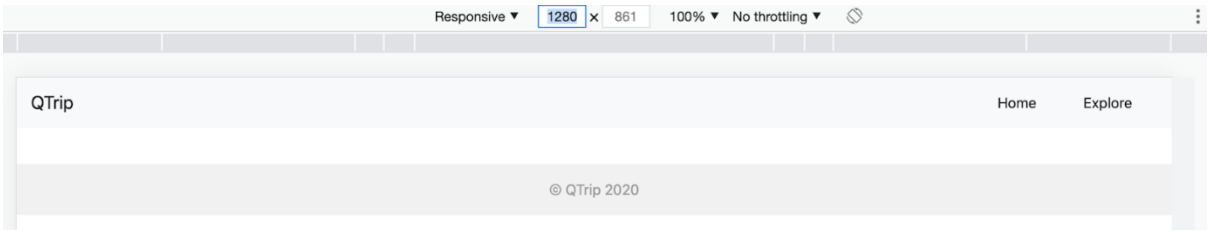
You will see an empty <nav> element within the html <body> in the Landing Page's index.html. This represents the Navigation bar for the page.
Given that the web page will be viewed on all kinds of devices including computer screens, mobiles and tables, the <nav> element needs to be made responsive. Throughout this ME, you'll be given screenshots of how the website should look like on 3 different screen sizes (unless otherwise specified)

- A Mobile device - 375px (xs i.e. <576px)
- A Tablet device - 640px (sm i.e. ≥576px)
- A Desktop device - 1280px (xl i.e. ≥1200px)

The navigation bar should function as shown



View on < 576px (left), View on 640px (right)



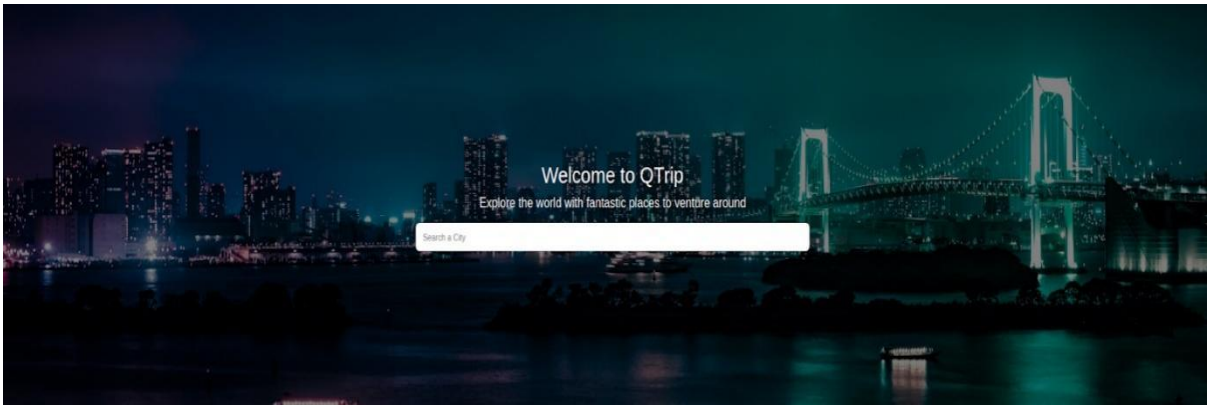
View on 1280px

- Add the required CSS and JS files to set up Bootstrap
- Use Bootstrap classes to set the behaviour and view of the <nav> element similar to the one shown above, for different screen sizes
- The “QTrip” brand, “Home” and “Explore” are all links
- The “QTrip” brand link should be left aligned while the other two are to be right aligned
- Set href of
- “QTrip” brand and “Home” as “#”
- “Explore” as the Adventures Grid page

1.5 : Hero Image

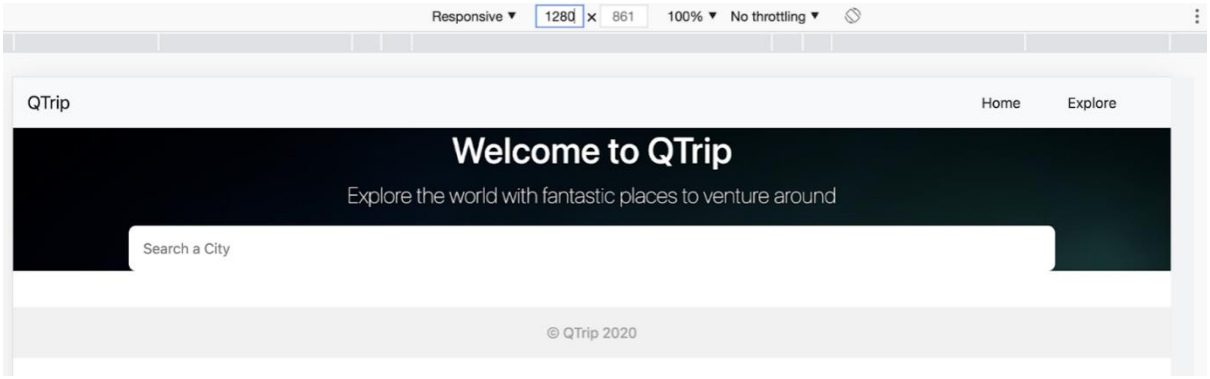
What is a Hero Image?

Also called the Hero header, a Hero image generally refers to the large image that shows prominently as a banner upon loading of the screen. In this case, the designer’s visualization has a welcome message and a search bar along with the hero image.



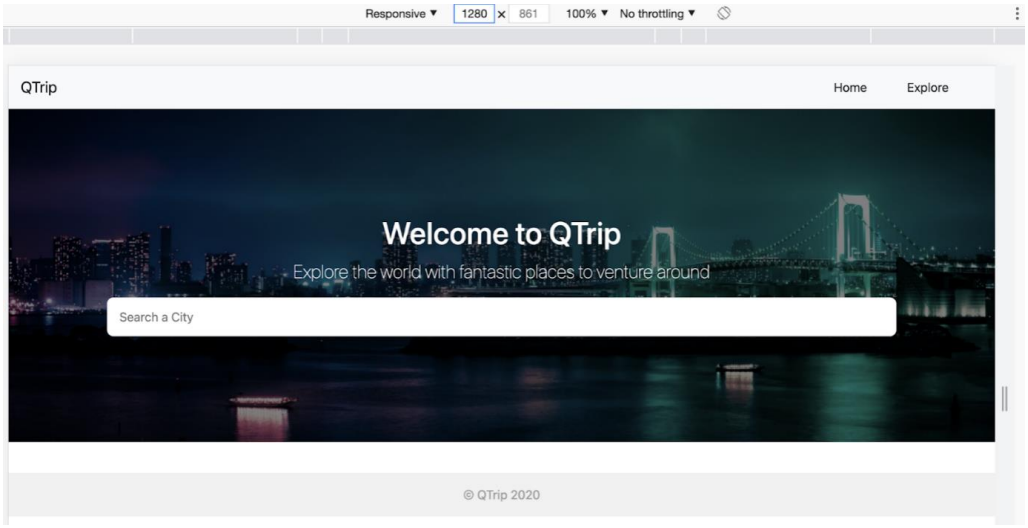
Adding the Hero section elements

Your first checkpoint will be adding the required HTML elements so that the page behaves as shown



View on 1280px

Add the required HTML to the element to produce the above behaviour



View on 1280px

Update the .hero-image class selector in css/styles.css to get the styling of the hero image as required

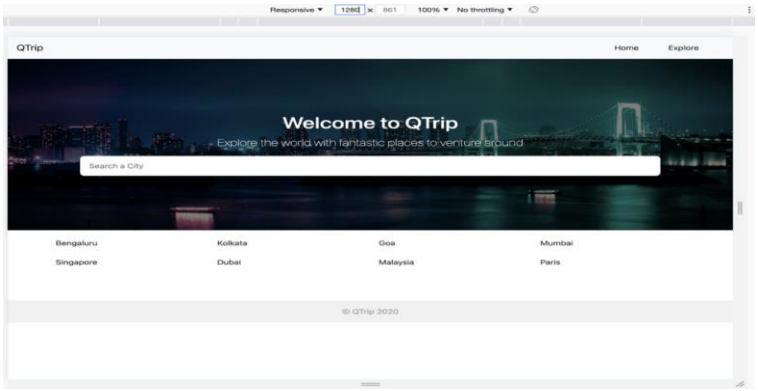
1.6 Create Grid of Cities

Add Rows and Columns of Cities in a Grid

It's time to add that final part of the Landing Page - the grid of cities.

You should add the grid structure such that your web page looks behaves as shown:

Use Bootstrap Grid's row and col-* classes to create the cities grid structure. You'll be adding HTML inside the element that follows the hero image section in the Landing Page's index.html page.



View on 1280px

1.7 Update the git repo

```
# Go to the workspace directory
cd ~/workspace/damuluru-sanjana-ME_QTRIPSTATIC/
# Add the updates for next commit
git add .
# Commit the changes
git commit -m "Module 1 commit"
# Push progress to the git repo
git push -u origin master
# Update to the next module id
python3 helper.py --update CARDS
```

Where does this module fit in the big picture?

We started with the Landing Page for our QTrip website.

- It is now taking shape with the Navigation bar, hero section and list of cities.
- We will enhance the city grid by adding images to them and making the web page responsive in the next module.

- Then we'll move on to the Adventures Grid and the Adventure Details pages.
-

1.8 Summary

- Bootstrap's navbar component makes it easier to create navigation bars. We can customise these further by using color utilities for styling as well as flex utilities for layout.
- Hero image gives a pleasant touch to our web pages. If a background image is used, we can utilise the background-* properties to configure how this image should be displayed within the wrapper container
- Bootstrap Grid makes it super easy to create responsive Grid structure.
  The row and col-* classes are utilised for this
- By focusing on the Grid layout first, we checked off one of the requirements in the Cities Grid view
- With that done, we can now focus on getting the individual Cities card or tile view correct

QTripDynamic

2.1 Overview

Objective

In this module, you will be working on the Landing Page of the QTrip website.

- You will start by adding JavaScript methods necessary to make API calls to the Backend.
- The goal is to fetch information related to the Cities being displayed on the Landing page using REST API.
- Once this data is received in the API Response, it needs to be processed and plugged into the HTML DOM of the page.

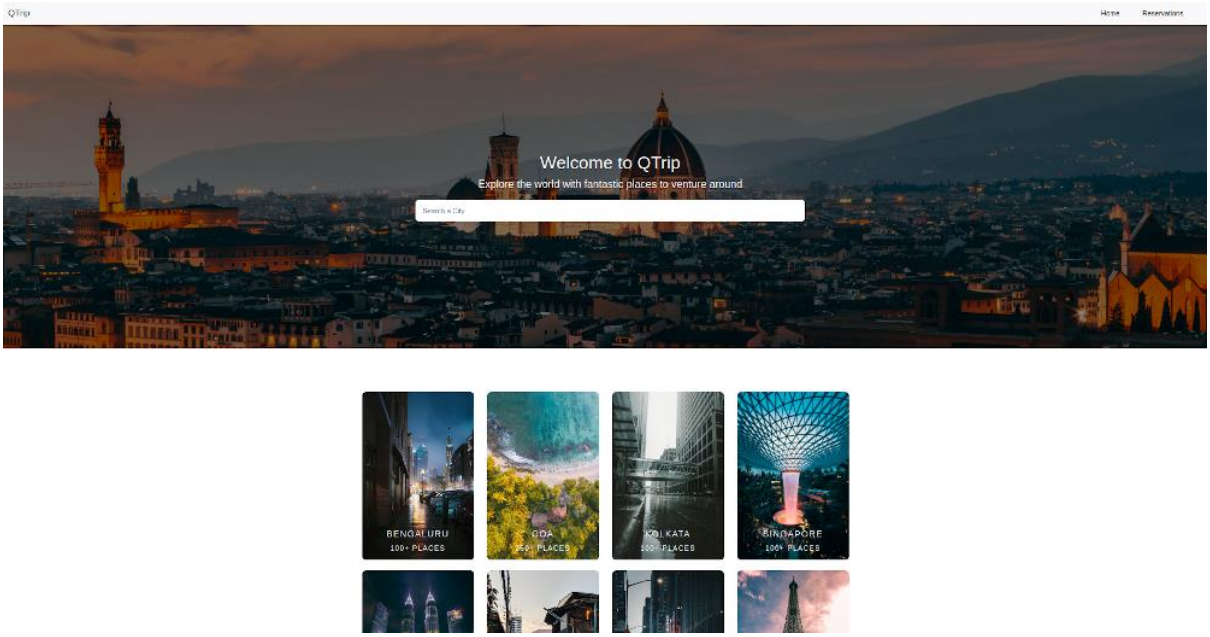
Primary goals

1. Startup the Backend and Frontend components
2. Invoke REST API to fetch City information
3. Understand backend Response and extract data
4. Populate the City cards by manipulating the HTML DOM
5. Link the City cards to the Adventures Page and pass the city param in the URL

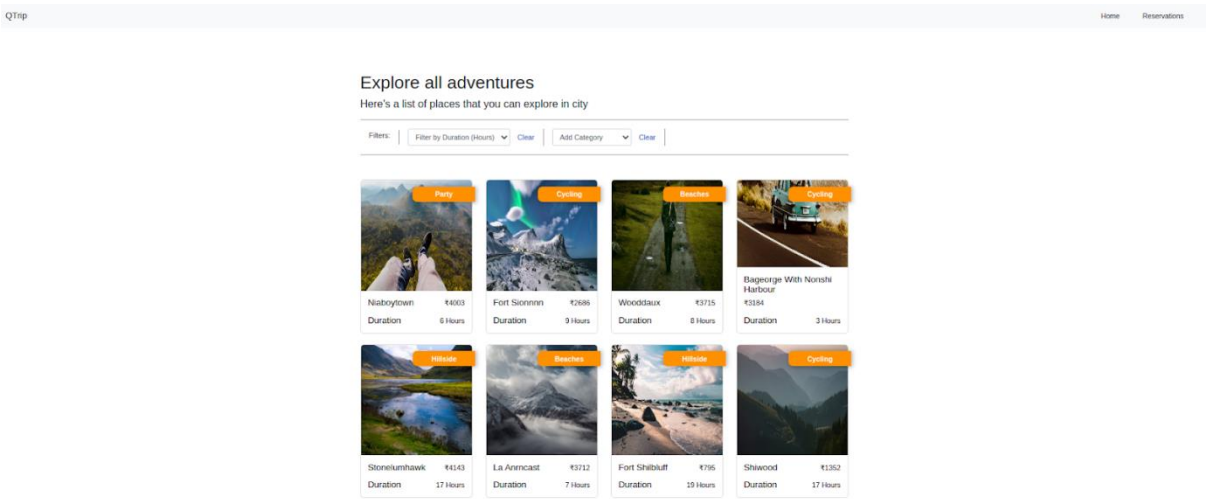
Prerequisites

This ME assumes you have basic working knowledge of JavaScript, HTML and CSS.

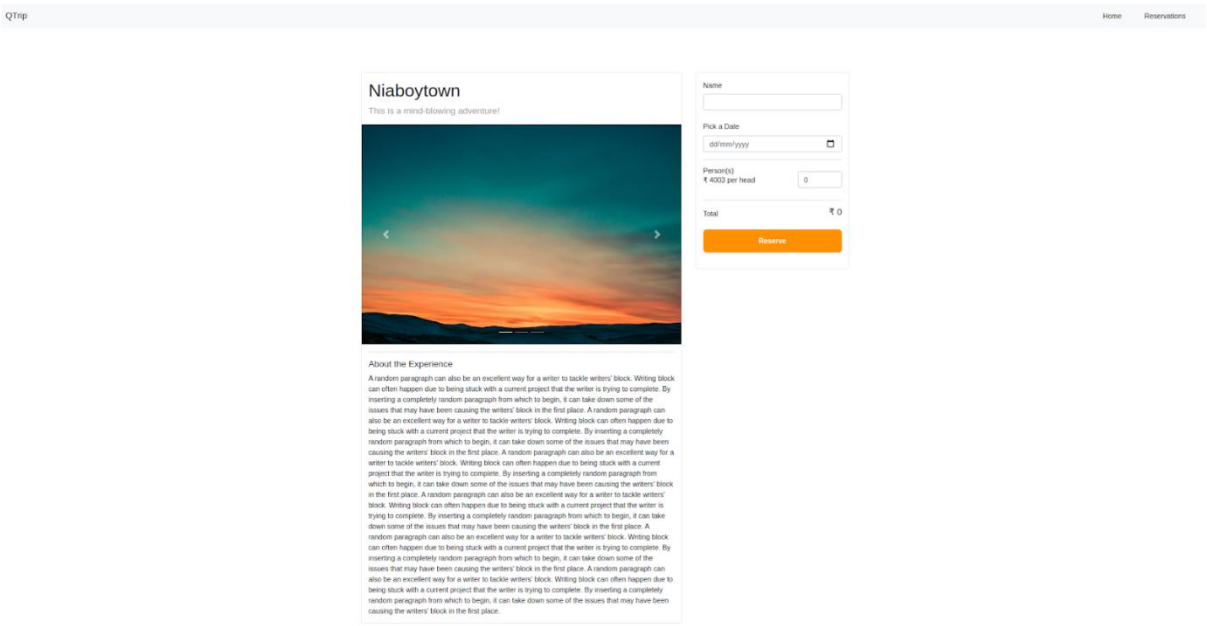
QTrip website will have these 4 pages



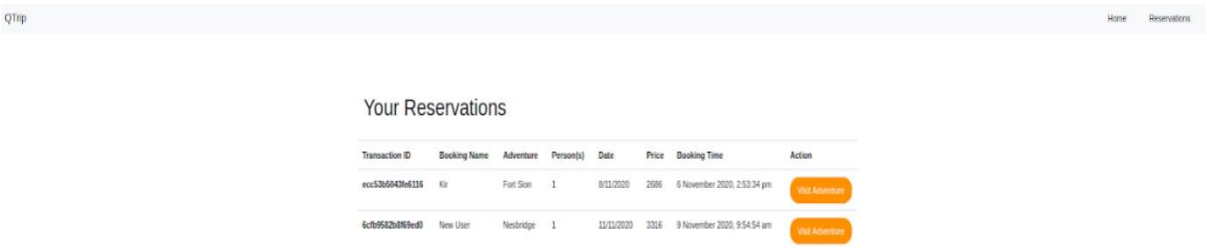
- **Landing Page** - This is the starting page for the website. It lists the various cities for which you can explore adventure



- **Adventure Details Page** - This is the page that you'll get to by clicking on one of the Adventures on the Adventures Page. It lists details about that particular Adventure.



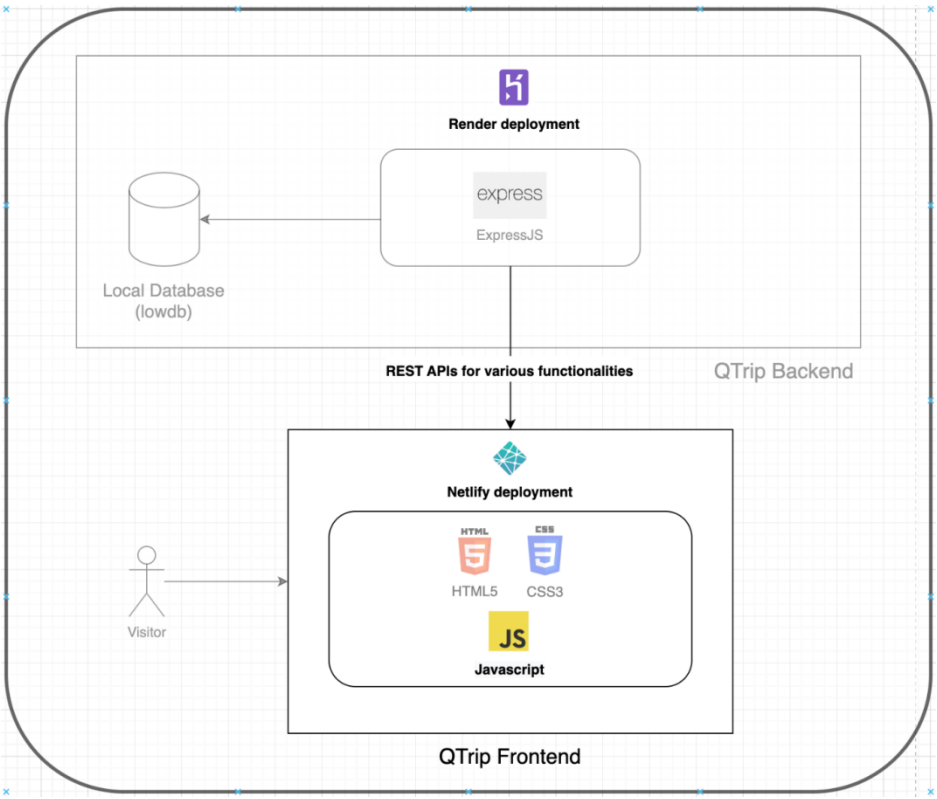
- **Reservations Page** - Once the users make reservations on the Adventure Details Pages, the Reservations Page can be used to view all the reservations.



2.2 Background

We will use JavaScript to build on top of the provided HTML and CSS files for QTrip.

- You will implement dynamic functionality such as fetching Cities, Adventures, Adventure Details and Reservations from a backend server
- The JavaScript code will fetch this information and manipulate the DOM of the HTML page to make the site interactive.



2.3 QTrip Architecture

From the architecture diagram above, we can see that there is a Frontend and a Backend component powering the QTrip site.

- The focus of this ME will be the JavaScript content for the Frontend which will use REST API to fetch the required data from the Backend
- It will also send data that needs to be stored to the Backend, using REST API. Without the Backend component, the QTrip site would only have static pages which cannot fetch or store data.
- You will also deploy the Frontend using Netlify and the Backend using Render to make your site publicly accessible.

2.4 Getting Started

Code Overview

Under the QTrip folder in your workspace, you will see this structure
You will see two directories, namely frontend and backend.

- The frontend folder consists of the UI related files (HTML, CSS and Javascript)
- The backend folder consists of the Node backend server with REST API support

In this ME, you will only be modifying the files related to frontend

Start the Frontend application using Live Server

Now that you have the overview of the code, let's get started by viewing the Landing Page.

Follow the instructions below to do that.

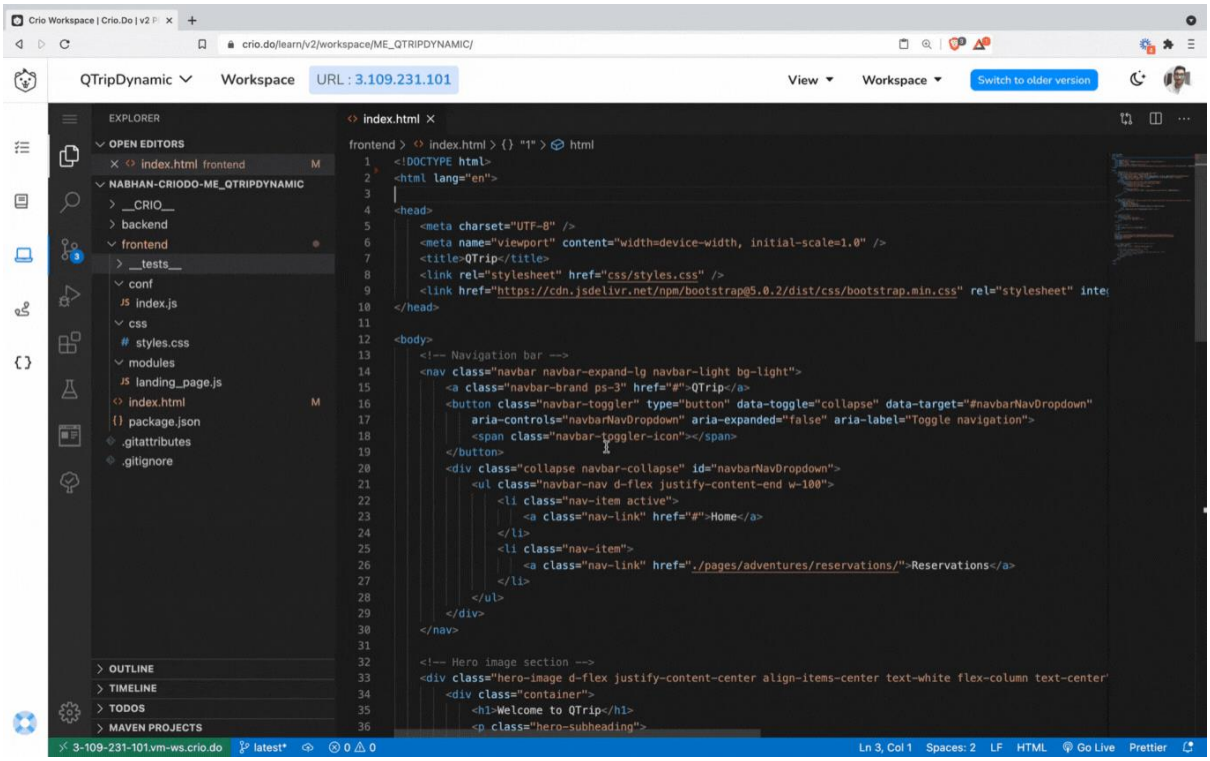
1. Right click on the index.html file at /home/crio-user/workspace/damuluru-sanjana-

ME_QTRIPDYNAMIC/frontend and click the "Open with Live Server" option.

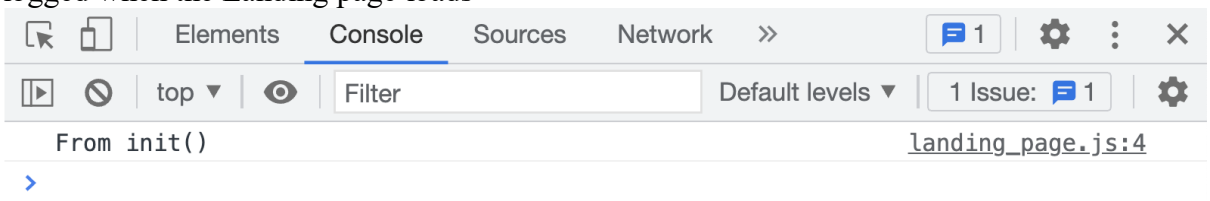
2. You'll see the "Go Live" option at the bottom of the page change to "Port: 8081".

3. Open a new browser tab and visit <your-workspace-ip>:8081/frontend to see the rendered

content of **index.html**



With the HTML file provided, the Landing Page would not have any city cards.
Add a console.log("From init()") statement inside the init() method and verify if the value is logged when the Landing page loads



2.5 Use REST API to fetch city data

Start the Backend server

The Backend server is provided out of the box. It supports REST APIs which can be invoked by the frontend to GET or POST data.

Activity

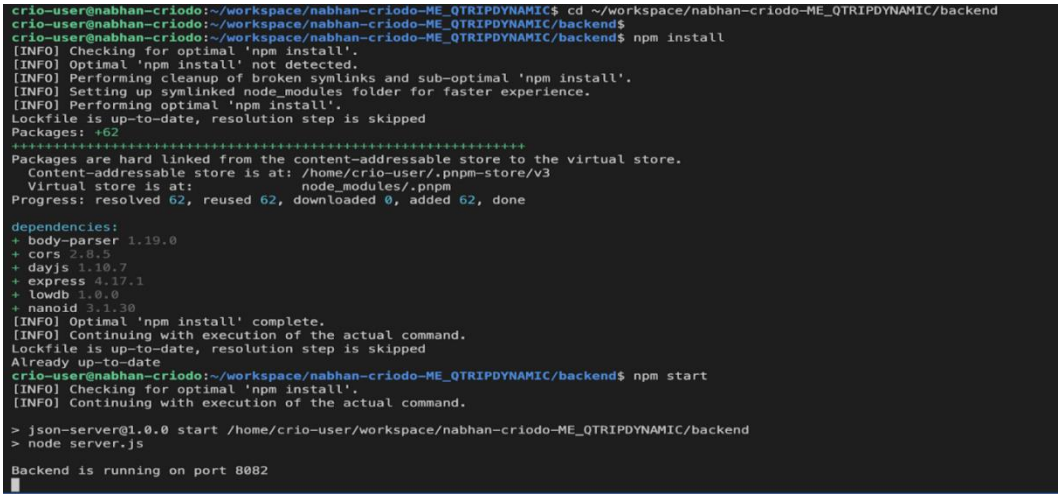
- Start the Backend server by running the npm install and npm start commands in the backend folder.

cd ~/workspace/damuluru-sanjana-ME_QTRIPDYNAMIC/backend

npm install

npm start

- This will start the server on port 8082

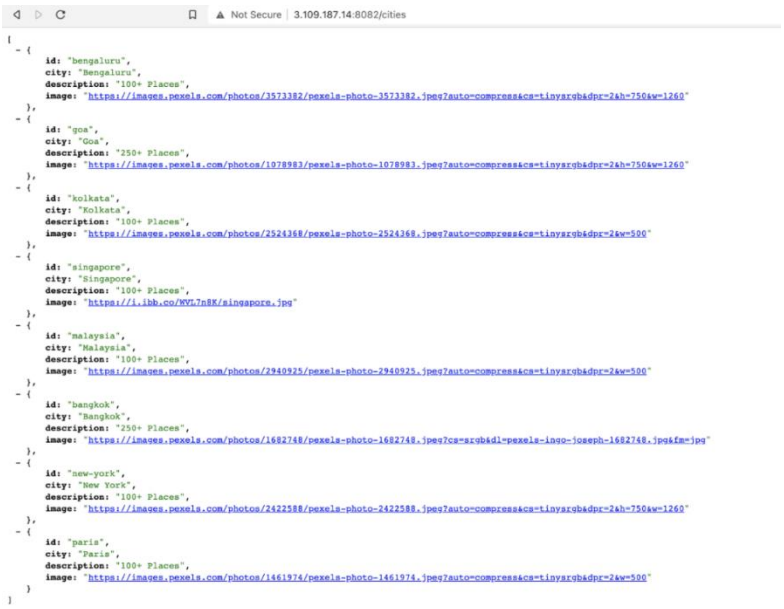


Using the Browser to verify backend server is up

As visiting a REST API URL (or API endpoint) on the browser sends a GET request, we can check if the backend server is up and running using our browser. Also, it's good practice to test the API endpoints to understand the request and response format better before using them in our programs.

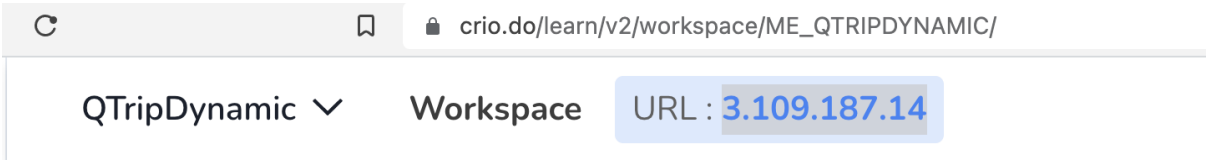
As you know, API endpoints have different components. Here's an example API endpoint (not of QTripDynamic backend)

You'll now try to make an API request to the QTrip Backend server using your browser and analyse the data returned (given below)



Response from "/cities" endpoint

- The API path to fetch Cities data is "/cities". Create the API endpoint by adding the protocol (**http**) and API root endpoint (<workspace-ip>:8082) correctly



- You should find a similar response as given above on visiting the API endpoint on your browser. Understand the JSON structure of the API response (You can use an extension like **JSONView** to format the JSON returned)
- Check out backend/server.js to get more context on this data from the comments added

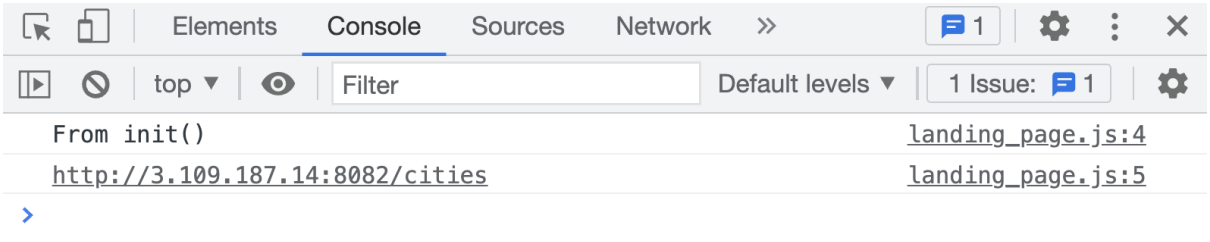
You'd have noticed that the logic to make the API call to the GET /cities endpoint is to be implemented in the frontend/modules/landing_page.js file. We'll have similar files for other

pages in the QTrip website as well. What'd happen if you were to add the API url as below in these files?

- http://3.109.187.14:8082/cities
- http://3.109.187.14:8082/adventures
- http://3.109.187.14:8082/adventures/detail
- In the frontend/conf/index.js file, you'll find the config.backendEndpoint value set to http://localhost:8082. Replace localhost with your Workspace IP. Here's an example,

```
// File: frontend/conf/index.js
const config = { backendEndpoint: "http://13.127.43.140:8082" };
export default config;
```

- You'll find the config object exported. Check out the landing_page.js file, verify if this config value is imported
- Add a log statement in the init() method to print the API endpoint to fetch the Cities data **using the** config **object** (Output format - http://<workspace-ip>:8082/cities)



Fetch Cities data from backend

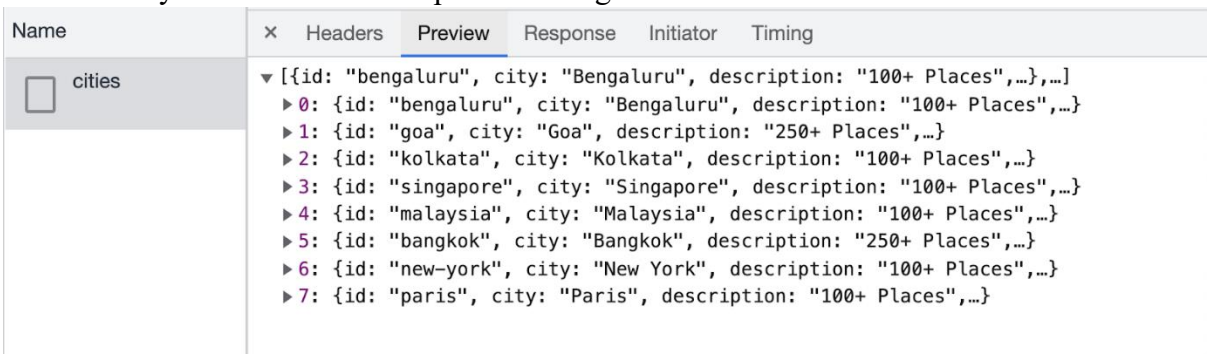
We are finally at the point where to start populating the City Cards on the Landing Page. Given that QTrip is dynamic, it has to fetch the list of cities and their information from the Backend server which exposes certain REST APIs.

The frontend/index.html file invokes the init() method
from frontend/modules/landing_page.js to achieve this. You'll find that the init() method in turn invokes two methods

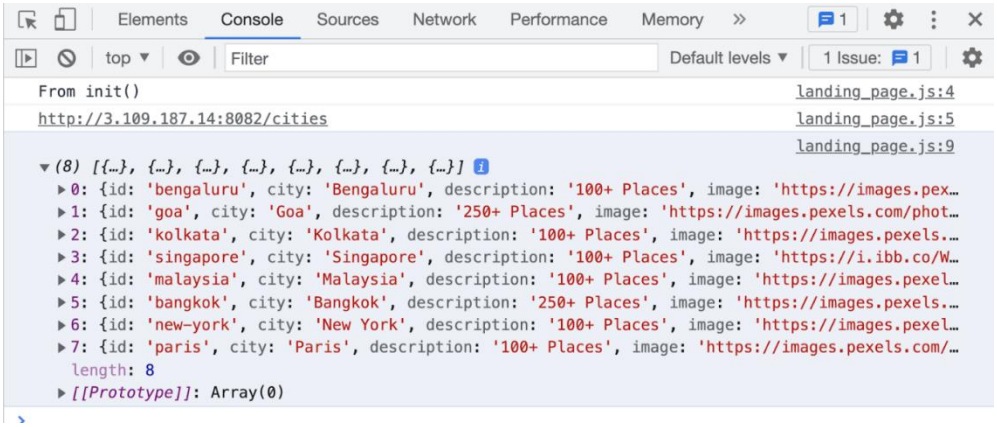
- The fetchCities() method to fetch and return the list of cities from the Backend server using an API call
- The addCityM() method to insert the city cards into the Landing Page's HTML DOM

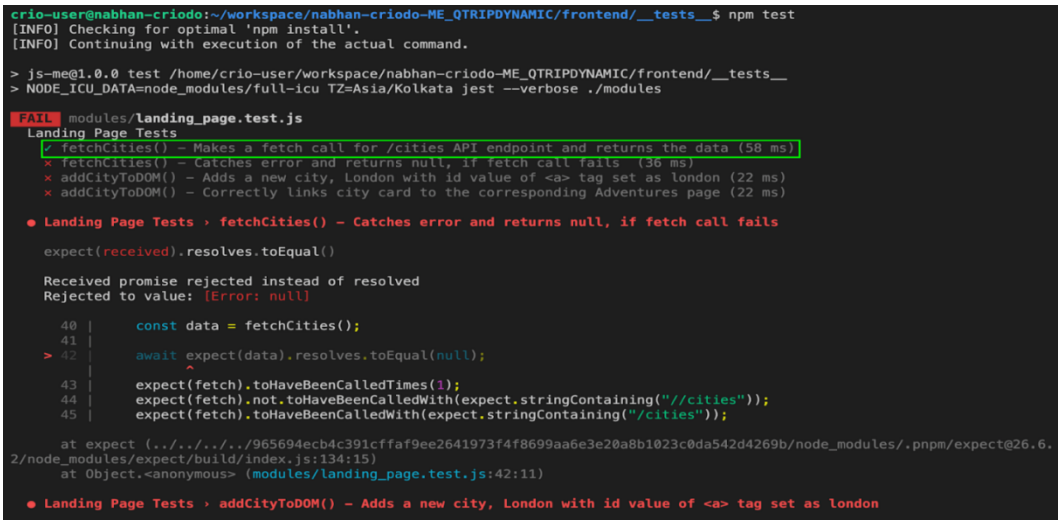
Now ,

- Check out Networks tab to verify an API call is made
- Verify the API call was successful (2xx status code) and response is as intended. If it's not,
- check if the correct API endpoint is called - http://<workspace-ip>:8082/cities
- verify Backend server is up and running



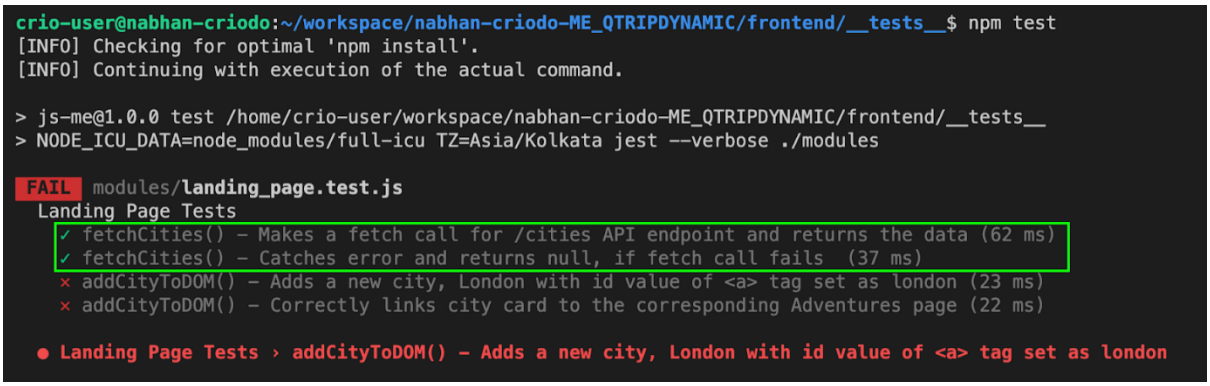
- Add a log statement to print the cities variable value in the init() which stores the value returned by the fetchCities() function. Verify this is the same as the API response from above.





Test you fetchCities() implementation

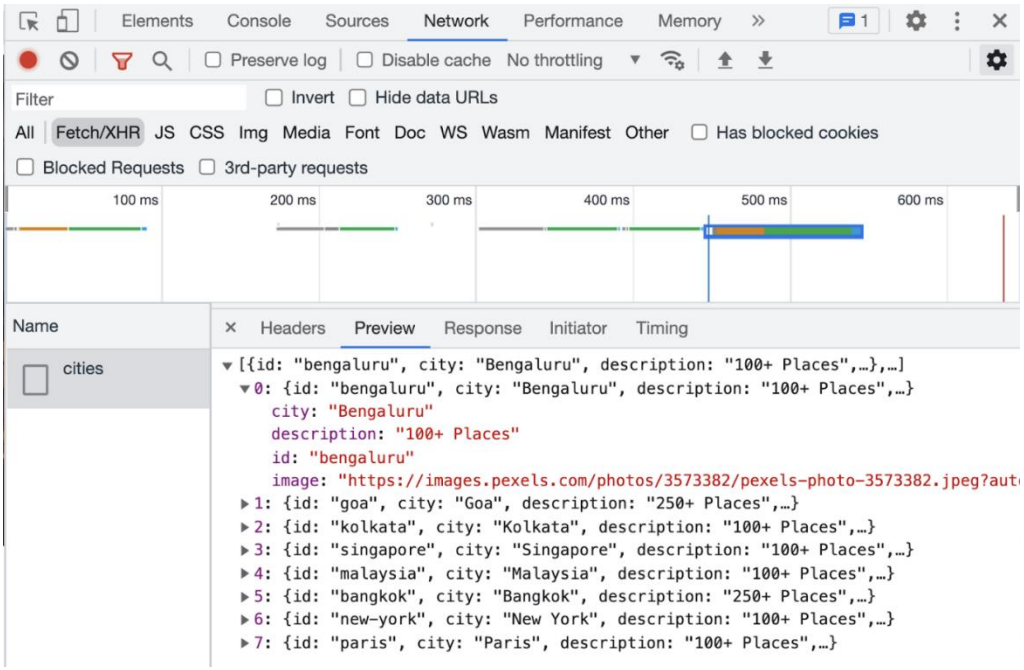
It is time to test if your fetchCities() implementation is done correctly. Execute the test cases by running npm test in the frontend/__tests_ folder. You should see all the fetchCities() function related tests passing now



2.6 Insert City Cards into DOM

Understand the Data Format of Response

You now have the Cities data from the API response. Before you jump into parsing and extracting the required fields, it's good to understand the data and its format. You can use the Networks tab for viewing the data as you did earlier.



The fields of interest are id, city, description and image

Iterate over Cities data and insert into HTML DOM

You will see the init() method iterating over the list of cities returned by fetchCities() and invoking the addCityM() for each of these cities

```
async function init() {
  let cities = await fetchCities();
          cities.forEach((key) => {
    addCityM(key.id, key.city, key.description, key.image);
  }); }
```

Complete the addCityM() function such that

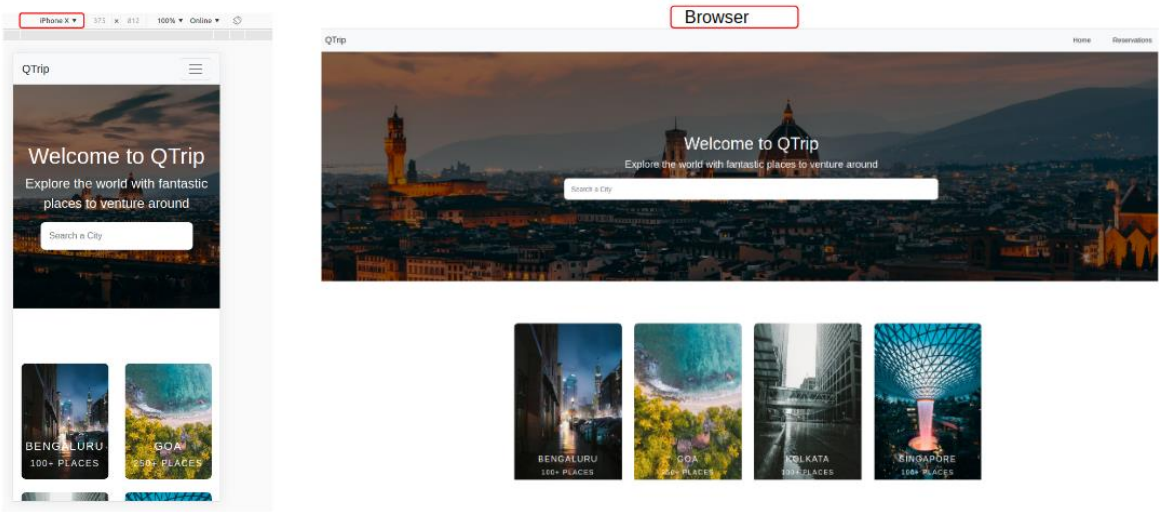
- A responsive City card is created with the data from the arguments passed to the function, and then
- Inserted to the “Content section” in the HTML as children of the div element with id, data

```
<!-- File: frontend/index.html →
      <!-- Content section -->
<div class="container">
   <div class="content text-white">
      <div class="row" id="data"></div>
   </div>
</div>
```

- Clicking on this card should redirect a user to the Adventures page for that city

- The link should be in this format - pages/adventures/?city=<id> where id is the parameter passed to this method

The end result will look like this



1. **Create the cards for adventures grid page**

   Background

   The QTrip Landing Page is complete. In this module, we'll focus on the Adventures Page. Clicking a city card on the Landing Page navigates users to this page, which lists adventures for that city.

- The city ID is extracted from the URL, and adventures are fetched via a REST API request.
- Adventure cards are populated based on the API response.
- Clicking an adventure card directs users to the Adventure Details page.

Objective : Display content of the Adventures page dynamically

Primary goals

In this module, we will:

1. Extract City Id from the URL and make an API call request to the backend
2. Understand the response and extract data for list of adventures
3. Populate the adventure cards by manipulating the HTML DOM

4. Link the adventure cards to the Adventure Details Page and pass the adventure id in the URL

2.7 Resolve Merge Conflicts

There could be merge conflicts when you pull stubs for a new module. Please resolve all merge conflicts before proceeding. Choose from one the following resolution steps:

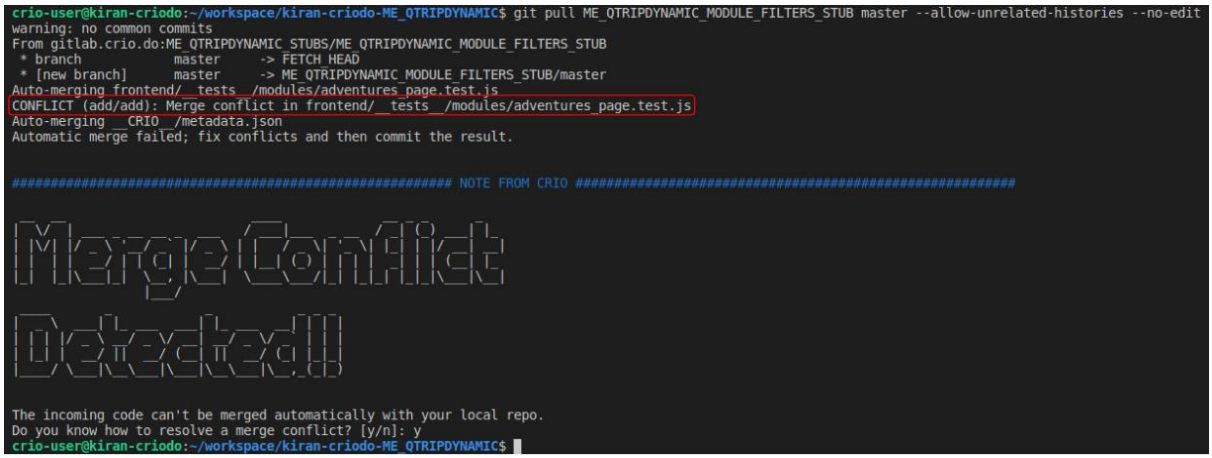
1. Accept Current Change
2. Accept Incoming Change
3. Accept Both Changes
4. Compare Changes

You could lose your work if you choose the wrong option. Please watch the following video to understand these options better and choose the appropriate one.

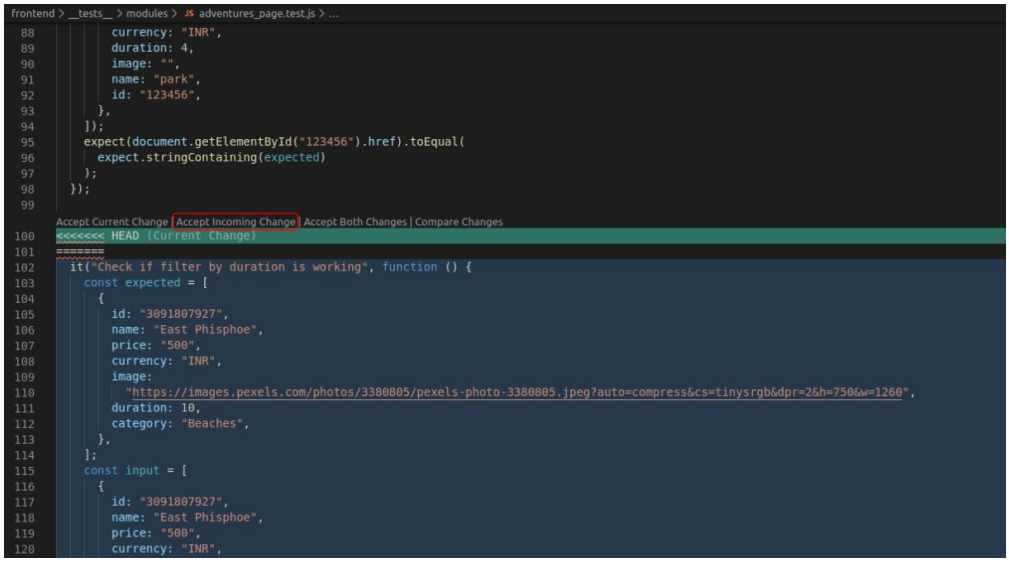
How to resolve a merge conflict?

In case you see merge conflicts for any files, resolve them as needed. For files you aren't to make any updates like (metadata.json,.test.js files and backend files), accept the incoming changes.

You would see something like this. Press y and complete it.



Here, one of the files you see conflict with is the frontend/__tests__/modules/adventures_page.test.js. Open this file in VS Code and as it is a test file, click on Accept Incoming Change as shown here



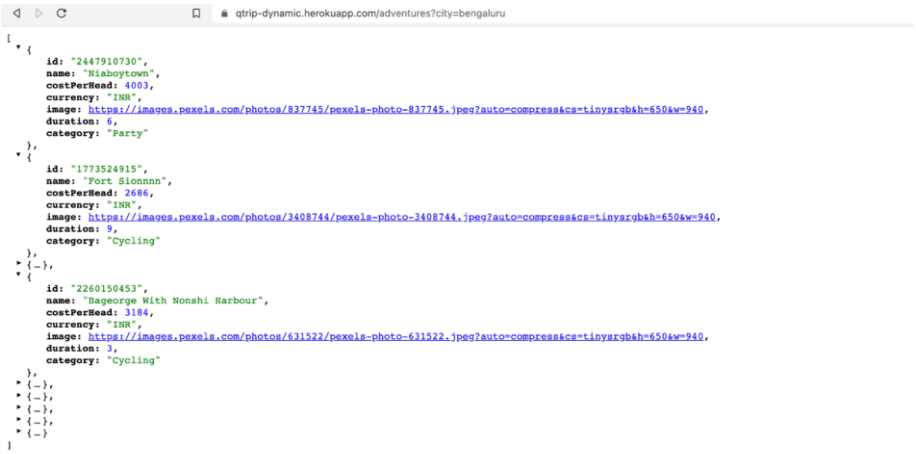
2.8 Backend

Milestone 2 : Use REST API to fetch adventures

Fetch information about adventures

To fetch adventures for a city, modify the fetchAdventures() method:

- Use the backendEndpoint and /adventures?city=<city_name> API endpoint, where city_name comes from getCityFromURL().
- Retrieve the JSON data from the Backend's response.
- Review the response fields using tools like curl, your browser, or Chrome Dev Tools' Network tab.
- Return the JSON data for further use in the next milestone.



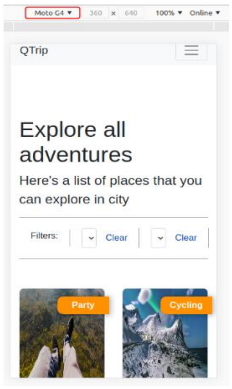
Response from "/adventures" endpoint

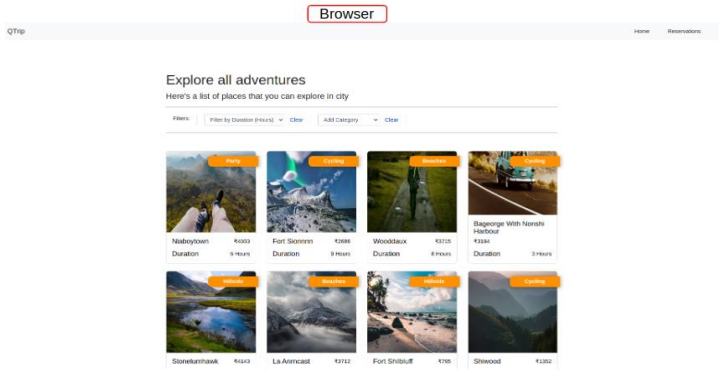
Milestone 3 : Insert Adventure cards into DOM

Iterate over Adventures and insert into HTML DOM

The method that needs to be filled in is addAdventureME()
This method has to take a list of adventures as input and insert the adventure cards into the DOM. Key things to note:

1. Use these fields from the adventure information to create the card
   - id, category, image (URL), name, costPerHead and duration
2. Provide a link to the adventure cards such that users can click on a card to get to the Adventure Details pages for that adventure. The link should be in this format
   - detail/?adventure=<adventure_id> where adventure_id is extracted from the input.
3. Use these classes from the provided frontend/css/styles.css file
   - category-banner
   - activity-card
   - activity-card img
4. Use divs, justification and flex as needed to create the cards shown below





2.9 DOM Manipulation
DOM manipulation is a key aspect that JavaScript is good at. This makes pages dynamic and capable of user interaction. With these modules you are now capable of making DOM changes to the HTML pages. You will continue to explore some more capabilities of JavaScript in a web application in the upcoming modules.

2. Create filters for adventures

Objective
In this module, we will:
1. Implement filters for adventures based on Duration and Category
2. Implement local storage to persist with the selected filters
3. Learn about filters in JavaScript, event handlers and local storage

Milestone 1 : Filter by Category

Fitler related methods

In this module, you'll be implementing these filter and local storage related methods in frontend/modules/adventures_page.js

1. filterByDuration()
2. filterByCategory()
3. filterFunction()
4. saveFiltersToLocalStorage()
5. getFiltersFromLocalStorage()
6. generateFilterPillsAndUpdateDOM()

In addition to this you'll also fill in the following methods in frontend/pages/adventures/index.html

1. selectDuration()
2. clearDuration()

Implement filter by Category

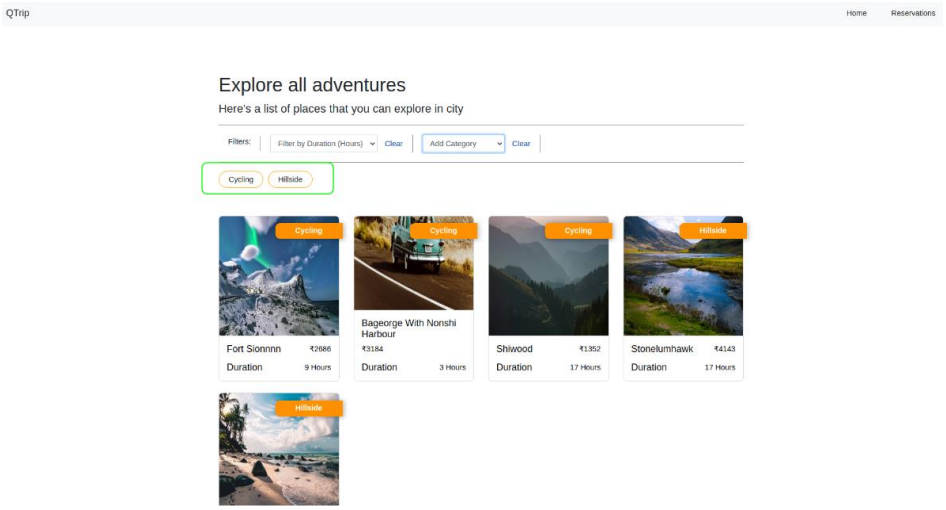
To implement the filterFunction() and filterByCategory() methods:

- filterFunction() is invoked by index.html to ensure only adventures that match the filter criteria are displayed.
- A global variable filters is used across the page. Review the HTML to understand its structure.
- filterFunction() handles all filters (Category, Duration, or both). For now, focus on invoking filterByCategory() when a Category filter is present.
- filterFunction() takes filters and the list of adventures as inputs. If the category filter is present, extract the category list and pass it, along with the adventures list, to filterByCategory(), which will return the filtered adventures.
- Return the filtered list of adventures from filterFunction().
- If no filters are provided, return the entire list of adventures.

You will notice that the clear filter functionality for Category filter is already functional. Go through the code in the html file to see how this is implemented in the selectCategory() and clearCategory() methods.

Populate the filter pills

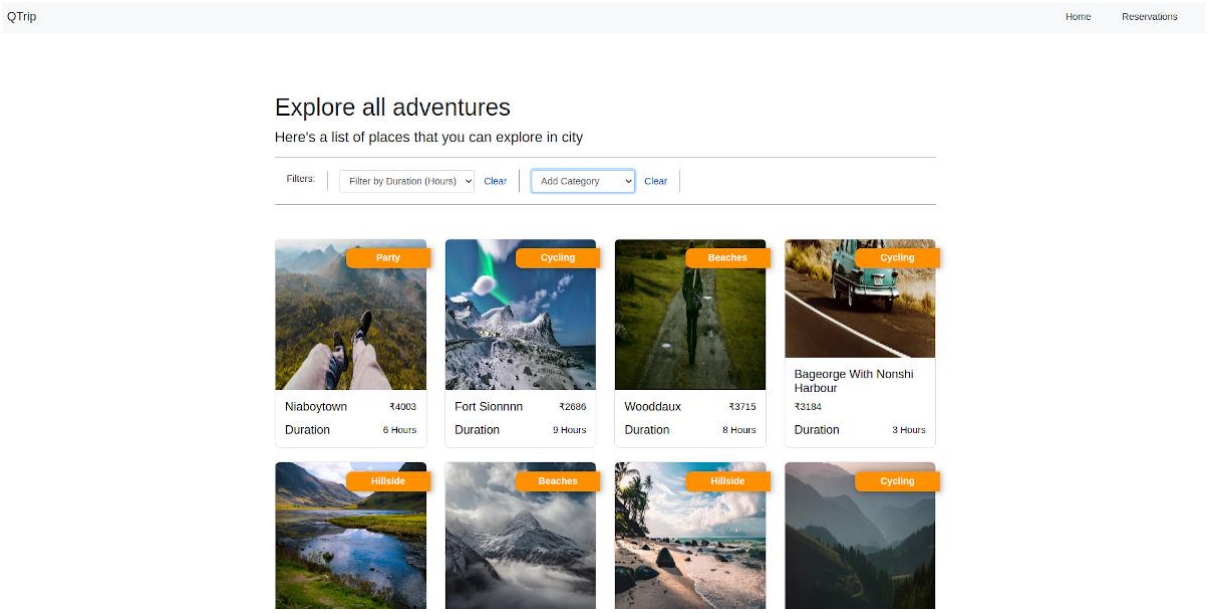
The highlighted section (in a green rectangle) in this snapshot is called "pills". The Category filter can have multiple filters selected and the purpose of having this is to show the user what filters have been selected



This has to be implemented in the generateFilterPillsAndUpdateDOM() method

- Check the input filters passed and for every category filter, insert an element into the DOM
- Insert them within the class with id category-list in the html file
- Use the category-filter css class provided in frontend/css/styles.css
- Clicking on "Clear" for the filter should remove all the pills

Ensure that your page looks like the image shown above, when Category filters are selected and they get removed when filters are cleared as shown below.



Milestone 2 : Filter by Duration

- Implement filter by Duration

You need to write the code for the method filterByDuration() and modify filterFunction() to invoke this when a Duration filter is selected.

Some observations:

- filterFunction() should handle all conditions
- Category filter
- Duration filter
- Category and Duration filter
- No filter
    - In this milestone you'll add support for filterByDuration()
    - In filterFunction(), if the duration filter is present in the filters, extract the low and high duration from it and pass these along with the list of adventures to filterByDuration() and that method should return the filtered adventures that have a duration falling between low and high.
    - Finally, return the filtered list of adventures from filterFunction()

- Implement selection and clearing of Duration filter

You may have noticed one thing. Even though the test cases pass for the Duration filter, it doesn't work on your web page!!

This is because there is a disconnect between the event of filter selection on the html page to the JS method that you've implemented. Let's fix this.

The methods that have to be populated for this to work are in the frontend/pages/adventures/index.html file. These are the methods you need to populate - selectDuration() and clearDuration(). Check when these are being invoked.

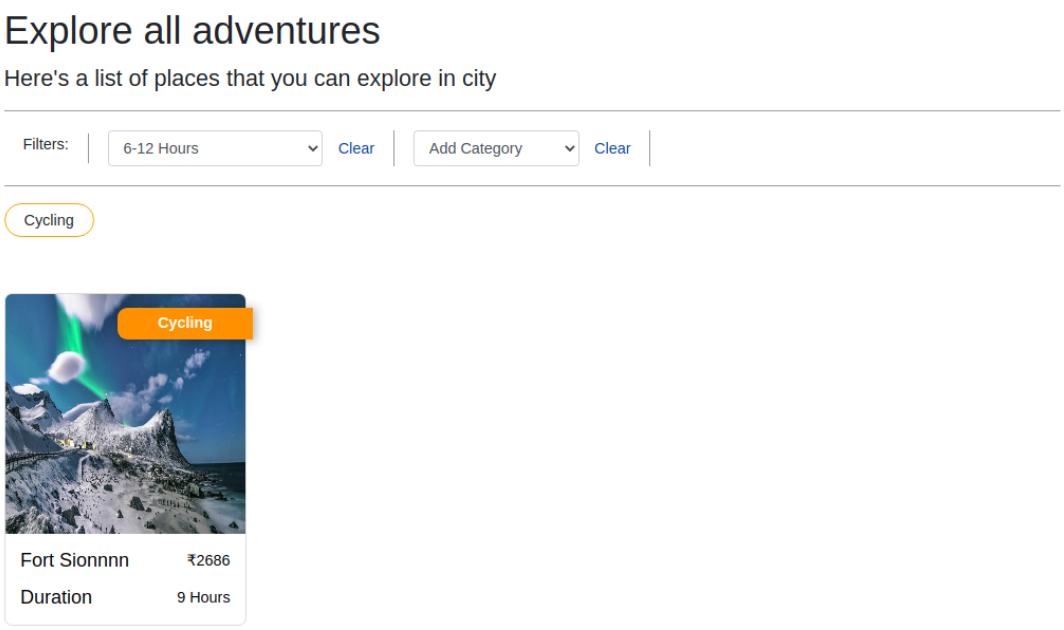
Some observations about selectDuration()

- Extract the filter value from the event.
- Invoke filterFunction() to apply the filter value
- Invoke addAdventurem() to update the HTML page
- If the functionality doesn't work correctly, try to compare this method to the selectCategory() method and see what could be missing.

Some observations about clearDuration()

- Cleanup the DOM to show nothing selected for this filter
- Reset the filters global variable for the duration filter
- Invoke filterFunction() to apply the new cleaned up filter value
- Invoke addAdventurem() to update the HTML page
- If the functionality doesn't work correctly, try to compare this method to the clearCategory() method and see what could be missing.

Now test the duration filter functionality on the web page. When you set the duration filter alone or in combination with category filter, only adventures satisfying those filters should be displayed. Example:



Milestone 3 : Data Persistence

Data Persistence

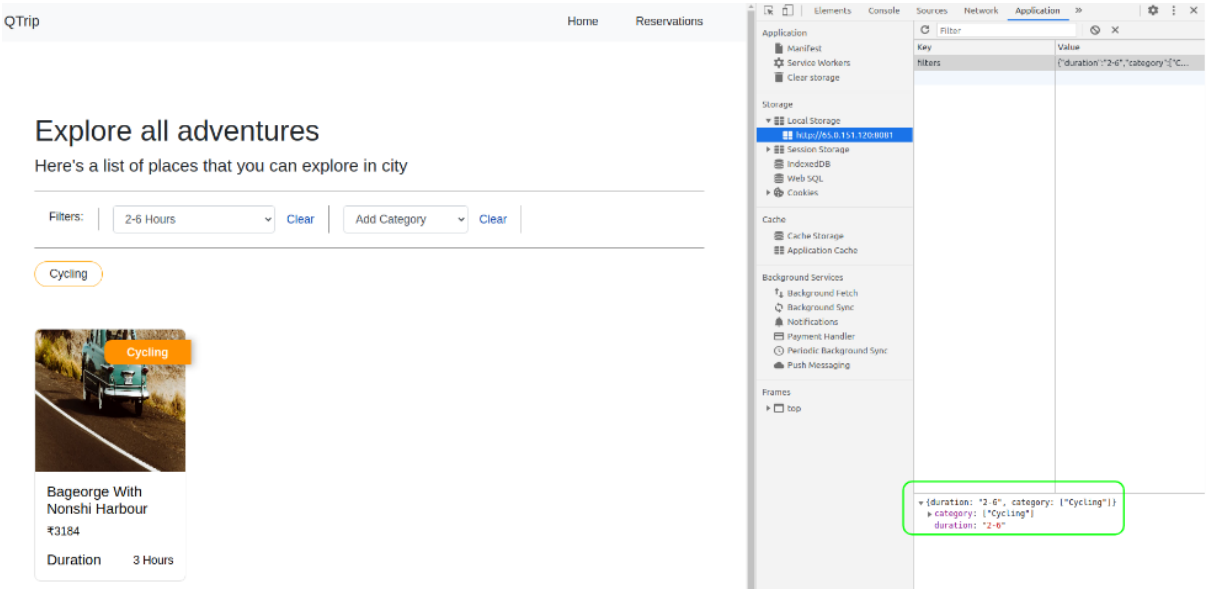
When users visit the QTrip website and specify filters for adventures, they should be able to retain their filter selection even if the page is refreshed or the website is opened in a different tab. The way to achieve this is through data persistence. The filter selection needs to be stored in the browser's memory. Read through the references provided below to understand the various options available for data persistence, before you proceed further.

Save and get filters from local storage

You have to implement these methods to persist and retrieve the filters
- getFiltersFromLocalStorage() and saveFiltersToLocalStorage()

- Use JSON.stringify() while storing filter data into localStorage in saveFiltersToLocalStorage() since local storage can only store string data.
- Use JSON.parse() to convert the string data from localStorage back to JSON format in getFiltersFromLocalStorage() and return the data

You can also see the local storage filters using **Chrome Dev Tools**



Filter functionality

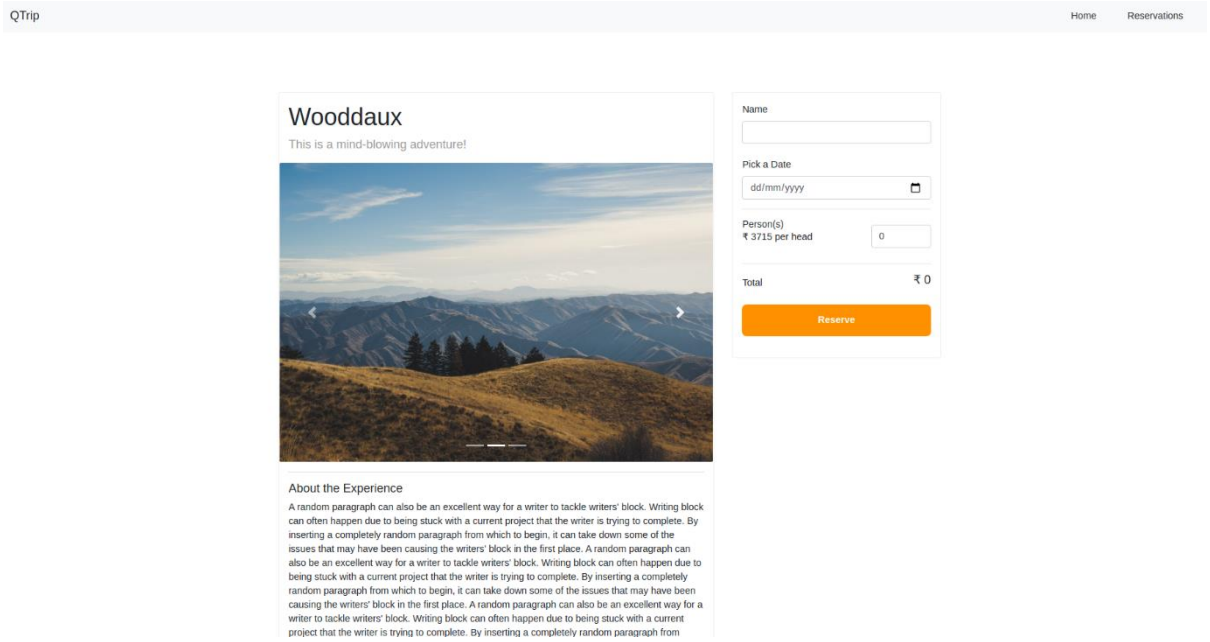
Providing filters using a drop down is a common requirement on web pages. You have learnt how to visually present filters and allow users to select them. Further, the user selection has to be visible so user knows what's been filtered. The next thing is to apply these filters everytime the selection changes using HTML and JavaScript to achieve the necessary result.

2.10 Create the adventure details page

Objective

The first two pages of QTrip are complete, namely, Landing Page and Adventures Page. Now, it's time to address the Adventure Details Page. Users can get to this page by clicking on a particular adventure card on the Adventures Page.

The Adventure Details Page contains details about the adventure along with an image gallery on a carousel. There is a reservation panel where users can put in their details and make a reservation. They can then view their reservations by going to the Reservations Page. We will address the adventure details in this module and handle the reservations in the next module.



The adventure id will be extracted from the URL and the details for that adventure will be fetched with a REST API request to the backend server. The adventure details will be populated and displayed from the contents of the API response.

Primary goals

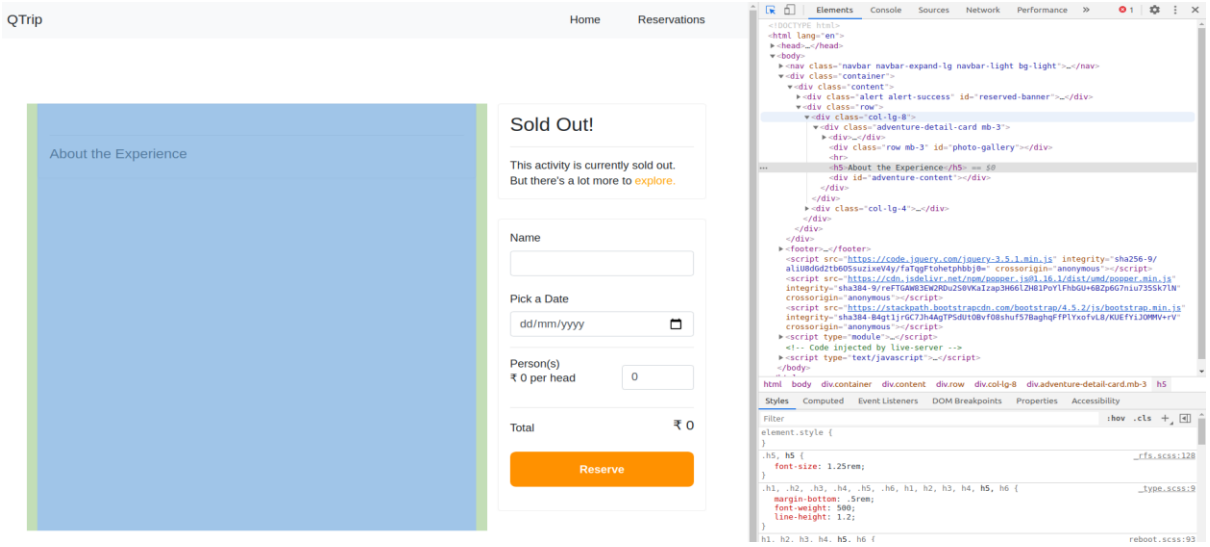
In this module, you will:

1. Extract Adventure Id from the URL and make an API call request to the backend
2. Understand the response and extract data for the adventure details
3. Populate the adventure details by manipulating the HTML DOM
4. Create an image gallery on a carousel using bootstrap

Milestone 1 : Extract parameters from URL and invoke API

Code Overview

The frontend/pages/adventures/detail/index.html file represents the Adventure Details page. It currently looks like this. Inspect the elements to understand the code and the structure.



Let’s make some observations here

1. The page has a header, footer and a navbar all of which are responsive

2. There is a section on the left to represent the adventure details along with images. It has some placeholders that need to be populated with details.

3. There is a section on the right to represent the reservation panel. It also displays a "Sold Out" section that should be conditionally rendered.

4. JS methods have been imported from frontend/modules/adventure_details_page.js. Spend a few minutes reading through the comments in this file to understand what’s expected of these methods.

    - getAdventureIdFromURL()
    - fetchAdventureDetails()
    - addAdventureDetailsM()
    - addBootstrapPhotoGallery()
    - conditionalRenderingOfReservationPanel()
    - calculateReservationCostAndUpdateDOM()
    - captureFormSubmitUsingJQuery()
    - showBannerIfAlreadyReserved()

5. The methods are invoked in the required order to populate the page.
6. An event trigger based method invocation is also present

Extract the Adventure Id from URL

Let's start filling in the methods one by one.

The first method to be addressed is getAdventureIdFromURL()

Let’s make some observations here

- This method is invoked from the html file.
- There is a search string parameter being passed to it. Use this input string and extract the adventure id from it. Return this adventure id.
-

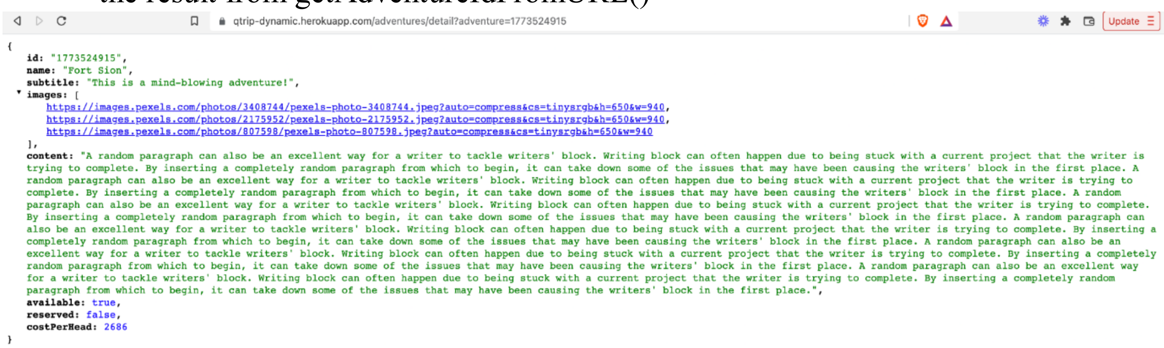
Fetch adventure details

Fetching the details for the adventure currently selected by making a REST API call to the Backend Server.

You will have to modify the fetchAdventureDetails() method to do this.

Some observations

- Populate the fetchAdventureDetails() method to invoke the Backend’s REST API using the configured backendEndpoint and
  the /adventures/detail?adventure=<adventure_id> API endpoint, where adventure_id is the result from getAdventureIdFromURL()



Response from the "/adventures/detail" endpoint

Milestone 2 : Insert Adventure Details into the DOM

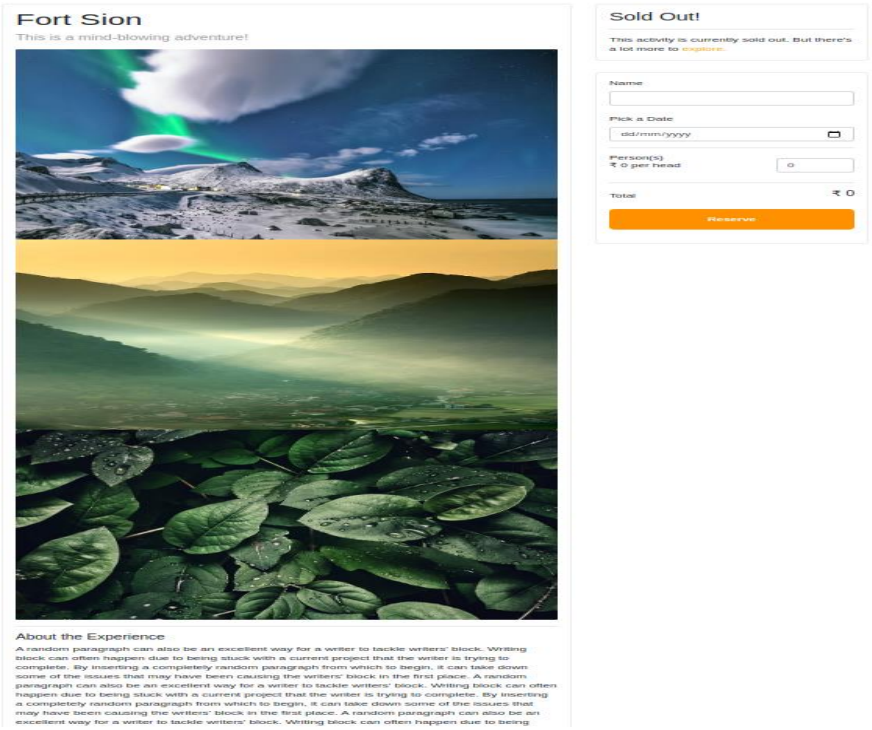
Extract Adventure details and insert into the DOM

The method that needs to be filled in is addAdventureDetails()
name, subtitle, images, content.

This method takes an adventure as input and inserts the adventure details into the DOM. Key things to note:

1. Use these fields from the adventure details to populate the DOM
   - name, subtitle, images, and content
2. Get the HTML elements where these need to be inserted by using their element ids
3. There could be more than one image for this adventure. Loop through the images, create a div element for each and insert it into the photo-gallery

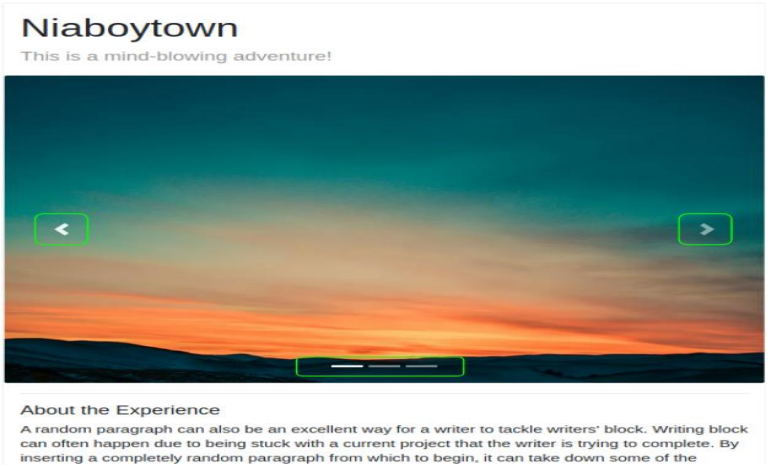
4. Use the activity-card-image class from the provided frontend/css/styles.css file for the images
5. The end result should look like this. As you can see, the images would be one below the other.



Milestone 3 : Add bootstrap photo gallery

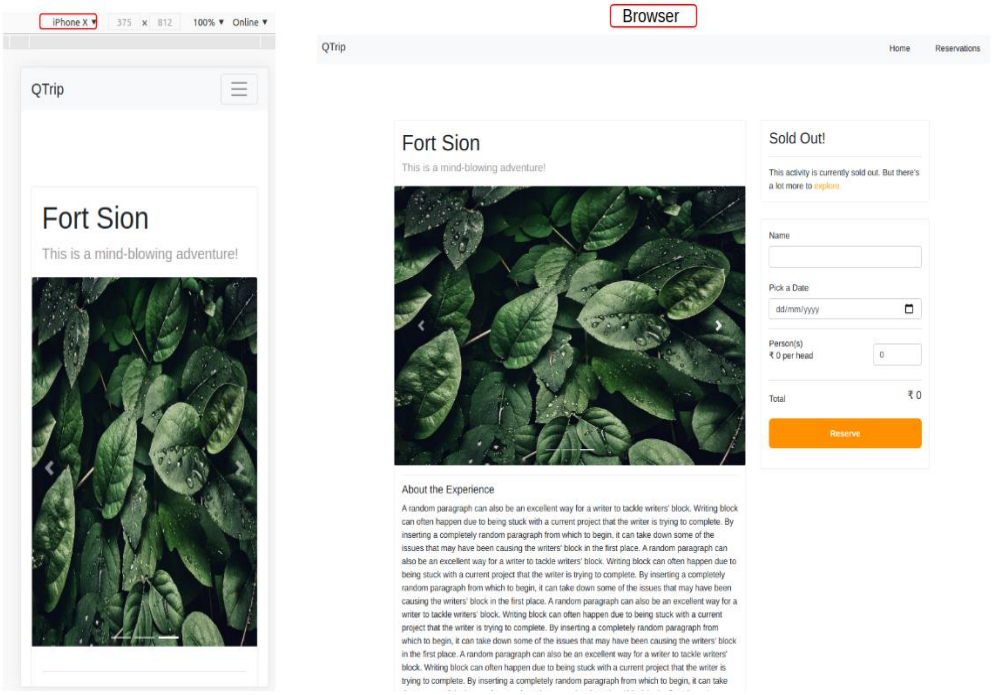
Add bootstrap photo gallery

We see that the images in the gallery are one below the other. We need to add the images to a carousel so that one image can be seen at a time and users can go through the images by clicking on the arrows on the carousel.



You will be updating the method addBootstrapPhotoGallery() for this

- Update the HTML content of the photo-gallery element using bootstrap's carousel component. Check the references provided below.
- Loop through the images provided as input, create a div element for each and insert it into the carousel-inner element of the bootstrap carousel.
- Make sure only one of the images (carousel-item) is set to active.
- The end result should look like this and it should be responsive



2.11 Add support for adventure reservations

Objective

The Adventure Details page now needs to support reservations. In this module, you'll enable this using the reservation form on this page. Users will be able to make reservations and the reservation data will be sent to the Backend server using a POST REST API call.
Also, you'll add functionality to the Reservations Page, which will fetch all user reservations from the Backend server and display them using the appropriate date format.

Primary goals

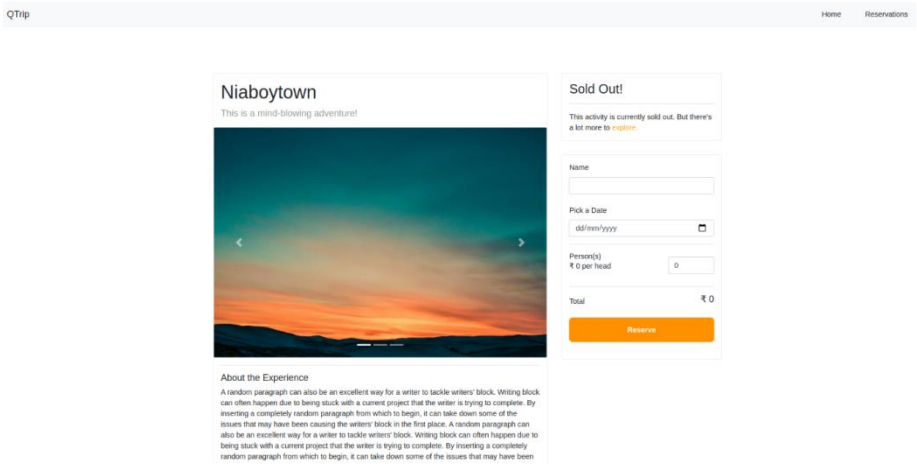
In this module:

1. Capture data from the reservation form
2. Make a POST API call to store reservation data, using Fetch API
3. Conditional rendering of reservation section based on user having booked an adventure
4. Fetch all reservations and update the Reservations Page DOM
5. Learn these skills - Forms, POST request, and JavaScript date functions

Milestone 1 : Conditional rendering and updates to reservation form

Code Overview

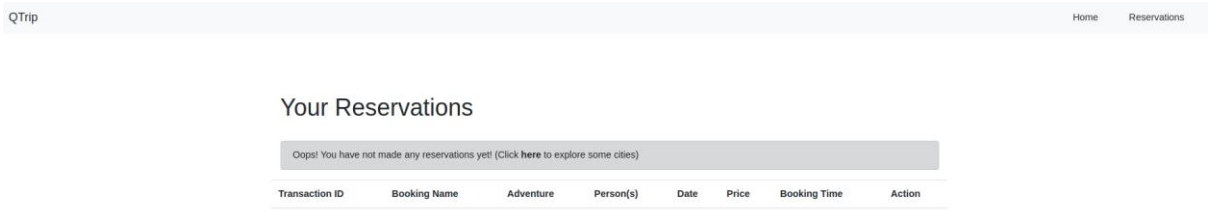
The frontend/pages/adventures/detail/index.html file represents the Adventure Details page. It currently looks like this



You can see that there are two sections shown on the right - the "Sold Out!" section and the Reservation form. But, only one of these should be shown. If a reservation has already been made for this adventure, you should show the Sold Out section, else show the Reservation form.

These are the methods to be updated in this module to get the reservations working

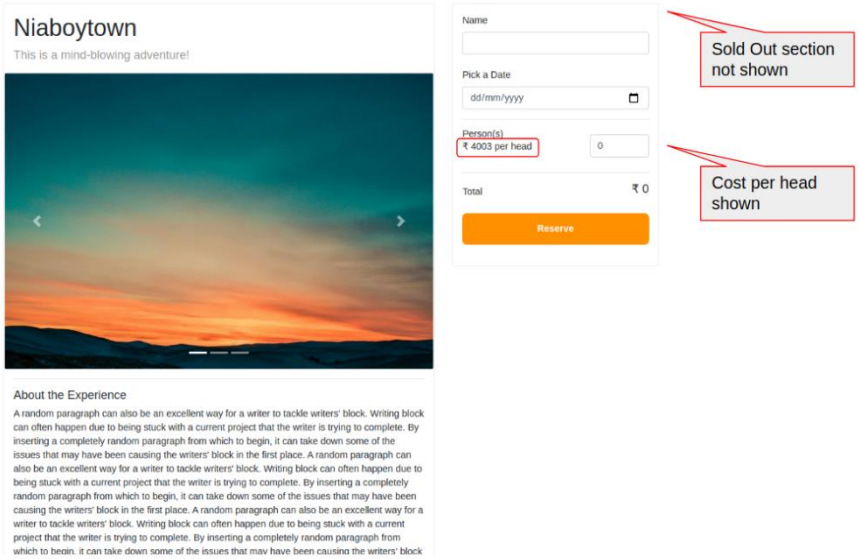
1. conditionalRenderingOfReservationPanel()
2. calculateReservationCostAndUpdateDOM()
3. captureFormSubmit()
4. showBannerIfAlreadyReserved()
   The frontend/pages/adventures/reservations/index.html file represents the Reservations page. You can get to this by clicking on the Reservations button on the navigation bar. It currently looks like this



Conditional rendering of reservation panel

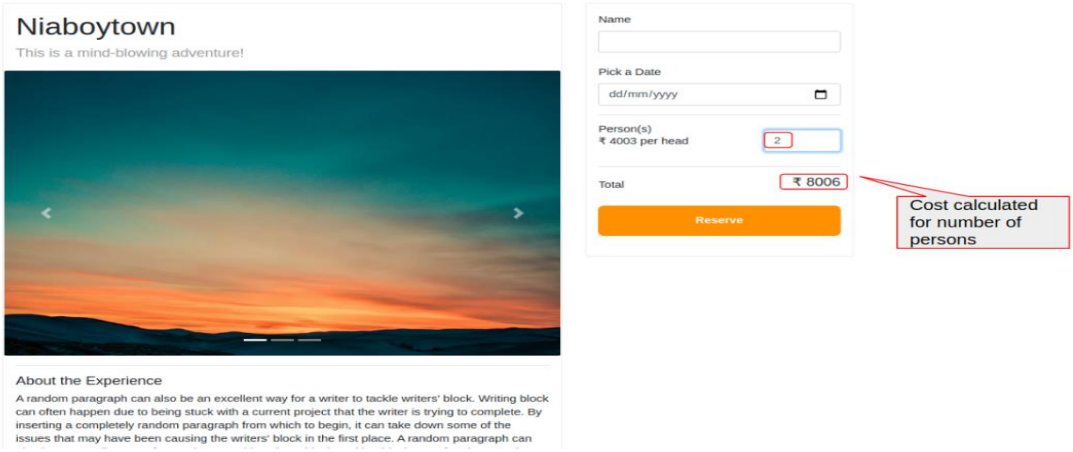
The first method to be addressed is conditionalRenderingOfReservationPanel()

```
crio-user@kiran-criodo:~/workspace/kiran-criodo-ME_QTRIPDYNAMIC/frontend/__tests__$ npm test

> js-me@1.0.0 test /home/crio-user/workspace/kiran-criodo-ME_QTRIPDYNAMIC/frontend/__tests__
> NODE_ICU_DATA=node_modules/full-icu jest --verbose ./modules

 FAIL  modules/adventure_details_page.test.js (9.354 s)
  Adventure Detail Page Tests
    ✓ Extract city from URL Params (114 ms)
    ✓ Check if fetch call for the adventure details was made and data was received (44 ms)
    ✓ Catches errors and returns null (30 ms)
    ✓ Tries adding a Adventure Details - Park (40 ms)
    ✓ Check if bootstrap gallery is working (43 ms)
    ✓ Check if conditional rendering is working (29 ms)
    ✕ Check if reservation cost is calculated correctly and updated in DOM (31 ms)
    ✕ Check if reservation banner is displayed (22 ms)
    ✕ Check if JQuery form submission is taking place (76 ms)
```



Calculate reservation cost and update DOM

You have to modify the calculateReservationCostAndUpdateDOM() method to achieve this.



Milestone 2 : Make a reservation

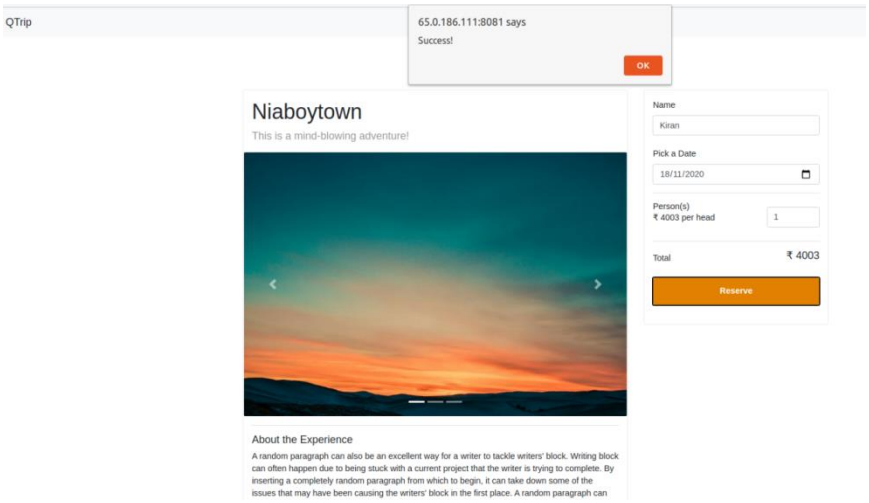
Capture data from reservation form and make POST request using Fetch API

The method that needs to be filled in is captureFormSubmit()
Key things to note while implementing this method:

1. Capture form data upon submit. Use the myForm id and handle the submit action for the form by adding an event handler
2. Make a POST API call to send this data to the Backend server using **Fetch API**

- Data to send in body should have 4 properties - name, date, person, adventure
- name, date and person from the Form data
- adventure is the id of the current adventure
- URL to use - configured backendEndpoint and the /reservations/new API endpoint
- If the reservation is successful, as seen in the response, show an alert with Success!
- Also, you should refresh the page if the reservation is successful.
- If the reservation fails, as seen in the response, show an alert with Failed!
- Note that this is the first time you'll be making a POST request in this ME. Till now you had been making GET requests.

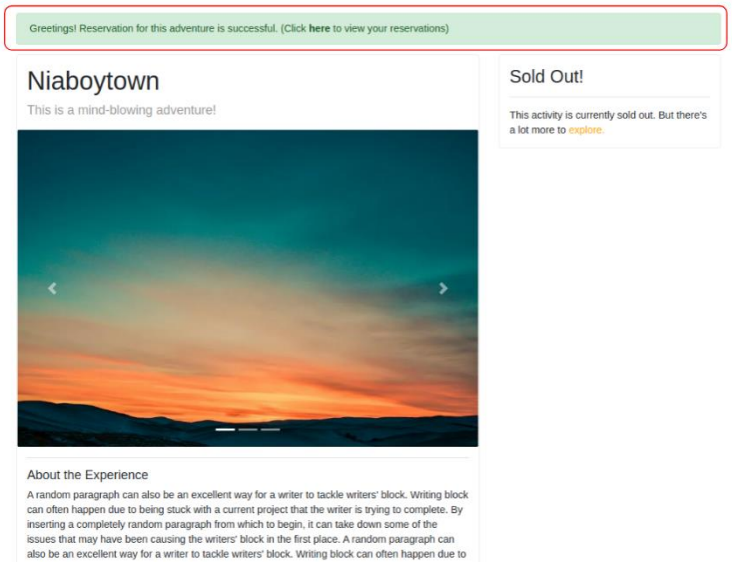
3. If the backend is up and responds with success, you should see a popup window, like this, when user makes a reservation



You can also check the reservation details being stored in the reservations section of the backend/db.json file.

Show banner if adventure is already reserved

The method to be addressed is showBannerIfAlreadyReserved(). This banner needs to show up if the adventure has been reserved previously. Highlighted here.



Let's make some observations about this method

- Use the reserved field of the input adventure to make the if else decision (this field gets set on the Backend server when the reservation is successful)
- If the adventure is reserved, show the reserved-banner, else don't show it.

If you implement this correctly (along with the previous methods), all the test cases under adventure_details_page.test.js should pass

```
crio-user@kiran-criodo:~/workspace/kiran-criodo-ME_QTRIPDYNAMIC/frontend/__tests__$ npm test

> js-me@1.0.0 test /home/crio-user/workspace/kiran-criodo-ME_QTRIPDYNAMIC/frontend/__tests__
> NODE_ICU_DATA=node_modules/full-icu jest --verbose ./modules

 PASS  modules/adventure_details_page.test.js (9.266 s)
  Adventure Detail Page Tests
    ✓ Extract city from URL Params (68 ms)
    ✓ Check if fetch call for the adventure details was made and data was received (47 ms)
    ✓ Catches errors and returns null (81 ms)
    ✓ Tries adding a Adventure Details - Park (62 ms)
    ✓ Check if bootstrap gallery is working (44 ms)
    ✓ Check if conditional rendering is working (30 ms)
    ✓ Check if reservation cost is calculated correctly and updated in DOM (18 ms)
    ✓ Check if reservation banner is displayed (22 ms)
    ✓ Check if JQuery form submission is taking place (36 ms)
```

Reservations Page

In this milestone, we'll be addressing these methods in frontend/modules/reservations_page.js

- fetchReservations()
- addReservationToTable()

These will help populate the last QTrip page which is the Reservations Page.

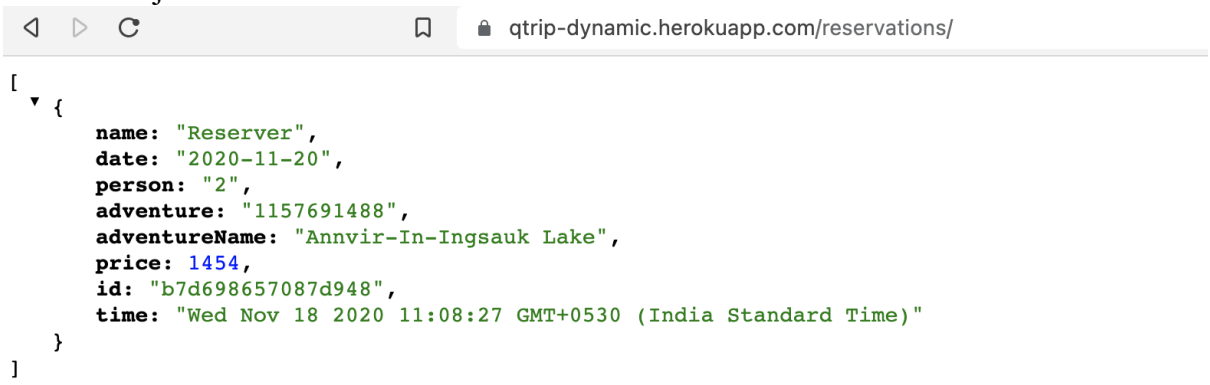
Fetch the reservations from the backend

You need to fetch the details of all the reservations by making a REST API call to the Backend Server. You will have to modify the fetchReservations() method to do this.

Some observations:

- Invoke the Backend's REST API using the configured backendEndpoint and the /reservations/ API endpoint

- Get the json data from the Backend's response
- Understand the fields in the response, you'll need to extract and use them in the next milestone
- Return this json data



Response from the "/reservations" endpoint

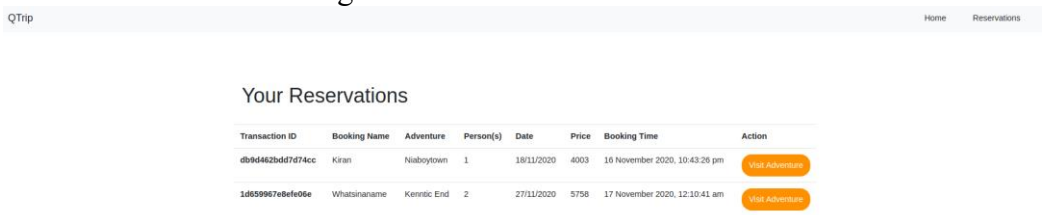
Display the reservations

The method to be addressed is addReservationToTable(), which will display the fetched reservations in a table format on the Reservations page.

Some key observations:

- Conditionally render the no-reservation-banner or reservation-table-parent based on whether the input reservations is empty or not.
- Loop through the input reservations.
- For each reservation, insert a row into the reservation-table, with the fields as shown in the table header on the Reservations page.
- Use the in-built Date functions to present the "Date" and "Booking Time" fields in the specified format. Ensure you get the Refer to the comments in the method.
    - **Note**: The locale must be set to "en-IN" for the test cases to pass
- Add a button using the provided reservation-visit-button css class. It should have a reference link that will take the user to that adventure page, upon clicking. Make use of the adventure id to create this reference link.
- Note that these reservations are not per user, they are the overall reservations made.

The resultant Reservations Page should look like this



2.12 Deploy the QTrip website using render and Netlify

Objective

Your QTrip website is ready with all the necessary content.

Till now, you had the backend running on your VM and were viewing the frontend (website) using the Live Server and your VM's IP address.

But this cannot be shared with others since the VM will be shut down after a while.

To be able to share this website with others and to be able to access it publicly, it needs to be deployed. We'll do this using render and Vercel. Render is the platform where you'll deploy your backend and Vercel is the platform where you'll deploy your frontend.

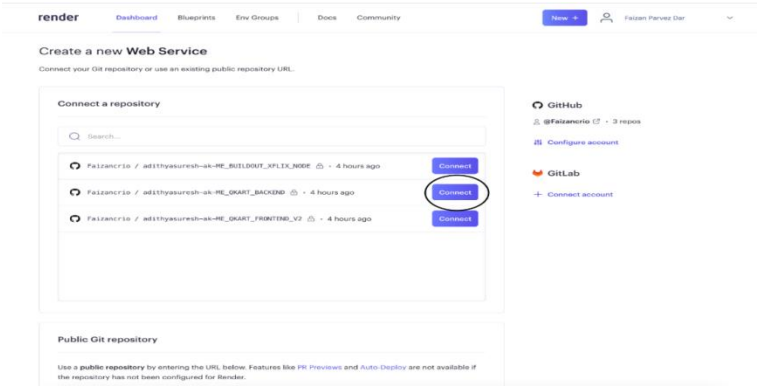
Primary goals

In this module, you will:

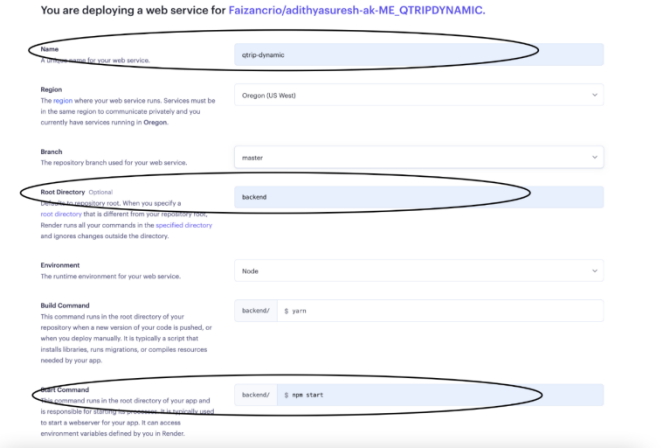
1. Deploy your QTrip backend to render and get a public URL
2. Configure this public URL for the frontend to use
3. Deploy your QTrip frontend to Vercel and get a public URL

Milestone 1 : Deploy the QTrip backend to Render

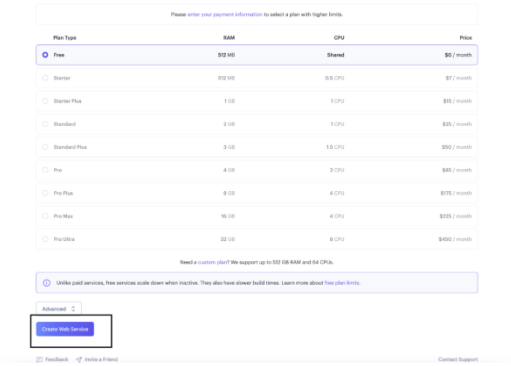
QTrip backend deployment to Render.
Till now you were running the Backend locally and have integrated it with a locally running Frontend. You need to deploy this Backend publicly so that it can be used by a publicly deployed Frontend.
You will deploy the Backend to Render. Render is a cloud service to deploy your apps and websites.



9. **You will be redirected to the below page, here just give the name for your web service i.e "Name" = qtrip-dynamic, “Root Directory” = backend and “Start Command” = npm start**

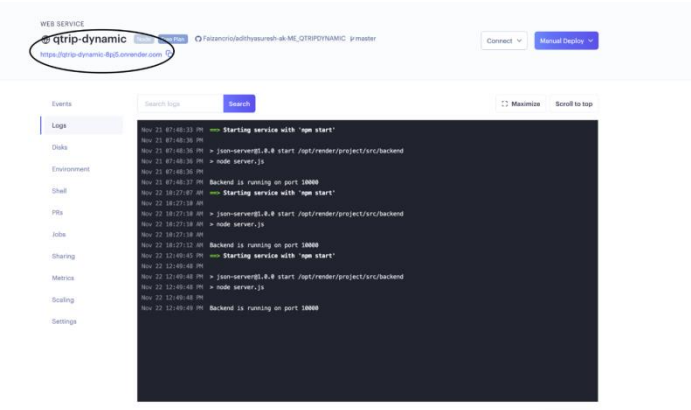


Then click on "Create Web Service"



10. **You are almost there, these are the logs that you will be getting upon successful deployment.**

11. You can now hit the API endpoints by using the below URL.

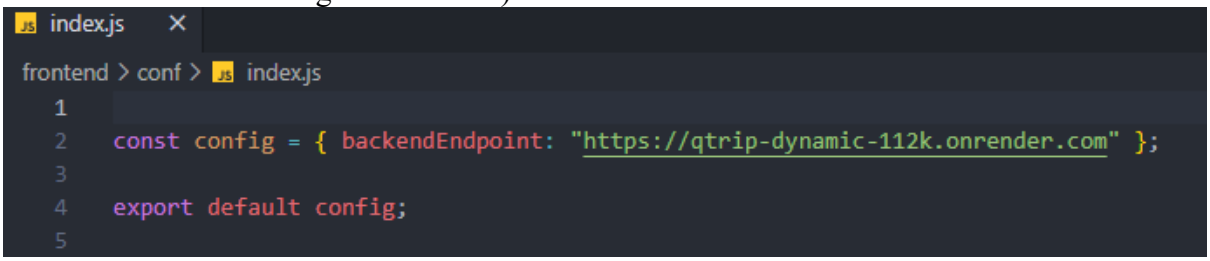


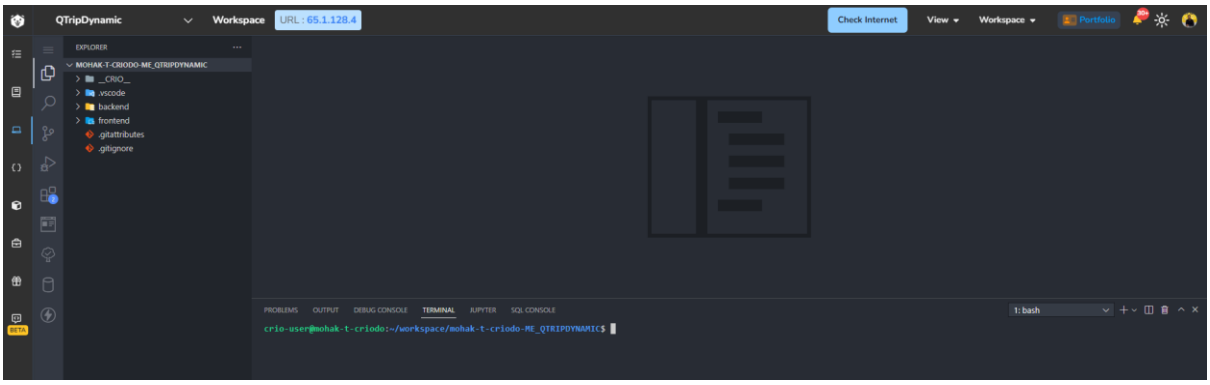
Milestone 2 : Deploy the QTrip frontend to Vercel

QTrip frontend deployment to Vercel

The QTrip dynamic website is ready. Now, let's deploy it publicly so that it can be shared with others. We will be using Vercel to do the deployment, which is a platform that can be used for this purpose. Use the references provided below to familiarize yourself with Vercel.

1. Configure the frontend/conf/index.js to point to the backend URL. Example (ensure that there is no trailing / in the URL)





5. Run vercel login command in the terminal

```
crio-user@mohak-t-criodo:~/workspace/mohak-t-criodo-ME_QTRIPDYNAMIC/frontend$ vercel login
Vercel CLI 32.4.1
? Log in to Vercel
● Continue with GitHub
o Continue with GitLab
o Continue with Bitbucket
o Continue with Email
o Continue with SAML Single Sign-On
──────────────────────────────────
o Cancel
```

6. Move down to the **"Continue with Email"** option using the down-arrow key and hit Enter.

```
crio-user@mohak-t-criodo:~/workspace/mohak-t-criodo-ME_QTRIPDYNAMIC/frontend$ vercel login
Vercel CLI 32.4.1
? Log in to Vercel
o Continue with GitHub
o Continue with GitLab
o Continue with Bitbucket
● Continue with Email
o Continue with SAML Single Sign-On
──────────────────────────────────
o Cancel
```

7. Input the email address associated with your GitHub account, and hit Enter.

```
crio-user@mohak-t-criodo:~/workspace/mohak-t-criodo-ME_QTRIPDYNAMIC/frontend$ vercel login
Vercel CLI 32.4.1
? Log in to Vercel email
? Enter your email address: trivedimohak@gmail.com
We sent an email to trivedimohak@gmail.com. Please follow the steps provided inside it and make sure the security code matches Pleasant Wallaby.
⠏ Waiting for your confirmation
```

8. You'll receive an email from Vercel, Click on the **Verify **button present in the mail.
9. Click on **Verify **again in the Redirected Page
10. Upon successful verification, it will show a success message like the below image:
11. Now, go back to your Crio Workspace terminal and run the command ****vercel**.**

```
crio-user@mohak-t-criodo:~/workspace/mohak-t-criodo-ME_QTRIPDYNAMIC/frontend$ vercel login
Vercel CLI 32.4.1
? Log in to Vercel email
? Enter your email address: trivedimohak@gmail.com
We sent an email to trivedimohak@gmail.com. Please follow the steps provided inside it and make sure the security code matches Pleasant Wallaby.
> Success! Email authentication complete for trivedimohak@gmail.com
Congratulations! You are now logged in. In order to deploy something, run `vercel`.
💡 Connect your Git Repositories to deploy every branch push automatically (https://vercel.link/git).
crio-user@mohak-t-criodo:~/workspace/mohak-t-criodo-ME_QTRIPDYNAMIC/frontend$ vercel
Vercel CLI 32.4.1
? Set up and deploy "~/workspace/mohak-t-criodo-ME_QTRIPDYNAMIC/frontend"? [Y/n]
```

- **For Setup and Deploy: **Press **Y** and hit Enter.

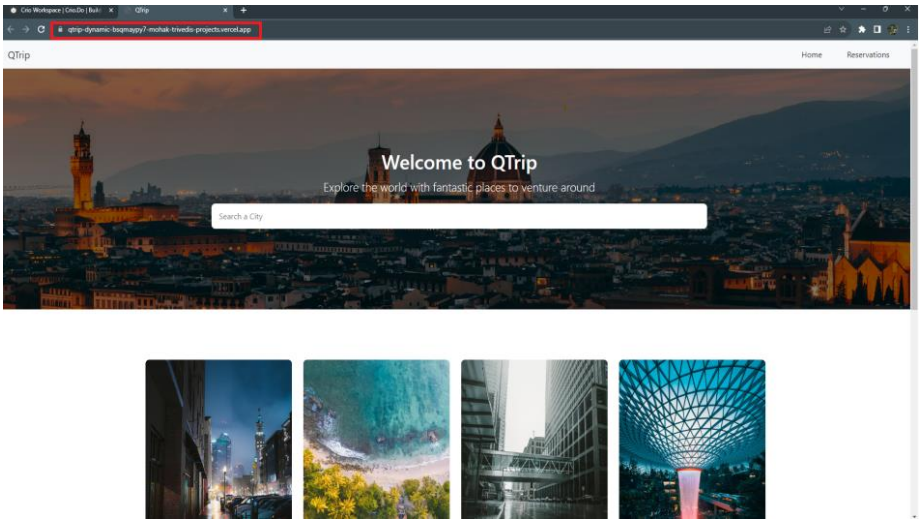
```
crio-user@mohak-t-criodo:~/workspace/mohak-t-criodo-ME_QTRIPDYNAMIC/frontend$ vercel
Vercel CLI 32.4.1
? Set up and deploy "~/workspace/mohak-t-criodo-ME_QTRIPDYNAMIC/frontend"? [Y/n] y
? Which scope do you want to deploy to?
● Mohak Trivedi's projects
```

- Ctrl+Click on the Preview link. This is your deployed link.

```
? Want to modify these settings? [y/N] n
Linked to mohak-trivedis-projects/qtrip-dynamic (created .vercel and added it to .gitignore)
Inspect: https://vercel.com/mohak-trivedis-projects/qtrip-dynamic/8OEvZvFYiANotHti82b3kxjzmmHA [1s]
Preview: https://qtrip-dynamic-bsqmaypy7-mohak-trivedis-projects.vercel.app [1s]
Deployed to production. Run `vercel --prod` to overwrite later (https://vercel.link/2F).
To change the domain or build command, go to https://vercel.com/mohak-trivedis-projects/qtrip-dynamic/settings
crio-user@mohak-t-criodo:~/workspace/mohak-t-criodo-ME_QTRIPDYNAMIC/frontend$
```

You will be redirected to the deployed application.

12. Thus, we have successfully deployed our QTripDynamic application!



QTrip deployment

The QTrip website was built and tested in the previous modules. In this module, you dealt with the deployment of the website. The cloud based deployment in this case made both the Backend (using render) and Frontend (using Vercel) publicly accessible and independent of your local system (of VM). Any time you want to update them, you can redeploy the updated code.