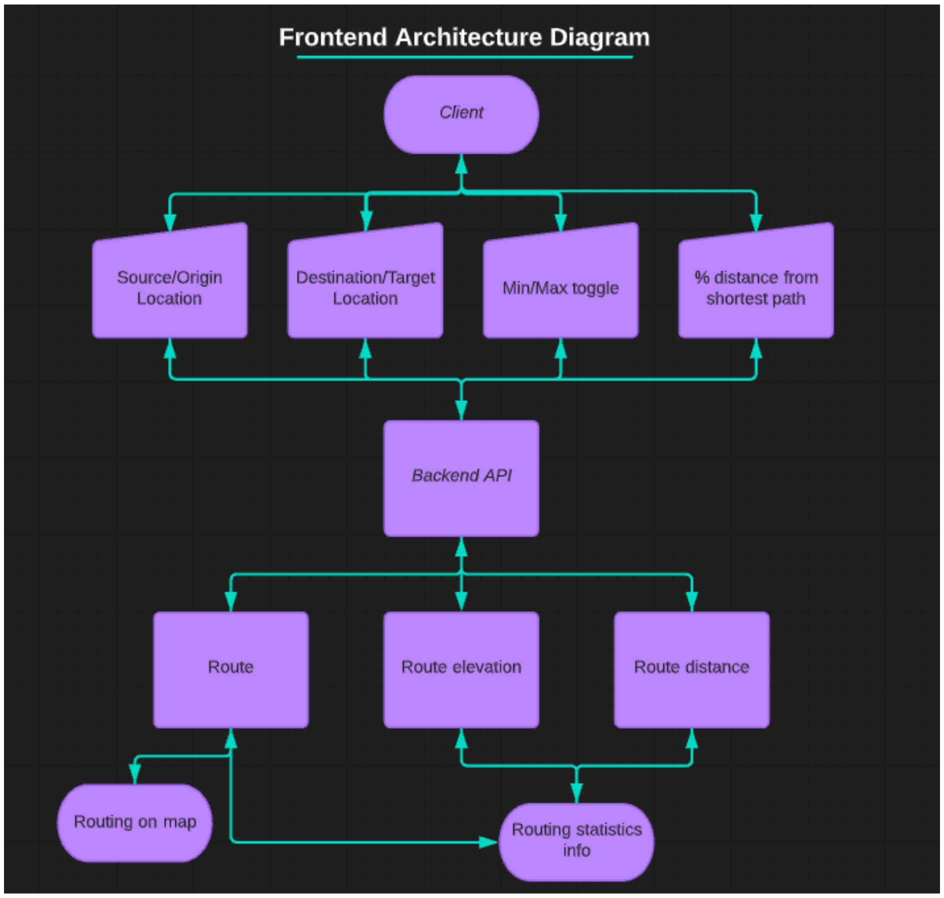
Developer Document: EleNa

Milo Cason-Snow, Jeffrey Chan, Enoch Hsiao, Mike Li

Architecture Diagram

We used a client-server architecture pattern. The client will first enter their source location, target location, minimized or maximized elevation, and % distance from the shortest path. These are the 4 user inputs that the client will choose from. All of this information will be sent to the backend to be processed for it to return a route on the map and routing info about elevation and distance to the client.

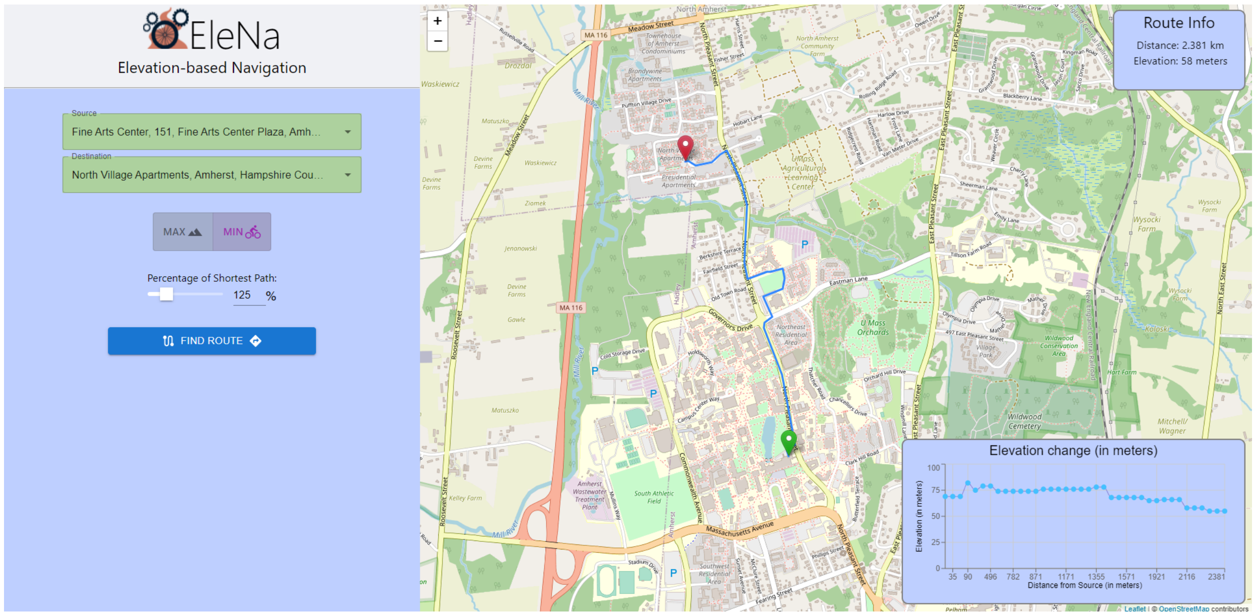


Third-Party Libraries/APIs Used:

Client	Server
● Leaflet.js ● React-leaflet ● Leaflet-Control-Geocoder ● OpenStreetMap	● Python ● Flask ● Flask-Cors ● OSMnx ● Networkx

Client:

User Input ↓ Map Interface ↓

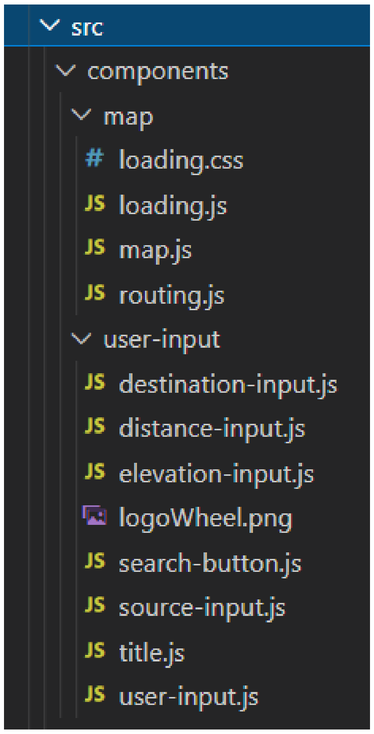


EleNa
Elevation-based Navigation

Source: Fine Arts Center, 151, Fine Arts Center Plaza, Amh...
Destination: North Village Apartments, Amherst, Hampshire Cou...
MAX MIN
Percentage of Shortest Path: 125 %
FIND ROUTE

Route Info
Distance: 2.381 km
Elevation: 58 meters

Elevation change (in meters)
Elevation (in meters)
Distance from Source (in meters)
Leaflet | © OpenStreetMap contributors



```
src
  components
    map
      loading.css
      loading.js
      map.js
      routing.js
    user-input
      destination-input.js
      distance-input.js
      elevation-input.js
      logoWheel.png
      search-button.js
      source-input.js
      title.js
      user-input.js
```

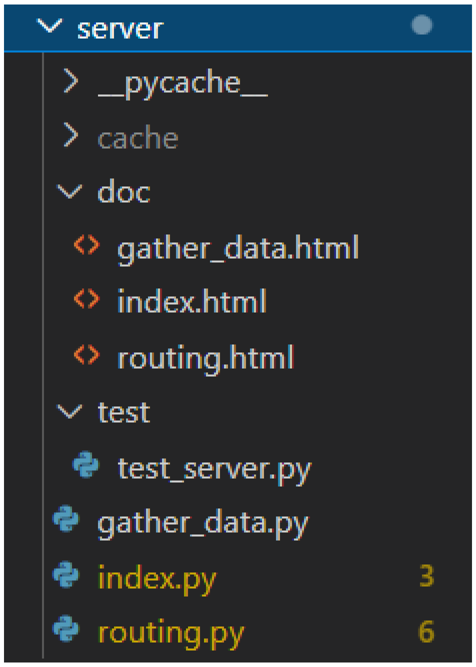
The client was designed using the React framework. The code for the individual components of the UI, the user inputs and the map interfaces, are located in `elena-frontend/src/components/*`.

`elena-frontend/src/components/map/*` houses the logic and design for the map interface. The map interface uses data from OpenStreetMap and displays the data using the Leaflet, Leaflet-Geocoder, and React Leaflet JavaScript libraries.

`elena-frontend/src/components/user-input/*` houses the logic and design for the user input side of the UI.

The client is hosted at https://elena-elevation-navigation.web.app/ using Google Firebase.

Server:



```
server
  _pycache_
  cache
  doc
    gather_data.html
    index.html
    routing.html
  test
    test_server.py
  gather_data.py
  index.py        3
  routing.py      6
```

The RESTFul API Server was developed using Flask, a Python micro web framework.

Some of the technologies used on the backend

Algorithm:

`routing.py` uses a modified version of networkx's single source dijkstra algorithm function to calculate the min or max route for a given start and end parameter. The routing is done as a strategy pattern, as explained in more detail later in this document.

Data Representation:

`gather_data.py` uses the osmnx library to get the map for the geocoded location where the person is requesting a route. It caches maps that it gathers so that for the next person to request a route in that place, the request will be much quicker. The map is stored in a networkx graph and cached using pickle in the cache/ folder. The Open Elevation API is used to add elevation information to each node of the graph for the algorithm. The elevation fetching is done up front, so requests don't have to be made for each node used in the algorithm.

Server Routing:

`index.py` contains the logic for the one public POST request, `/route`. This POST request has several options, including the type of route (min or max), the percent distance, the place to get a map for, and the start and end points in (lat, lng) form.

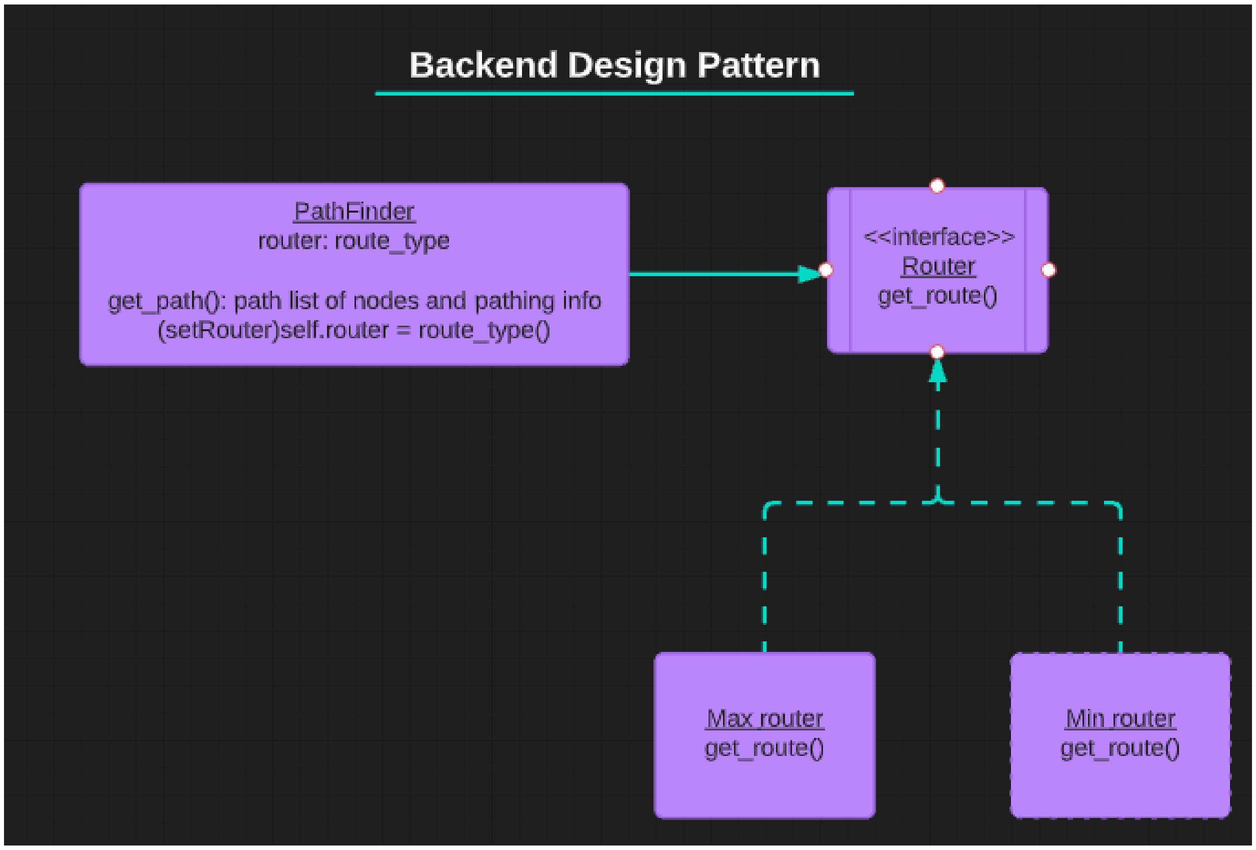
HTTP Method: POST
Endpoint: /route
Request Body Parameters:
{

```
      start: tuple of lat, lng for start node
      end: tuple of lat, lng for end node
      place: place name string
      percent: int, percent of shortest path route can be
      route_type: string, either 'min' or 'max'
}
```
Return:
Path list of node nums, dist of path, elevation change of path
Example Response:
```
200 OK
[{elevation: 81, street_count: 1, x: -72.5101952, y: 42.3764996},
{elevation: 81, street_count: 3, x: -72.5096417, y: 42.3768564}.
{elevation: 81, street_count: 3, x: -72.509623, y: 42.3769771}]
```

Server Hosting:
The server is hosted at
http://ec2-3-135-186-75.us-east-2.compute.amazonaws.com:8080/route. (now down)
The server will need to run on HTTPS in order to talk to the frontend hosted on firebase.

Key Design Elements (e.g., caching, design patterns, Dijkstra’s algorithm)

Modified Strategy Pattern:



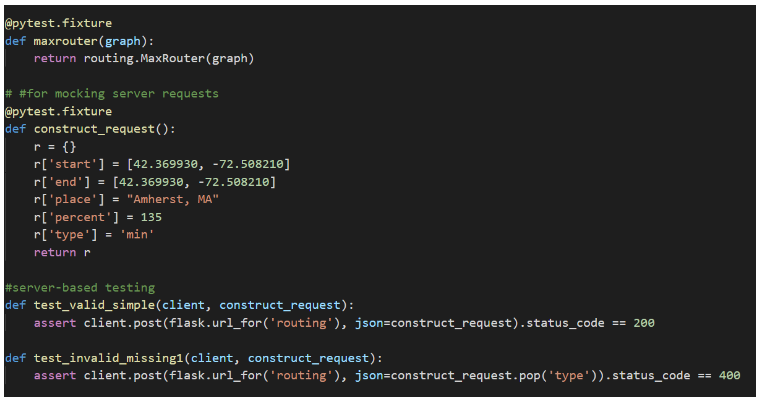
Used a modified version of dijkstra’s algorithm that min/max elevation and keeps track of path distance so it cuts off paths that go past the desired distance based on the routing type(min or max).

Caching:

Developers running this app need to have a “cache” folder in their server folder. Caching is implemented using python pickle files which can be easily loaded back into memory as networkx graphs with all of the necessary information already in them.

Test Plan

Testing will be done primarily on the backend with a few tests in the frontend.
For the backend, there will be testing done for choosing which router to use, server requests, server-based testing, caching-based testing, and routing-based testing which can be found in `EleNa/server/test/test_server.py.` This backend testing has testing for the different components of the app - the server, the routing and the caching system.

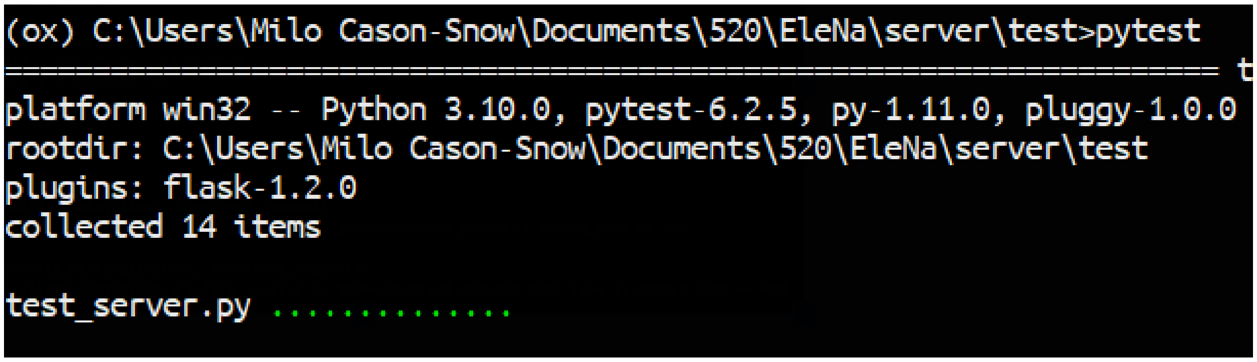


```python
@pytest.fixture
def maxrouter(graph):
    return routing.MaxRouter(graph)

# #for mocking server requests
@pytest.fixture
def construct_request():
    r = {}
    r['start'] = [42.369930, -72.508210]
    r['end'] = [42.369930, -72.508210]
    r['place'] = "Amherst, MA"
    r['percent'] = 135
    r['type'] = 'min'
    return r

#server-based testing
def test_valid_simple(client, construct_request):
    assert client.post(flask.url_for('routing'), json=construct_request).status_code == 200

def test_invalid_missing1(client, construct_request):
    assert client.post(flask.url_for('routing'), json=construct_request.pop('type')).status_code == 400
```

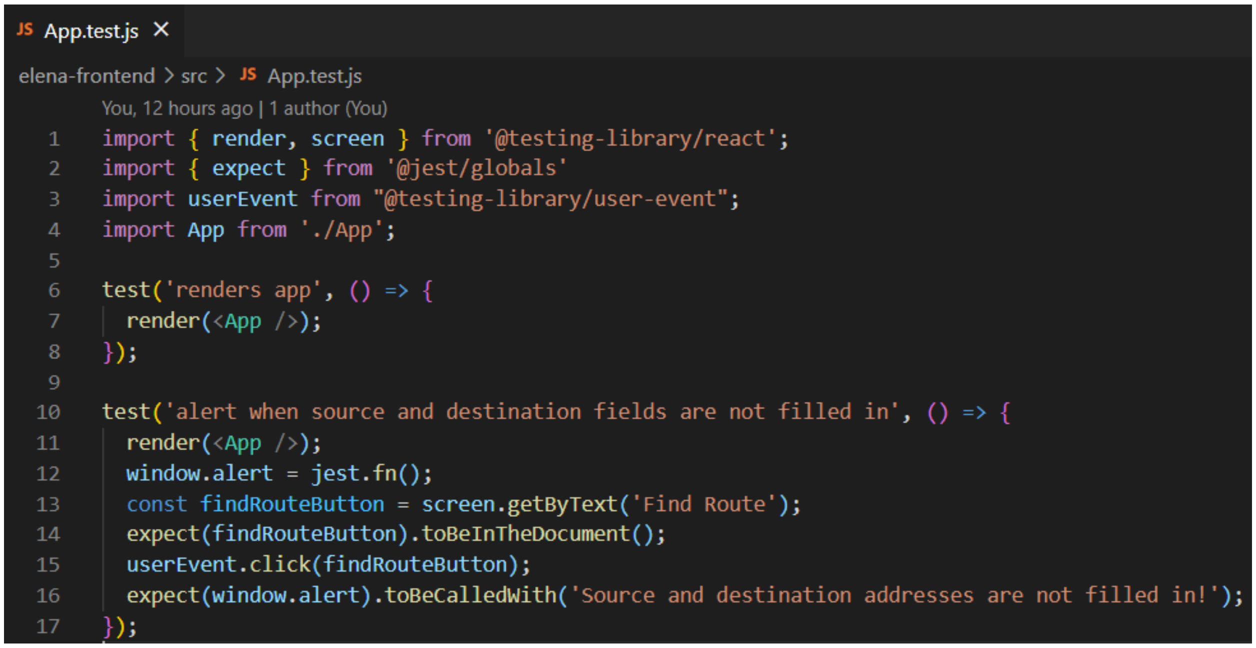


```
(ox) C:\Users\Milo Cason-Snow\Documents\520\EleNa\server\test>pytest
=========================================================================== t
platform win32 -- Python 3.10.0, pytest-6.2.5, py-1.11.0, pluggy-1.0.0
rootdir: C:\Users\Milo Cason-Snow\Documents\520\EleNa\server\test
plugins: flask-1.2.0
collected 14 items

test_server.py ..............
```

For the frontend, there are tests to validate that the app renders and to validate that a pop-up alert occurs when the source and destination fields are not filled in. One can run `npm test` in the `elena-frontend` directory, and the code for the test can be found in `EleNa/elena-frontend/src/App.test.js.`

Test code:

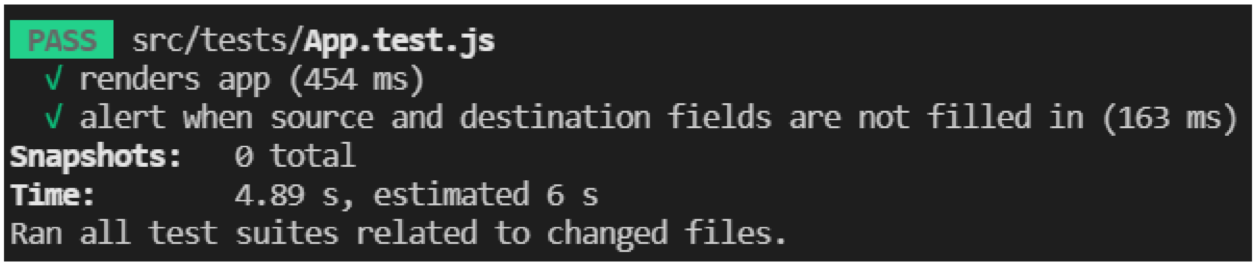

JS App.test.js ×
elena-frontend > src > JS App.test.js
You, 12 hours ago | 1 author (You)

```
import { render, screen } from '@testing-library/react';
import { expect } from '@jest/globals'
import userEvent from "@testing-library/user-event";
import App from './App';

test('renders app', () => {
  render(<App />);
});

test('alert when source and destination fields are not filled in', () => {
  render(<App />);
  window.alert = jest.fn();
  const findRouteButton = screen.getByText('Find Route');
  expect(findRouteButton).toBeInTheDocument();
  userEvent.click(findRouteButton);
  expect(window.alert).toBeCalledWith('Source and destination addresses are not filled in!');
});
```

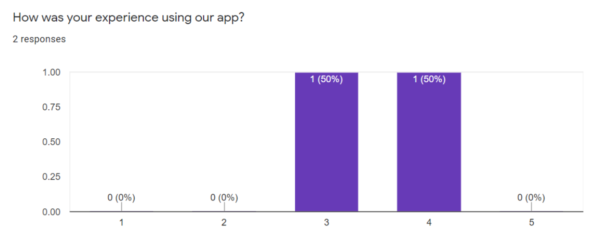
Test Suite UI (after running npm test):



```
 PASS  src/tests/App.test.js
  √ renders app (454 ms)
  √ alert when source and destination fields are not filled in (163 ms)
Snapshots:   0 total
Time:        4.89 s, estimated 6 s
Ran all test suites related to changed files.
```

Experimental Evaluation

We asked users to complete a google survey after using our app. We asked about their overall experience, if they would recommend the app to others, common non-functional requirements like readability, usability, reliability, performance, etc., features they would like added, and any bugs they encountered.
https://docs.google.com/forms/d/e/1FAIpQLSeHOTgYzQcRdqfUSPI_91Jzxj-RwXTHcRNWsKPreNGamPY6BA/viewform?usp=pp_url


How was your experience using our app?
2 responses
0 (0%)
0 (0%)
1 (50%)
1 (50%)
0 (0%)