

SW Engineering CSC648/848 Spring 2019

GatorState

Team 6

Team Members:

Team Lead: Rowvin Dizon (email: rdizon1@mail.sfsu.edu)

Front End Lead: Jonathan Gurdal

Front End: Daisy Sun

Back End Lead: Marlon Johnson

Full Stack: Minho Cha

Full Stack: Rene Elias

GitHub Master: Kayla Musleh

Milestone 4

May 8th, 2019

History Table

Date Submitted:	5/8/19
Date Revised:	N/A

1) Product summary (e.g. how would you market and sell your product – about ½ page)

- **Product Name:** Gator State
- **Major Committed Functions (P1):**
 - Home page
 - About page
 - Search (including search field validation)
 - Search results
 - Filtering
 - Search Details and Maps
 - Messaging/contact seller/user
 - Posting (Data Upload)
 - Dashboards (user, admin)
- **What is unique about our product?**
 - GatorState is unique because it is specifically tailored to SF State students. Our filter feature allows students to search for possible listings based on price and distance from the campus. GatorState also has a feature that allows for students to directly contact the seller of the listing on the website.
- **URL:** <http://gatorstate.tk/>

2) Usability Test Plan

Test objective:

We are testing the search function and how intuitive it is to use for any users in terms of easily being able to use our search function and apply filter. Users shall submit searches either by zip code, address, city, or empty search, and give their feedback on usability. In addition, we are testing if users can effectively figure out and use the Filter section along with the Search if needed. Users will be tested on how effectively they can read and use provided search results. These are all being tested to see and ensure that the main function of our website--the Search feature, is easy to use. Users of all technical background should be able to quickly navigate the website enough to use the main feature of searching for listings. Provided enough testing, the goal is to eliminate any confusion from users when navigating website, and follow up with simplicity in usage.

Test description:

System setup:

Go to our provided URL, this will direct you to GatorState homepage where our search bar is located.

Starting point:

The starting point for testing our search usability will be on GatorState home page, the search is located in the navbar at the top of the page.

Intended users:

The intended users are any students who are looking for potential housing. These intended users can have little to none WWW skills nor technical background.

URL:

<http://gatorstate.tk/>

What is being tested:

We are testing user satisfaction from using our search as well as the efficiency of using it. We are also testing that applying the filter to searches are intuitive and simple.

Usability Task description:

Task	Task Description
Search	Search for available listings
Search Keyword	Search for listings using a keyword
Search using Filter	Search for a specific housing type using filter selection

Questionnaire:

I found it quick and easy to search for listings (check one):

- ☐ Strongly disagree
☐ Disagree
☐ Neither agree or disagree
☐ Agree
☐ Strongly agree

It was easy to navigate the homepage(check one):

- ☐ Strongly disagree
☐ Disagree
☐ Neither agree or disagree
☐ Agree
☐ Strongly agree

The search results gave me what I was looking for (check one):

- ☐ Strongly disagree
☐ Disagree
☐ Neither agree or disagree
☐ Agree
☐ Strongly agree

3) QA Test Plan

Test Objective:

The test objective is to check if search functionality performs accordingly. Users should be able to submit searches that are either empty, a zip code, an address or a city. Any searches that do not satisfy the search requirements can also be submitted. For every search submitted, the correct output should be returned. Additionally, there shall be testing for the filter selection regarding how Housing Types narrows down search results and if it does it properly. The purpose of testing the Search function is to check if every part of the feature works accordingly to the specs. Users should be able to search utilizing any specific feature they choose and see it work properly. Tester reports should contain a PASS or FAIL.

HW and SW setup (including URL):

Have a working functioning laptop connected to WiFi. Open a web browser. Go to our provided URL: <http://gatorstate.tk/>, this will direct you to GatorState homepage where our search bar and search function is located.

Feature to be tested: Test if Search functionality and Filter function works accordingly.


QA Test plan:

Test	Test Description	Test Input	Expected Output	Test Results
Empty Search	Submit a search with nothing in the search field and verify you get all 4 results		Everything	PASS/FAIL
Zip Code Search	Submit a search with a 5 digit zip code in the search field and verify the expected result	“94132”	Listings with zip code ‘94132’ will be returned	PASS/FAIL
City Search	Submit a search with a city in the search field and verify you get the expected results	“San Francisco”	Listings with cities listed with ‘San Francisco’ will be returned	PASS/FAIL
Address Search	Submit a search with an address as the input and verify you get the expected result	“2546 26th Ave”	Listing that matches the inputted address will be returned	PASS/FAIL
Filter Search	Submit a search with a filter option selected and verify		Listings that match the selected filter will be returned	PASS/FAIL


4) Code Review:


We followed a general coding conventions for javascript.

Ref : https://www.w3schools.com/js/js_conventions.asp



Rowvin Dizon
Wed 5/8/2019 8:18 PM





search.js
4 KB

3 attachments (782 KB)

Hi Minh,

I've check the code and here is what I came up with.

<Documentation>

- All methods have to be commented in clear language.
If it is unclear to the reader, it is unclear to the user.
But there are no comments at all.
You'd better clarify the methods you are using so you make sure what the methods do.
- You'd also better to have @author for all authors.
I checked that you have @author on the main file, but it's also good to have it on each page so you can easily check who's an author for the file.
- All class, variable, and method modifiers are examined for correctness and all of them are fine.
- While all the headers look unique and good!

<Testing>

- You'd also better to have unit tests which are added for each code path, and behavior, and unit tests must cover error conditions and invalid parameter cases.
Also, unit tests for standard algorithms should be examined against the standard for expected results.
- Ensure that the code fixes the issue, or implements the requirement, and that the unit test confirms it. If the unit test confirms a fix for issue, add the issue number to the documentation.
Also check for possible null pointers are always checked before use.

<Error Handling>


- It's always good to have an Error Handling.
Consider using a general error handler to handle known error conditions and an Error handler must clean up state and resources no matter where an error occurs.

<Performance>

- Do not leave debugging code in production code.
- It's good that you avoid large objects in memory, or using String to hold large documents which should be handled with better tools. For example, don't read a large XML document into a String, or DOM.

Please respond with what your feedback and what thoughts you have on the review.

Best,
Rowvin



Minho Cha



Minho Cha



Minho Cha

Wed 5/8/2019 9:25 PM

Rowvin Dizon



FilterSelection.js

1 KB



Search.js

4 KB

2 attachments (10 KB)

Hi Rowvin,

I've just commented on the files.

It's going to be a great help if you can add some comments on that!

Thanks,
Minho



```
1 /*****
2 Peer Review
3
4 <Documentation>
5
6 - All methods have to be commented in clear language.
7 If it is unclear to the reader, it is unclear to the user.
8 There's no needless, obsolete and redundant comments.
9 Comments are consistent in format, length, and level of detail.
10 Also there's no code commented out, which is good!
11
12 - You'd also better to have @author for all authors.
13 I checked that you have @author on the main file, but it's also good to have it on each page so you can easily check who's an author for the file.
14
15 - All class, variable, and method modifiers are examined for correctness and all of them are fine.
16
17 - While all the headers look unique and good!
18
19 <Testing>
20
21 - You'd also better to have unit tests which are added for each code path, and behavior, and unit tests must cover error conditions and invalid parameter cases.
22 Also, unit tests for standard algorithms should be examined against the standard for expected results.
23
24 - Ensure that the code fixes the issue, or implements the requirement, and that the unit test confirms it.
25 If the unit test confirms a fix for issue, add the issue number to the documentation.
26 Also check for possible null pointers are always checked before use.
27
28 <Error Handling>
29
30 - It's always good to have an Error Handling.
31 Consider using a general error handler to handle known error conditions and an Error handler must clean up state and resources no matter where an error occurs.
32
33 <Performance>
34
35 - Do not leave debugging code in production code.
36
37 - It's good that you avoid large objects in memory, or using String to hold large documents which should be handled with better tools.
38 For example, don't read a large XML document into a String, or DOM.
39
40 *****/
```

```

import React, { Component } from 'react';
import Container from 'react-bootstrap/Container';
import InputGroup from 'react-bootstrap/InputGroup';
import Button from 'react-bootstrap/Button';
import FormControl from 'react-bootstrap/FormControl';
import styled from 'styled-components';
import { Link } from 'react-router-dom';
import { withRouter } from 'react-router';
import FilterSelection from './FilterSelection';

import { connect } from 'react-redux';
import { updateSearch } from '../components/redux/actions/searchActions';
//Styling the link to match other buttons
const LinkButton = styled(Link)`
  textDecoration: none;
  display: inline-block;
  font-size: 1em;
  color: white;
  background: rgb(0, 123, 255);
  border-radius: 3px;
  font: sans serif;
  height: 38px;
  margin: auto;
  padding: 0.25em 1em;
`;

//Search component will handle taking the user input and pass it to the db to generate results.
class Search extends Component {
  state = {
    searchInput: {
      search: '',
      address: '',
      zipcode: '',
      city: '',
      price: '',
      distance: '',
      type: ''
    }
  };

  //Set the state based on the input in the search form.
  onSearchHandler = event => {
    const { searchInput } = this.state;
    this.setState({
      searchInput: event.target.value
    });

    console.log(searchInput);
  };
}

```



```

render() {
  const { searchInput } = this.state;
  // console.log(this.state);
  return (
    <div className="App">
      // Search is called in Navbar
      <Container style={{width: '600px'}}>
        <InputGroup>
          //Prepend the filter before the search bar
          <InputGroup.Prepend>
            <FilterSelection />
          </InputGroup.Prepend>
          //Search bar
          <FormControl>
            placeholder="Search by address, zipcode, or city..."
            // onChange={this.onSearchHandler}
            onChange={e => {
              this.props.updateSearch(e.target.value);
            }}
            //value={searchInput}
          </>
          //Append the search button to the end of the search.
          <InputGroup.Append>
            // Link to connect to the result page after the submit button has been pressed
            <LinkButton to="/results" style={{ textDecoration: 'none', color: 'white' }}>Search</LinkButton>
          </InputGroup.Append>
        </InputGroup>
      </Container>
    </div>
  );
}

//Passing the search to the database
const mapStateToProps = state => {
  // console.log(state);
  return {
    searchValue: state.searchReducer.searchValue
  };
};

const mapDispatchToProps = {
  updateSearch
};

export default connect(
  mapStateToProps,
  mapDispatchToProps
)(Search);

```

```
import React, {Component} from 'react';
import Dropdown from 'react-bootstrap/Dropdown';
import DropdownButton from 'react-bootstrap/DropdownButton';
//This component handles storing to the state what filter the user selected.
class FilterSelection extends Component {
  constructor(props, context){
    super(props, context);
    //Used to change the state value
    this.handleChange = this.handleChange.bind(this);

    this.state = {
      value: ""
    };
  }

  handleChange(event){
    this.setState({value: event.target.value});
  }

  render(){
    return(
      //Dropdown selection that contains listing types
      //when a value is selected we call onChange to handle the state change.
      <DropdownButton
        type="checkbox"
        value={this.state.value}
        onChange={this.handleChange}
        title="Listing Type"
      >
        <Dropdown.Item value={"apartment"}>Apartment</Dropdown.Item>
        <Dropdown.Item value={"house"}>House</Dropdown.Item>
        <Dropdown.Item value={"dorm"}>Dorm</Dropdown.Item>
      </DropdownButton>
    );
  }
}

export default FilterSelection
```

5) Self-check on best practices for security – ½ page

- **Major Assets being Protected:**

- Passwords

- **Are we encrypting PW in the DB?**

- Passwords are being encrypted on our database.

- **Confirming Input Data Validation**

- We are validating search bar input by limiting it to 40 characters max for a search.
- Code used to limit amount of characters:

```
state = { chars_left: 40; }

handleWordCount = event => {

  const charCount = event.target.value.length;

  const charLeft = 40 - charCount;

  this.setState({ chars_left: charLeft});

}
```

- With this function, if the user tries to input more than 40 characters, the search bar will not accept any characters inputted beyond that limit for the search.

6) Self-check: Adherence to Original Non-Functional Specs

Original Non-Functional Specs	Status
Application shall be developed, tested and deployed using tools and servers approved by Class CTO and as agreed in M0 (some may be provided in the class, some may be chosen by the student team but all tools and servers have to be approved by class CTO).	DONE
Application shall be optimized for standard desktop/laptop browsers e.g. must render correctly on the two latest versions of two major browsers	DONE
Selected application functions must render well on mobile devices	ON TRACK
Data shall be stored in the team's chosen database technology on the team's deployment server	DONE
No more than 50 concurrent users shall be accessing the application at any time	ON TRACK
Privacy of users shall be protected and all privacy policies will be appropriately communicated to the users.	ON TRACK
The language used shall be English.	DONE
Application shall be very easy to use and intuitive.	DONE
Google analytics shall be added	DONE
No email clients shall be allowed	DONE

Pay functionality, if any (e.g. paying for goods and services) shall not be implemented nor simulated.	DONE
Site security: basic best practices shall be applied (as covered in the class)	ON TRACK
Before posted live, all content (e.g. apartment listings and images) must be approved by site administrator	ISSUE: We currently do not have an admin page setup so we aren't able to have the admin approve listings.
Modern SE processes and practices shall be used as specified in the class, including collaborative and continuous SW development	ON TRACK
The website shall <u>prominently</u> display the following <u>exact</u> text on all pages <i>"SFSU Software Engineering Project CSC 648-848, Spring 2019. For Demonstration Only"</i> at the top of the WWW page. (Important so as to not confuse this with a real application).	DONE