Algorithms Visualiser in ReactJS

Final Year Project
B.Sc.(Hons) in Software Development

BY
Kevin Niland

May 9, 2020

Advised by Dr. Martin Kenirons
Department of Computer Science and Applied Physics
Galway-Mayo Institute of Technology (GMIT)

Contents

About this project

Abstract This project is intended to be an project that can be used in an educational environment. During the Data Structures and Algorithms module we had in 2nd year, short videos were shown to us showing various sorting algorithms being visualized. It is a web-based application written in ReactJS, a web framework for building user interfaces. The project allows users to visualize various sorting algorithms, which could be helpful in an educational context.

This project also allows users to register and sign in to an account. From here, the user can record their screen and upload these sorts to a database. These sorts are available to be viewed by all users once logged in.

Chapter 1

Introduction

This chapter serves as an introduction to the project. In it, the various aspects of the project and the main objectives of the project are discussed. Each of the chapters are briefly discussed as well, detailing what each contains and what is discussed in each.

1.1 Discussion of the dissertation

Throughout this dissertation, the following six chapters each covered a distinct aspect of the project:

- **Chapter 1 - Introduction** - Chapter 1 serves as the introduction to the project, discussing the aim of the project, an overview of sorting algorithms, the project itself, the objectives of the project, and the scope of the project.
- **Chapter 2 - Methodology** - Chapter 2 covers the various research and software methodologies employed throughout, meetings with the project supervisor. development tools used, source control, and types of testing carried out.
- **Chapter 3 - Technology Review** - Chapter 3 introduces each of the different technologies, tools, and languages used to develop the project itself.
- **Chapter 4 - System Design** - Chapter 4 discusses the design of the system and goes over each of the main components of the application.
- **Chapter 5 - System Evaluation** - Chapter 5 evaluates the system as a whole, analysing the various aspects of the project and further discusses the type of testing carried out.
- **Chapter 6** - Chapter 6 serves as the conclusion to the project, evaluating the project further relative to initial objectives set out and testing.

1.2 Aim of the project

The aim of this project is to create a web-based application that allows users to visualise various different sorting algorithms, allow user authentication and upload visualizations to a database from which other users can view and compare the performance. This project is intended to be used in an educational sense, providing an interactive application where user's can actively choose a sorting algorithm to visualise and save these visualisations to view them at a later point.

1.3 Sorting Algorithms

A sorting algorithm is an algorithm that rearranges elements of a list in a certain order, according to a comparison operator on the elements [38]. The comparison operator is used to decide the new order of element in the respective data structure. The most frequently used orders are numerical order and lexicographical order. Efficient sorting is important for optimizing the efficiency of other algorithms (such as search and merge algorithms) that require input data to be in sorted lists. Sorting is also often useful for canonicalizing data and for producing human-readable output. More formally, the output of any sorting algorithm must satisfy two conditions:

1. The output is in non-decreasing order (each element is no smaller than the previous element according to the desired total order)

2. The output is a permutation (a reordering, yet retaining all of the original elements) of the input

Furthermore, the input data is often stored in an array, which allows random access, rather than a list, which only allows sequential access; though many algorithms can be applied to either type of data after suitable modification.

1.4 Discussion of the project

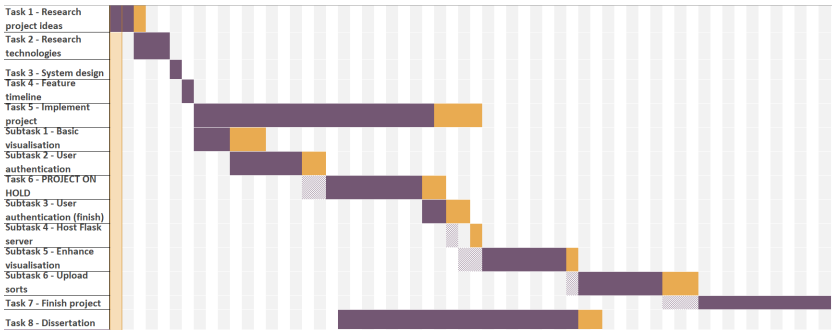
The project is sorting algorithm visualiser that allows users to choose a sorting algorithm to visualise. It also supports user authentication, where users can register and sign-in to upload, save, and view previous sorts. The project came about from viewing similar projects online and and an interest in finding out how such projects were developed. The project also came about from developing something that could be used in an educational sense. In 2nd year of our course, we were enrolled in a Data Structures and Algorithms module from which we learned about various different algorithms, how they were performed, space time complexity, etc.

1.5 Objectives of the project

There were several objectives I set out when first designing this project:

- Create an application that can be used in an educational sense, which can possibly be used for modules where students learn about data structures, algorithms, space time complexity, etc.
- Create a web application that allows users to visualise several different sorting algorithms.
- Allow users to register for an account and log in to save visualisations for later viewing.
- Allow user to upload previous sorts to viewing at a later point.

1.6 Scope of the project



Above is the projected and actual timeline of the project. The first few weeks of the project comprised of different researching project ideas, evaluating each based on complexity and scope and, after the project idea was settled on, the next couple of weeks comprised of researching various technologies to see which would be the most suitable to develop the chosen project idea. After this, the system was designed, along with a timeline of approximately when each feature should be implemented and how long each should take. With all this complete, the actual development of the project was started. The first feature to be implemented was to get basic visualization working. With this done, user authentication was started, which included writing the relevant functions to allow a user to register and log in to an account. After Christmas the project was resumed and user authentication finished. The Flask server, containing the functionality to allow the user to register and log in, was hosted on PythonAnywhere. With this done, more algorithms were written and incorporated and the visualization were enhanced. The final steps of the project included enabling the user to record sorts and upload them, and hosting the project.

1.6.1 Project Requirements

SORTING VISUALISER PROJECT TIMELINE

AT RISK	TASK NAME	DESCRIPTION	STATUS	ASSIGNED TO	START DATE	END DATE	DURATION
☐	Task 1 - Research project ideas	Research project ideas, with focus on project viability and scope	Complete	Kevin	09/08	09/25	17
☐	Task 2 - Research technologies	Research technologies that can be used to develop the project	Complete	Kevin	09/26	10/16	20
☐	Task 3 - Decide on system design	Decide on the system design, how each component will	Complete	Kevin	10/17	10/24	7
☐	Task 4 - Decide on feature timeline	Decide on what features will be developed first	Complete	Kevin	10/25	11/01	5
☐	Task 5 - Implement project	Work on project begins	Complete	Kevin	11/02	04/24	180
☐	Subtask 1 - Get basic visualisation working	Get basic visualisation working - implement a default sorting algorithm and visualise it	Complete	Kevin	11/03	11/11	8
☐	Subtask 2 - Start user authentication	Start on user authentication, which allows a user to register and sign in to an account	Complete	Kevin	11/12	11/20	8
☐	Task 6 - PROJECT ON HOLD	Project on hold for Christmas/Work on other projects	Complete	Kevin	11/21	12/28	37
☐	Subtask 3 - Finish user authentication	Finish user authentication	Complete	Kevin	12/29	01/10	12
☐	Subtask 4 - Host Flask server on PythonAnywhere	Set up Flask server on PythonAnywhere	Complete	Kevin	01/11	01/12	1
☐	Subtask 5 - Enhance visualization	Enhance visualisation - add more sorting algorithms, fix bugs, etc.	Complete	Kevin	01/13	02/13	14
☐	Subtask 6 - Implement uploading of past sorts	Implement uploading of past sorts to database	Complete	Kevin	02/13	02/21	8
☐	Task 7 - Wire everything together, bug fix, and test	Wire everything up - push to GitHub	Complete	Kevin	02/22	03/31	38
☐	Task 8 - Work on dissertation	Dissertation work	Complete	Kevin	12/29	04/24	117

There were several project requirements that I set out when designing this project. These can be seen in 1.6. The functions and features required were:

- Visualisation of sorting algorithms
- Support of various sorting algorithms
- Upload of user created data sets
- User authentication
- Database support

1.6.2 Project Limitations

As a whole, the application is primarily built to sort elements represented in a numerical format. The limitations of the project would be the inability to sort elements represented in a non-numerical format, special characters, files, etc. In the context of an algorithms visualiser, while being able to sort items like letters, special characters, files, etc. could very well be viable, visualising such things mightn't be as suitable as visualising the sorting of numbers.

1.6.3 Scope and context in relation to degree

This project was about designing and implementing an application that could visualize various sorting algorithms, allow users to register and login to an account, record the sorts, upload previous sorts, and view these past sorts. This application is similar to several different applications already developed, such as the applications researched in Chapter 2, section 2.1. As such, the project related to the degree in several different areas. It contains a wide array of technologies and elements that were previously covered in the course. The elements contained in the application are databases, user authentication, data handling, data transfer, web front-end, and testing - much of which were covered in several modules throughout the course such as:

- **Data Structures and Algorithms**, which covered sorting algorithms - a major component of the application.
- **Software Testing**, which covered testing and greatly aided in making sure the application worked as intended and provided testing processes to thoroughly test the application.
- **Professional Practice in IT**, which first introduced to us undertaking a large project over a period of time and having a project supervisor to which we would meet with regularly to discuss the progress of the project.
- **Database Management Systems**, which primarily covered the use and integration of databases with applications.

Chapter 2

Methodology

This chapter covers the various methodologies that were implemented in this project. It takes a look at the different types of research methodologies that were used such as Qualitative Research, Quantitative Research, and Mixed Methods Research and the different types of software development methodologies that were used such as Test Driven Development, Continuous Delivery, Extreme Programming, etc. and why each was chosen. Other areas also covered in this chapter include meetings, project management, development tools, source control, and testing.

2.1 Research Methodology

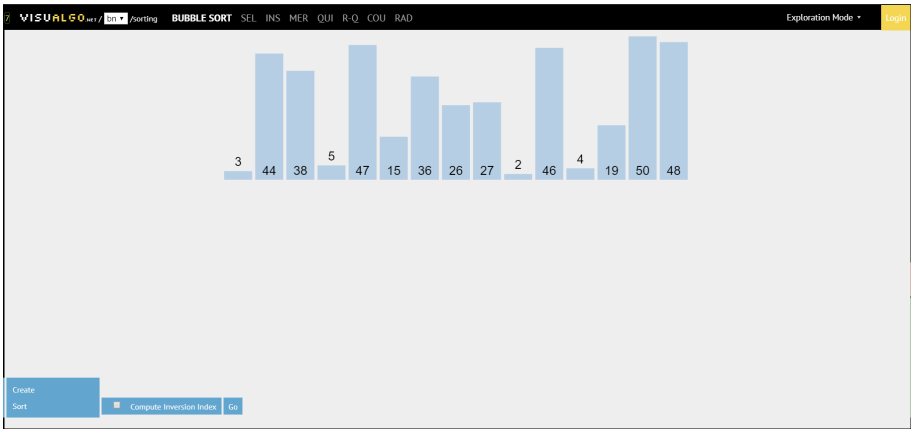
The research methodology that was used in this project was a Mixed Methods Research methodology, using a mix of both Qualitative Research and Quantitative Research. Qualitative research approaches are employed across many academic disciplines and is useful at an individual level. Qualitative data collection methods vary using unstructured or semi-structured techniques.

Research for this project included looking at and researching how to build a sorting algorithms visualizer [8] and researching similar applications, such as VisuAlgo, Sorting Visualizer by Clement Mihailescu, and Comparison Sorting Visualization hosted by University of San Francisco.

2.1.1 VisuAlgo

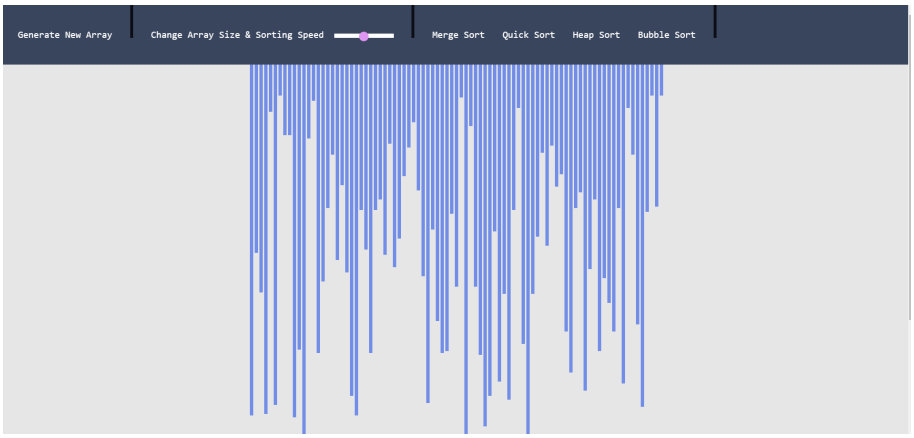
VisuAlgo [39] was conceptualised in 2011 by Dr Steven Halim as a tool to help his students better understand data structures and algorithms, by allowing them to learn the basics on their own and at their own pace. VisuAlgo contains many advanced algorithms that are discussed in Dr Steven Halim's book ('Competitive Programming', co-authored with his brother Dr Felix Halim) and beyond. Today, some of these advanced algorithms visualization/animation can only be found in VisuAlgo. VisuAlgo is an ongoing project and more complex visualisations are still being developed.

VisuAlgo contains animations and visualizations for many different algorithms and concepts in Computer Science such as sorting algorithms, linked lists, hash tables, graph traversals, etc. Inspiration for this application was taken from the sorting component of VisuAlgo.



Like VisuAlgo's sorting algorithms visualization component, the developed application allows the user to select different algorithms to visualize, the current elements being sorted are highlighted and swapped, and the user can enter their own data set to sort. I considered these features important for an application that would visualize sorting algorithms and, as such, implemented these in my own application.

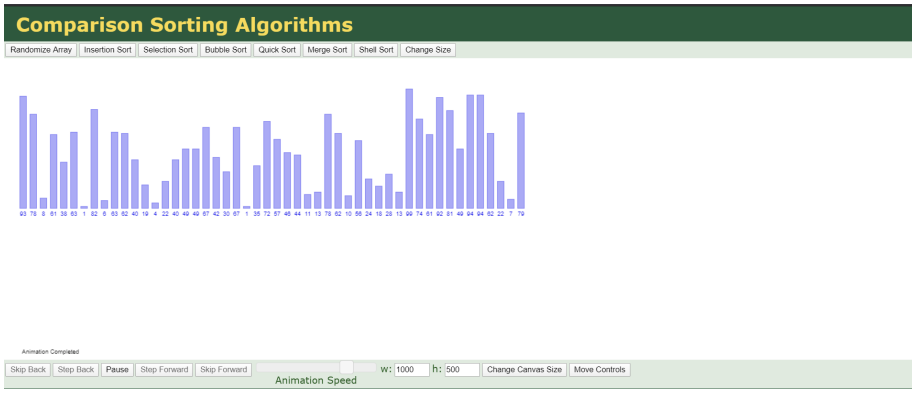
2.1.2 Sorting Visualizer by Clement Mihailescu



Like the other sorting visualizers, Clements Mihailescu's Sorting Visualizer [19] follows a similar trend: allows the user to choose from several different sorting algorithms, generate a new array, and change the size of the array and the speed at which the array is sorting. Clements Mihailescu's Sorting Visualizer was developed in React, which was one of the main motivators behind using React for my application as, from research, it was found that React would be one of the best options for developing a visualization application.

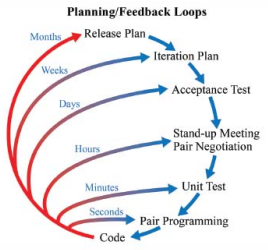
On a side note, the developed application is somewhat aesthetically similar to the Sorting Visualizer developed by Clements Mihailescu.

2.1.3 Comparison Sorting Visualization hosted by University of San Francisco



Again, like the other sorting visualizer, the Comparison Sorting Visualization hosted by the University of San Francisco [32] follows a similar trend. This application seemed to be a standard web application in HTML, JavaScript, and CSS. With this application, the user can randomize the array, select different algorithms to visualize, pause the sort, skip forward and back through the visualization, step back and forward through the sort, and change the size of the array of elements. Some of the features from this inspired features in my application, such as pausing the sort midway and randomizing the array multiple times before the user decides to sort it.

2.2 Software Development Methodology



The software development methodology that was used in this project was Extreme Programming (XP). Extreme Programming is a software development methodology designed to improve the quality of software and its ability to properly adapt to the changing needs of the customer or client. While there was no customer or client for this project, this methodology was still applicable.

It is a form of Agile software development. The Agile methodology was developed as a response to growing frustrations with Waterfall and other highly structured, inflexible methodologies. This approach is designed to accommodate change and the need to produce software faster. Similar to other Agile methods of development, Extreme Programming aims to provide iterative and frequent small releases throughout the project, allowing both team members and customers to examine and review the project's progress throughout the entire SDLC (Software Development Life Cycle).

It became clear early on that the Agile methodology would be the most suitable methodology to use for this project as the Agile Methodology allows for incremental development, changing requirements, prototyping, and sustainable development. As there would be weekly meetings with the project supervisor, being able to show and discuss the progress of the project would be a bonus.

2.3 Meetings

Project meetings were held weekly for the duration of the project with the project supervisor. The majority of the meetings consisted of:

- Project progress updates.
- Feedback on progress.
- Planning of the next development iteration.
- Discussions on possible additional features that could be incorporated.
- QA on various project elements.

Initial project meetings were more focused on brainstorming possible projects that could be developed. The first few weeks comprised of research, whereby possible projects ideas, appropriate technologies, and a project timeline was developed. Next, the system design was designed and a feature timeline was discussed. After this, the project was implemented with weekly meetings used to review the current progress and state of the project each week. Throughout the development of the project, any changes and problems were discussed in the meetings, and solutions were examined to see how these challenges would be overcome.

2.4 Development Tools

The main Integrated Development Environment used throughout the project was Visual Studio Code. Visual Studio Code is a source-code editor developed by Microsoft for Windows, Linux and mac OS. It includes support for debugging, embedded Git control and GitHub, syntax highlighting, intelligent code completion, snippets, and code refactoring.

Visual Studio Code is based on Electron, a framework which is used to develop Node.js applications for the desktop running on the Blink layout engine. Visual Studio Code is a source code editor that can be used with a variety of programming languages, including Java, JavaScript, Go, Node.js and C++.

Visual Studio Code was used throughout the development of the project as an obvious choice. Visual Studio Code provides support for many different types of files, which aids in development. It also has an in-built terminal which provides a detailed output and updates on project changes and output and feedback when the project is run. It also supports the installation of various different libraries and extensions through the command line or the built-in Marketplace. While certain IDEs are more suitable to certain languages (such as Eclipse being used to develop Java projects), Visual Studio Code is very suitable to developing web applications due to it's design.

2.5 Source Control

Source control (or version control) is the practice of tracking and managing changes to code. GitHub provides hosting for software development version control using Git. Git is an open-source distributed source code management system.

Advantages

There are a number of advantages to using Git:

- **Distributed Development** - Git is a distributed version control system. Instead of a working copy, each developer gets their own local repository, complete with a full history of commits. Having a full local history makes Git fast, since it means you don't need a network connection to create commits, inspect previous versions of a file, or perform diffs between commits.
- **Faster Release Cycle** - As a result of feature branches, distributed development, pull requests, etc. is a faster release cycle. This facilitates an agile workflow, encouraging developers to share smaller changes more frequently. This results in changes get pushed down the development pipeline faster

Disadvantages

Git is a very well-designed tool and, as such, has very few disadvantages:

- **Pricing** - Some of GitHub features, as well as features on other online repositories, are locked behind a SaaS paywall. If you have a large team, this can add up fast. Those who already have a dedicated IT team and their own internal servers are often better off using their own internal git for cost reasons, but for most, the cost isn't outrageous.
- **Security** - GitHub does offer private repositories, but this isn't necessarily perfect for many. For high value intellectual property, you're putting all of this in the hands of GitHub as well as anyone who has a login, which like many sites has had security breaches before and is targeted constantly. In addition, some clients/employers will only allow code on their own secure internal Git as a matter of policy.

Chapter 3

Technology Review

This chapter discusses the different technologies used in throughout the project. It discusses the the advantages and disadvantages of each technology and why certain technologies were used over others.

3.1 Overview

This project is comprised of React as the frontend, Flask as the server which is hosted on PythonAnywhere, MongoDB as the database for user authentication, Firebase as the database to store sorts, and Docker and Firebase to deploy/host the application. Throughout the project, the following technologies were also used and tested before the above was ultimately chosen:

- Angular/Ionic [15]
- Django
- MySQL
- Amazon Web Services
- Heroku

3.2 Main Technologies

This section will discuss the main technologies currently in use in the web application. The preceding section will discuss other technologies tried but ultimately weren't incorporated.

3.2.1 React

React (also known as React.js or ReactJS) is a JavaScript library for building user interfaces [29]. It is maintained by Facebook and a community of individual developers and companies [26].

React can be used as a base in the development of single-page or mobile applications [1]. However React is only concerned with rendering data to the DOM and so creating React applications usually requires the use of additional libraries for state management and routing. Redux and React Router are respective examples of such libraries.

Advantages

React has many advantages, several of which apply to this project:

- **Wide React and Redux toolset** - Both React and Redux come with a set of related tools that make a developer's life easier. For instance, React Developer Tools extension for Chrome and a similar one for Firefox allow for examining component hierarchies in the virtual DOM and editing states and properties. Additionally, you can check React Sight that visualizes state and prop trees; Reselect DevTools that helps with debugging and visualizing Reselect, a selector library for Redux. Redux DevTools Profiler Monitor allows for profiling actions in well.
- **Faster Rendering** - Building a high-load application is essential to consider how the structure will impact the overall app performance. Even latest platforms and engines can't ensure the absence of bottlenecks, because DOM (Document Object Model) is tree-structured and even small changes at the upper layer can cause awful ripples to the interface. To solve the issue, Facebook development team has introduced Virtual DOM – currently, one of the benefits of using React for heavy loaded and dynamic software solutions. As the name suggests, it is a virtual representation of the document object model, so all the changes are applied to the virtual

DOM first and then, using diff algorithm, the minimal scope of necessary DOM operations is calculated. Finally, the real DOM tree is updated accordingly, ensuring minimum time consumed. This method guarantees better user experience and higher app performance.

- **Guaranteed Stable Code** - To make sure that even small changes that take place in the child structures won't affect their parents, ReactJS uses only downward data flow. Changing an object, developers just modify its state, make changes, and, after that, only particular components will be updated. This structure of data binding ensures code stability and continuous app performance.

Disadvantages

There are a few disadvantages to using React:

- **Poor Documentation** - The problem with documentation traces back to constant releases of new tools. Different and new libraries like Redux and Reflux are promising to accelerate the work of a library or improve the entire React ecosystem. At the end, developers struggle with integrating these tools with ReactJS. Some members of the community think that React technologies are updating and accelerating so fast that there is no time to write proper instruction. To solve this, developers write their own documentation for specific tools used by them in current projects.

- **JSX as a barrier** - ReactJS uses JSX. It's a syntax extension, which allows mixing HTML with JavaScript. JSX has its own benefits (for instance, protecting code from injections), but some members of the development community consider JSX to be a serious disadvantage. Developers and designers complain about JSX's complexity and consequent steep learning curve.

- **Limitations in certain aspects** - As discovered with this project, certain tasks that can be done easily in other languages can't be done as easily in React. For example, if you wish to list all files in Firebase Storage using HTML, this can be done easily using a standard table to list the files. However, in React, this does not seem to be possible at the moment as it only seems to be possible to list one file at a time. Fortunately, the React community has developed libraries to get around these problems.

Purpose in application

A vast majority of the application was written in React, namely the visualization aspect [13], user registration and login forms, recording of sorts [22], upload of sorts, and viewing of sorts. Below discusses how each of the main parts of main page were developed:

Main Page

```
/**
 * MainPage - Main page of the application. Allows users to choose a sorting algorithm to visualize
 */
class MainPage extends Component {
  constructor(props) {
    super(props);

    // Contains various states related to sorting aspect
    this.state = {
      array: [], // Array of elements to sort
      isSelected: [], // Array of selected elements up for sorting
      show: true, // Will enable access to various functionality if user is logged in
      stillSorting: false, // Determines whether the array of elements is still being sorted
      sortName: 'Bubble Sort', // Default sort name
      dataset: '' // Contains the user dataset
    };

    this.sortHistoryIndex = 0;
    this.isFinished = null;
    this.sortSize = defaultDatasetSize;
    this.sortSpeed = defaultSortSpeed;
```

In the React sense, "state" is an object that represents the parts of the app that can change. Each component can maintain its own state, which lives in an object called this.state. If you'd like your app to do anything – if you want interactivity, adding and deleting things, logging in and out – that will involve state. In the context of this application, state contains information on the array of elements to be sorted, whether or not a certain element has been selected for sorting, if the application is still sorting the array, the current sort selected and the user's data set. The application will change and update these values as necessary.

```
this.props.history.listen((algorithm) => {
  this.generateRandomArray();

  // Determines which sort name to display based on chosen sorting algorithm
  switch (algorithm.pathname) {
    case '/bogo-sort':
      this.setState({sortName: 'Bogo Sort'});
      break;
    case '/bubble-sort':
      this.setState({sortName: 'Bubble Sort'});
      break;
    case '/heap-sort':
      this.setState({sortName: 'Heap Sort'});
      break;
    case '/insertion-sort':
      this.setState({sortName: 'Insertion Sort'});
      break;
    case '/merge-sort':
      this.setState({sortName: 'Merge Sort'});
      break;
    case '/quick-sort':
      this.setState({sortName: 'Quick Sort'});
      break;
    case '/selection-sort':
      this.setState({sortName: 'Selection Sort'});
      break;
```

Here, the application simply sets the current name of the selected sort and displays it the user.

```
/**
 * Handles dataset defined by user
 */
handleDataset = ({ arr }) => {
  this.setState({ dataset: arr.value });
};

/**
 * Submits dataset to be sorted. Generates the user array from this
 */
handleSubmit = () => {
  const { dataset } = this.state;

  this.generateUserArray(dataset);
};
```

Here the application handles the user's data set: handleDataset() will first save the user's data set to an array in state. handleSubmit() will submit the data set to be sorted i.e. the application will generate a visual representation of the user's data set. The user can then choose what sorting algorithm to apply and visualize.

```
userArray(dataset) {
  this.handleIsFinished();

  let array = [];
  this.counter = 0;
  this.sortedElements = [];
  this.selectedElements = [];

  /**
   * Process the dataset, enabling a visual representation to be generated from it
   */
  for (i = 0; i < dataset.length; i++) {
    array = dataset.split("");
    dataset = dataset.split(/[ ,]+/).join(',');
    array = dataset.split(',');
  }

  /**
   * Shuffle the user array using the Fisher-Yates Shuffle
   *
   * Implementation of the Fisher-Yates Shuffle: https://stackoverflow.com/a/2450976/8721358
   */
  var currentIndex = array.length, temp, random;

  while (currentIndex !== 0) {
    random = Math.floor(Math.random() * currentIndex);
    currentIndex -= 1;

    // And swap it with the current element.
    temp = array[currentIndex];
    array[currentIndex] = array[random];
    array[random] = temp;
  }
```

To generate a visual representation of the array, I first needed to process the user's input correctly. The data set was iterated over and any unnecessary character was removed. The contents of the data set was then stored in array. Next, I needed to ensure the array was always shuffled when a visual representation of it was generated. To do this, I used the Fisher-Yates shuffle which is an algorithm for generating a random permutation of a finite sequence—in plain terms, the algorithm shuffles the sequence. The algorithm effectively puts all the elements into a hat - it continually determines the next element by randomly drawing an element from the hat until no elements remain.

```
/**
 * Generates a random array of a fixed size
 */
randomArray() {
  this.handleIsFinished();

  this.counter = 0;
  this.sortedElements = [];
  this.selectedElements = [];

  let array = [];

  for (i = 0; i < SIZE; i++) {
    array.push(Math.floor(Math.random() * 50) + 1);
  }

  this.setState({array: array, isSelected: -1});
}
```

To generate a random array, a sort size was first set. A random array of elements was then generated of that size.

```
/**
 * Determines the color of each bar during the sorting process
 *
 * @param {*} isSelected - Selected index in array
 * @param {*} index - Current index
 */
setColor(isSelected, index) {
  for (i = 0; i < isSelected.length; i++)  {
    if (isSelected[i] === index) {
      return BARCOLORS[i];
    }
  }

  // Determines the color of the bars when not being sorted
  return '#48d0fa';
}
```

setColor() determines the color of bars that represent each element in the array. In this function, the application iterates over the array of selected elements, and if index i of that array is the index, the bar will be changed to the color in index i of highlightedColors.

```
 * @param {*} array - Array to be sorted
 * @param {*} sortedElements - Elements that have been sorted
 * @param {*} selectedElements - Elements that have been selected for sorting
 */
sortArray(array, sortedElements, selectedElements) {
  switch (this.props.location.pathname) {
    case '/bogo-sort':
      BogoSort.bogoSort(array.slice(), sortedElements, selectedElements);
      break;
    case '/bubble-sort':
      BubbleSort.bubbleSort(array.slice(), sortedElements, selectedElements);
      break;
    case '/heap-sort':
      HeapSort.heapSort(array.slice(), sortedElements, selectedElements);
      break;
    case '/insertion-sort':
      InsertionSort.insertionSort(array.slice(), sortedElements, selectedElements);
      break;
    case '/merge-sort':
      MergeSort.mergeSort(array.slice(), sortedElements, selectedElements);
      break;
    case '/quick-sort':
      QuickSort.quickSort(array.slice(), sortedElements, selectedElements);
      break;
    case '/selection-sort':
      SelectionSort.selectionSort(array.slice(), sortedElements, selectedElements);
      break;
    case '/shell-sort':
      ShellSort.shellSort(array.slice(), sortedElements, selectedElements);
      break;
    default:
      BubbleSort.bubbleSort(array.slice(), sortedElements, selectedElements);
    }
  }
```

This piece of code will execute the correct sorting algorithm. The application will first get 'path', which is comprised of location.pathname. Then, based on this, will pass the array, sortHistory, and selectedHistory to the correct algorithm. Each of these algorithms are discussed in section 4.2.1 of the System Design chapter. The visualization of the array being sorted will then be shown.

```
      <div className="record-wrapper">
        <a href="https://jmperezperez.com/screenflow/" target="_blank" rel="noopener noreferrer">
          <Button className={classes.button} style={styles.title}>Screen Record API</Button>
        </a>
      </div>}
    </div>

    {/* Displays state of application */}
    <div>
    {this.state.stillSorting &&
      <span className="sort-name">Sorting array...</span>}
    </div>

    <div>
    {!this.state.stillSorting &&
      <span className="sort-name">Waiting...</span>}
    </div>

    <div className="bar-wrapper">
      {this.state.array.map((item, index) => <ArrayElement key={index} size={item} color={this.determineBarColor(this.state.isSelected, index
    </div>

    {/* Display the current sorting algorithm chosen */}
    <span className="sort-name">Current sorting algorithm: {this.state.sortName}</span>

    <div className="buttons-wrapper">
      <Button className={classes.button} style={styles.title} onClick={ () => this.generateRandomArray()}>Generate random array</Button>
      <Button className={classes.button} style={{backgroundColor: this.state.stillSorting ? 'red' : classes.button.backgroundColor, textTrans
       <div>
       {this.state.show &&
        <div className="clearfix">
          <div className="buttons-wrapper">
            <Button className={classes.button} style={styles.title} onClick={ this.handleSubmit }>Add to own dataset</Button>
            <TextField
            className="dataset-box"
            variant="outlined"
```

The render() function renders all the appropriate GUI, ranging from the toolbar for the various sorting algorithm, the bars themselves, and the various buttons for interacting with the application.

MainToolbar

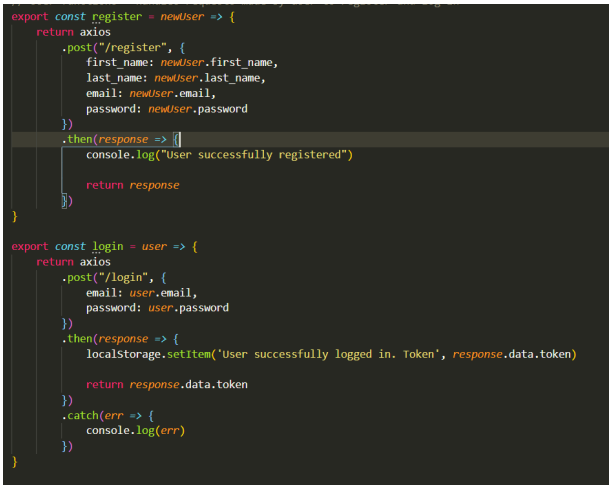
```
const loggedIn = localStorage.getItem('loggedIn') === 'true'; // Gets loggedIn if true
const loggedInUser = loggedIn ? localStorage.getItem('user') : ""; // Will display logged in user if loggedIn is true

function logOut() {
  localStorage.clear();

  window.location.reload();
}

/**
 * Buttons for the MainToolbar - Allows user to choose sorting algorithm to visualize
 *
 * @param {*} props
 */
function MainToolbarButtons(props) {
  return (
      <div className="sort-bar">
        <Button style={styles.title} onClick={() => props.history.push('bubble-sort')}>Bubble Sort</Button>
        <Button style={styles.title} onClick={() => props.history.push('heap-sort')}>Heap Sort (Almost)</Button>
        <Button style={styles.title} onClick={() => props.history.push('insertion-sort')}>Insertion Sort</Button>
        <Button style={styles.title} onClick={() => props.history.push('merge-sort')}>Merge Sort</Button>
        <Button style={styles.title} onClick={() => props.history.push('quick-sort')}>Quick Sort</Button>
        <Button style={styles.title} onClick={() => props.history.push('selection-sort')}>Selection Sort</Button>
        <Button style={styles.title} onClick={() => props.history.push('shell-sort')}>Shell Sort</Button>
        <Button style={styles.title} onClick={() => props.history.push('bogo-sort')}>Bogo Sort (Not working)</Button>
      </div>
  );
}
```

When a component is rendered by React Router, that component is passed three different props: location, match, and history. This history prop comes from the History library and has properties on it related to routing. In this case, the one we're interested in is history.push. What it does is it pushes a new entry into the history stack - redirecting the user to another route. When an algorithm is selected, the correct route to the algorithm is selected. This allows the algorithm to be correctly visualized.

```
return (
  <div>
  <AppBar className="main-toolbar" position="static" >
      <Toolbar className="main-toolbar">
        {/* Dropdown menu for links to other pages */}
        <Dropdown as={ButtonGroup}>
          {/* <Button color="#FFFFFF" style={styles.title}>Sorting</Button> */}
          <Button color="#FFFFFF" style={styles.title}>Sorting</Button>

          <Dropdown.Toggle split variant="success" id="dropdown-split-basic" />

          <Dropdown.Menu className="dropdown-button">
            <Dropdown.Item href="/login">Login/Register</Dropdown.Item>
            <Dropdown.Item href="/sorts">Sorts</Dropdown.Item>
          </Dropdown.Menu>
        </Dropdown>

        <div style={styles.line}></div>

        <MediaQuery minWidth={1000}>
          <Typography style={{ flexGrow: 1 }}>
            <MainToolbarButtons className="menu" history={ props.history } />
          </Typography>
        </MediaQuery>

        <MediaQuery maxWidth={1000}>
          <IconButton onClick={(e) => menu(e)} edge="start" style={styles.button} aria-label="menu">
            <MenuIcon />
          </IconButton>
        </MediaQuery>

        <div style={styles.line}></div>
```

The above code simply renders the toolbar. The toolbar is 'exported' as a component, which can then be used in other React pages.

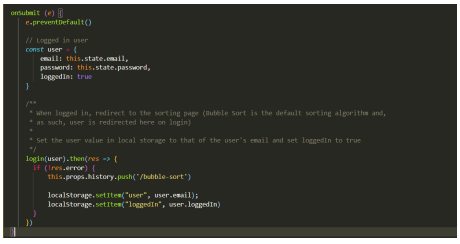
User Functions



```
export const register = newUser => {
    return axios
        .post("/register", {
            first_name: newUser.first_name,
            last_name: newUser.last_name,
            email: newUser.email,
            password: newUser.password
        })
        .then(response => {
            console.log("User successfully registered")

            return response
        })
}

export const login = user => {
    return axios
        .post("/login", {
            email: user.email,
            password: user.password
        })
        .then(response => {
            localStorage.setItem('User successfully logged in. Token', response.data.token)

            return response.data.token
        })
        .catch(err => {
            console.log(err)
        })
}
```

The user functions file handles the register and login requests [2]. These requests are delegated to the Flask server, where the appropriate action is taken. A response will then be returned based on the initial request.

Login Page



```
onSubmit (e) {
    e.preventDefault()

    // Logged in user
    const user = {
        email: this.state.email,
        password: this.state.password,
        loggedIn: true
    }

    /**
     * When logged in, redirect to the sorting page (Bubble Sort is the default sorting algorithm and,
     * as such, user is redirected here on login)
     *
     * Set the user value in local storage to that of the user's email and set loggedIn to true
     */
    login(user).then(res => {
      if (!res.error) {
        this.props.history.push('/bubble-sort')

        localStorage.setItem("user", user.email);
        localStorage.setItem("loggedIn", user.loggedIn)
      }
    })
}
```

```
<LoginRegisterToolbar />
<div className="row">
    <div className="col-md-6 mt-5 mx-auto">
        <form noValidate onSubmit={this.onSubmit}>
            <h1 className="h3 mb-3 font-weight-normal">Please sign in</h1>
            <div className="form-group">
                <label htmlFor="email">Email Address</label>
                <input type="email"
                    className="form-control"
                    name="email"
                    placeholder="Enter Email"
                    value={this.state.email}
                    onChange={this.onChange} />
            </div>
            <div className="form-group">
                <label htmlFor="password">Password</label>
                <input type="password"
                    className="form-control"
                    name="password"
                    placeholder="Enter Password"
                    value={this.state.password}
                    onChange={this.onChange} />
            </div>

            <button type="submit" className="btn btn-lg btn-primary btn-block">
                Sign in
            </button>
        </form>

        {/* Allow the user to redirect to the register page if user doesn't have an account */}
        <div className="linkToLoginRegister">
            Don't have an account? <Link to="/register">Create an account</Link>
```

The Login page uses the login function defined in the user functions file to handle a login request. If the user has successfully logged in, the local storage will be appropriately updated and the application will redirect to the sorting page. The login form is a simple HTML inspired form.

Register Page

```
    this.state = {
        first_name: '',
        last_name: '',
        email: '',
        password: ''
    }

    this.onChange = this.onChange.bind(this)
    this.onSubmit = this.onSubmit.bind(this)
}

onChange (e) {
    this.setState({ [e.target.name]: e.target.value })
}

onSubmit (e) {
    e.preventDefault()

    // New user
    const newUser = {
        first_name: this.state.first_name,
        last_name: this.state.last_name,
        email: this.state.email,
        password: this.state.password
    }

    // If user has been register successfully, redirect to login page
    register(newUser).then(res => {
        this.props.history.push(`/login`)
    })
}
```

The Register page, like the Login page, uses the register function defined in the user functions file to handle a register request. If the user has successfully registered an account, the application will redirect to the login page where the user can then log in to their newly created account. The register form is a simple HTML inspired form.

3.2.2 Flask

Flask is a micro web framework written in Python [10], [23]. It is classified as a micro-framework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself. Some of the main features of Flask are:

- Development server and debugger
- Integrated support for unit testing
- Support for secure cookies (client side sessions)
- 100% WSGI 1.0 compliant

For this project, Flask was used as the middle-man for the React application and MongoDB database. Certain functionality, such as enabling users to register, login, upload sorts, etc. are written in Flask which are then accessed by the React application when needed. The Flask application is hosted on PythonAnywhere.

Advantages

There are a number of advantages to using Flask:

- **Flexibility** - There are very few parts of Flask that cannot be easily and safely altered, due to its simplicity and minimality.
- **Modularity** - Modular code provides a huge number of benefits. With Flask, you have the ability to create multiple Flask applications or servers, distributed across a large network of servers, each with specific purposes. This creates more efficiency, better testability, and better performance.
- **Performance** - A micro framework can be thought of as being slightly more "low-level" than something like Django. There are fewer levels of abstraction between the user and the database, the requests, the cache, etc. so performance is inherently better from the start.

- **Scalable** - Flask can be argued to be more scalable than monoliths if using modern methods. Today, applications are often running in containers or using cloud computing with auto-scaling. Applications do not typically "scale" themselves. The infrastructure scales. With a smaller application, it's easier to deploy instances across thousands of server easily to handle increased traffic/load. For example, it's partly the reason why Pinterest needed to migrate from Django to Flask as they grew to support more of a micro-services pattern.
- **Simpler Development** - If one understands Python well, then you'll be able to move around and contribute to a Flask application easily. It's less opinionated so there are fewer standards to learn.

Disadvantages

There are a few disadvantages to using Flask:

- **Community** - The monoliths provide such a large toolset, focused on providing solutions for a larger set of use cases out of the box, that they typically have a larger community. This means more eyes on the code, more code reviews, and better-tested core code. This is a little bit of a generalization though.
- **Fewer tools** - You don't have a full toolset underneath you. So you may need to build more on your own or search out extensions/libraries from the community.
- **Not standardized** - While Flask is simple, it's not very opinionated. A Python developer without Flask experience will get adjusted to a Flask application quicker than a Python developer without Django experience would get adjusted to a Django application. But Django is building up a large group of talent who knows the framework very well. A Python developer with Django experience will get adjusted to a new Django app quicker than a Python developer with Flask experience would get adjusted to a large Flask application.

Purpose in application

The backend aspect of the application was written in Python/Flask [2]. The serve.py script handles requests made by the user, such as attempting to register for an account and attempting to login. Below discusses how this is achieved:

```
# Flask server - Used for user authentication
# This code is hosted on PythonAnywhere. React makes requests to the Flask server on PythonAnywhere
#
# If you wish to run the React application locally, you must also run this Python script if you wish to utilize
# the functionality specific to users, such as registering/logging in, uploading previous sorts
from flask import Flask, jsonify, request, json
from flask_pymongo import PyMongo
import pymongo
from bson.objectid import ObjectId
from datetime import datetime
from flask_bcrypt import Bcrypt
from flask_cors import CORS
from flask_jwt_extended import JWTManager, create_access_token
from pprint import pprint

app = Flask(__name__)

app.config['MONGO_DBNAME'] = 'algorithms_visualizer'
app.config['MONGO_URI'] = 'mongodb://admin:admin123@ds217809.mlab.com:17809/algorithms_visualizer?retryWrites=false'
app.config['JWT_SECRET_KEY'] = 'secret'

mongo = PyMongo(app)
bcrypt = Bcrypt(app)
jwt = JWTManager(app)

cors(app)
```

The user authentication aspect uses a lot of different libraries to achieve the end result. The most notable libraries used and how they fit into this script are as follows:

- Flask-PyMongo - MongoDB was chosen as one of the databases used in this application. MongoDB is an open source database that stores flexible JSON-like "documents," which can have any number, name, or hierarchy of fields within, instead of rows of data as in a relational database. MongoDB as a persistent, searchable repository of Python dictionaries - this is how PyMongo actually represents MongoDB documents.
- Flask-Bcrypt - Flask-Bcrypt is a Flask extension that provides bcrypt hashing utilities.
- Flask-CORS - A Flask extension for handling Cross Origin Resource Sharing (CORS).
- Flask-JWT-Encoded - An open source Flask extension that provides JWT support. Adds support for using JSON Web Tokens (JWT) to Flask for protecting views.

To connect to the Mongo database, the database name, it's URI, and the secret key are defined. PyMongo will then connect to the MongoDB database. The Flask app object is also passed to Bcrypt, and the Flask-JWT extension is setup [20].

```
@app.route('/register', methods=['POST'])
def register():
    users = mongo.db.users
    first_name = request.get_json()['first_name']
    last_name = request.get_json()['last_name']
    email = request.get_json()['email']
    password = bcrypt.generate_password_hash(request.get_json()['password']).decode('utf-8')
    created = datetime.utcnow()

    user_id = users.insert({
        'first_name': first_name,
        'last_name': last_name,
        'email': email,
        'password': password,
        'created': created
    })

    new_user = users.find_one({
        '_id': user_id
    })

    result = {'email': new_user['email'] + ' registered'}

    return jsonify({
        'result': result
    })
```

The register method allows a user to create an account. Firstly, the script gets a handle on the database itself using `users = mongo.db.users`. Four values are passed to the database: the user's first name, the user's last name, the user's email, and the user's password.

```
first_name = request.get_json()['first_name']
last_name = request.get_json()['last_name']
email = request.get_json()['email']
password = bcrypt.generate_password_hash(request.get_json()['password']).decode('utf-8')
```

In the case of the user's password, Bcrypt is used to generate a password hash of the user's password before being passed to the database. When inserting a user into the database, the time and date the user's account was created is also passed to the database.

`user_id = users.insert(...` will then create a new document (user) in the MongoDB database with these details.

```python
@app.route('/login', methods=['POST'])
def login():
    users = mongo.db.users
    email = request.get_json()['email']
    password = request.get_json()['password']
    result = ''

    response = users.find_one({ 'email': email })
    unique = users.find({'email': { "email": email}}).count()
    print(unique)

    if response:
        if bcrypt.check_password_hash(response['password'], password):
            access_token = create_access_token(identity = {
                'first_name': response['first_name'],
                'last_name': response['last_name'],
                'email': response['email']
            })

            result = jsonify({ 'Successfully logged in. Token': access_token })
        else:
            result = jsonify({ 'ERROR: Invalid username or password' })
    else:
        result = jsonify({ 'result': 'No results found' })
    return result
```

The login method allows a user to a login into their account. To log in, the user needs to provide two details: their email and their password. The script will search the database by user's emails. If the script finds the provided email and the hash of the provided password is equal to that of the hashed password linked to the account the user is attempting to log into, the script will create an access token and the user will be logged into the application.

Again, the script gets a handle on the database itself using `users = mongo.db.users`. Using `response = users.find_one({ 'email': email })`, the script will try and find a matching email with the one provided.

```
if response:
if bcrypt.check_password_hash(response['password'], password):
```

This will check if `response` returns true and if the hash of the password associated with the account found is correct. If so, the access token will created and the user will be logged in.

3.2.3 PythonAnywhere

PythonAnywhere is an online integrated development environment and web hosting service (Platform as a service) based on the Python programming language. it provides in-browser access to server-based Python and Bash command-line interfaces, along with a code editor with syntax highlighting. Program files can be transferred to and from the service using the user's browser. Web applications hosted by the service can be written using any WSGI-based application framework. For this project, the user authentication side of things was initially handled entirely by Firebase. However, as suggested by my supervisor, I decided to implement registering a user, logging a user into the application etc. myself. The functionality for this was written in Python and could be accessed by the application by using a proxy to delegate any requests made by the application to the Flask server. However, to enable the functionality to be accessed from anywhere (and, in turn, remove the need to run the Flask server alongside the application everytime), I decided to use PythonAnywhere [35]. While the application itself, can be run locally or accessed from the hosting site, I decided to host the Flask server on PythonAnywhere. This removes the need to run the script as well when running/accessing the main application. To enable requests to be made and handled by the Flask server on PythonAnywhere, a proxy must be setup to handle specific requests. A proxy, in general, is a server or service which can introduce additional layers in communication to obfuscate or modify content. In package.json, the prxoy is defined as follows:

```
      "last 1 safari version"
    ]
  },
  "proxy": "http://kniland97.eu.pythonanywhere.com"
}
```

What this does is when a user makes a request to register for an account or log into their account, the application will send the request to the link shown above (which is essentially the Flask server as seen in the last section). The request will then be handled and processed, with the appropriate action being carried out.

Advantages

There are a number of advantages to using PythonAnywhere:

- **Always running** - Even on a free tier account, PythonAnywhere never sleeps (compared to something like Heroku). This means real time services are viable with PythonAnywhere. This proved to be very beneficial for this application since the Flask server (which is needed to allow users to register and login to an account) can always be available and requests can be made to it any time.
- **Easy Setup** - It does not take long to have Python-backed website up and running. In the case of the Flask server for this application, the file simply needed to be uploaded, the appropriate libraries downloaded, and the flask file itself imported into the WSGI file (which would then 'deploy' it and allow requests to be made to it).
- **Free** - PythonAnywhere is free to use. On a free tier account, web applications stay running 24/7 for 3 months. After 3 months, it will shut down. However, one needs only to start it up again.

Disadvantages

There are a few disadvantages to using PythonAnywhere:

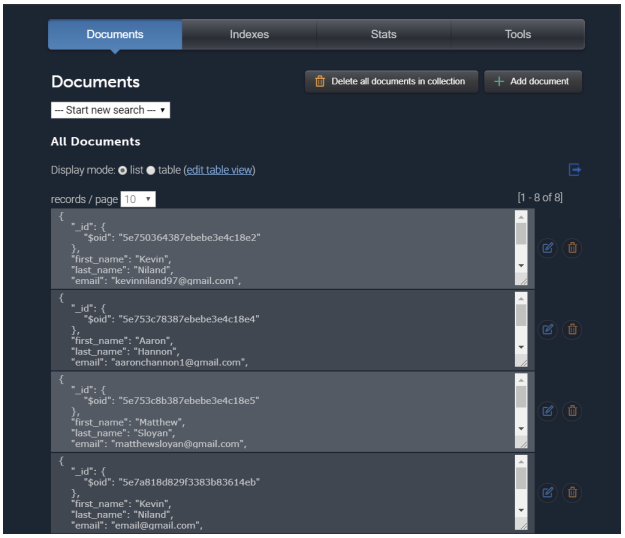
- **Python-only on the server side** - You are free to use JavaScript in your web pages and so on, but you can't use Rails or Node on the server side of things.
- **No WebSocket Support** - A WebSocket is a computer communications protocol, providing full-duplex communication channels over a single TCP connection. PythonAnywhere currently doesn't support WebSockets.

3.2.4 MongoDB

MongoDB is a cross-platform document-oriented database program [21]. Classified as a NoSQL database program, MongoDB uses JSON-like documents with schema. Some of the main features of MongoDB are:

- Ad-hoc queries (MongoDB supports field, range query, and regular-expression searches)
- Indexing (Fields in a MongoDB document can be indexed with primary and secondary indices)
- File storage (MongoDB can be used as a file system, called GridFS, with load balancing and data replication features over multiple machines for storing files)

For this project, MongoDB was used as the main database. It stores user details and user sorts. To write to the database, a request is first made through the application. Using a proxy, these requests are delegated to a flask application. The flask application then processes the request and will then access the database to perform a certain action (such as writing user details to the database, accessing user details, storing user sorts, etc).



Advantages

There are a number of advantages to using MongoDB:

- **Schema-less NoSQL Database** - MongoDB is a schema-less NoSQL database, meaning there is no need to design the schema of the database when using MongoDB. The code defines the schema.
- **Performace** - Performance of MongoDB is much higher than compared to any relational database.
- **Internal Memory** - MongoDB uses internal memory for storage which enables faster access to the data.
- **No Joins** - MongoDB doesn't use complex joins. There is no relationship among data in MongoDB.
- **JSON** - MongoDB supports JSON. Because MongoDB uses JSON format to store data, it is very easy to store arrays and objects.

Disadvantages

There are a few disadvantages to using MongoDB:

- **High Memory** - MongoDB uses high memory for data storage.
- **Document Size Limit** - MongoDB has a limit to the size of documents it can store.
- **Lack Of Transaction Support** - MongoDB does not support transactions.

3.2.5 Firebase

Firebase is a mobile and web application development platform developed by Firebase, Inc [9]. With this project, Firebase was used to store the saved sorts and to host the application.

Storing Sorts

To store and upload sorts to the database, a library called React Firebase File Uploader [27] was used. This allows one to upload images, videos, and other files to Firebase Storage.

```
handleUploadStart = () => {
    // On upload start, progress of upload is set to 0
    this.setState ({
      progress: 0
    })
}

handleUploadSuccess = filename => {
    this.setState ({
      sort: filename,
      progress: 100
    })

    // File will be uploaded to the sorts folder and download URL will be saved to state
    firebase.storage().ref('sorts').child(filename).getDownloadURL()
    .then(url => this.setState ({
      videoURL: url
    }))
}
```

As soon as the user selects a file to upload, these functions will be called and will execute accordingly. Once the file has finished uploading, it will be stored in the `sorts` directory in Firebase Storage, from which it will then be retrieved and listed.

```
/**
 * Get all files from storage and list them
 */
firebase.storage().ref().child('sorts/').listAll().then(function(res) {
    res.items.forEach(function(ref) {
        ref.getMetadata().then(function(url) {
            ref.getDownloadURL().then(function(file) {
                let new_html = '';

                new_html += '<tr>';
                new_html += '<td>';
                new_html += url.name;
                new_html += '</td>';
                new_html += '<td>';
                new_html += file;
                new_html += '</td>';
                new_html += '</tr>';

                $('#List').find('tbody').append(new_html);
            });
        });
    });
});
```

To retrieve and list all files from Firebase Storage, there seems to be no clear cut way to do it purely with React. To achieve this, I had to use a mix of HTML and JavaScript [16]. As can be seen above, Firebase itself has a lot of inbuilt functions to retrieve files from storage. First, all files stored in storage were retrieved. A download link for each file was then generated. To get around this, I had to use a library called React HTML Parser [28].

```
/**
 * Using React HTML Parser, display the HTML required to list all files in storage. React HTML Parser is a
 * utility for converting HTML strings into React components. Avoids the use of dangerouslySetInnerHTML and
 * converts standard HTML elements, attributes and in-line styles into their React equivalents
 *
 * This is was the only way I found that could list all files from storage - seems very difficult to do in React itself
 */
const html = '<style>table.List { table-layout: fixed; } #List { font-family: "Trebuchet MS", Arial, Helvetica, sans-serif;
// const rectangle = '<style>.square { height: 300px; width: 300px; background-color: #555; } </style><body><div class="squa

$('#List').find('tbody').html('');
```

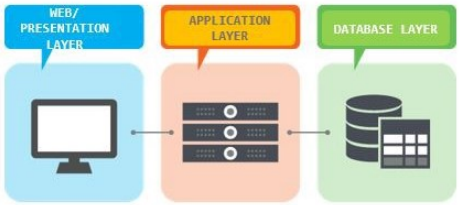
This allows you define HTML code which can then be rendered using `{ReactHtmlParser(html)}` in the `render()` method. To style the HTML, I simply added style tags in the constant `html`.

Chapter 4

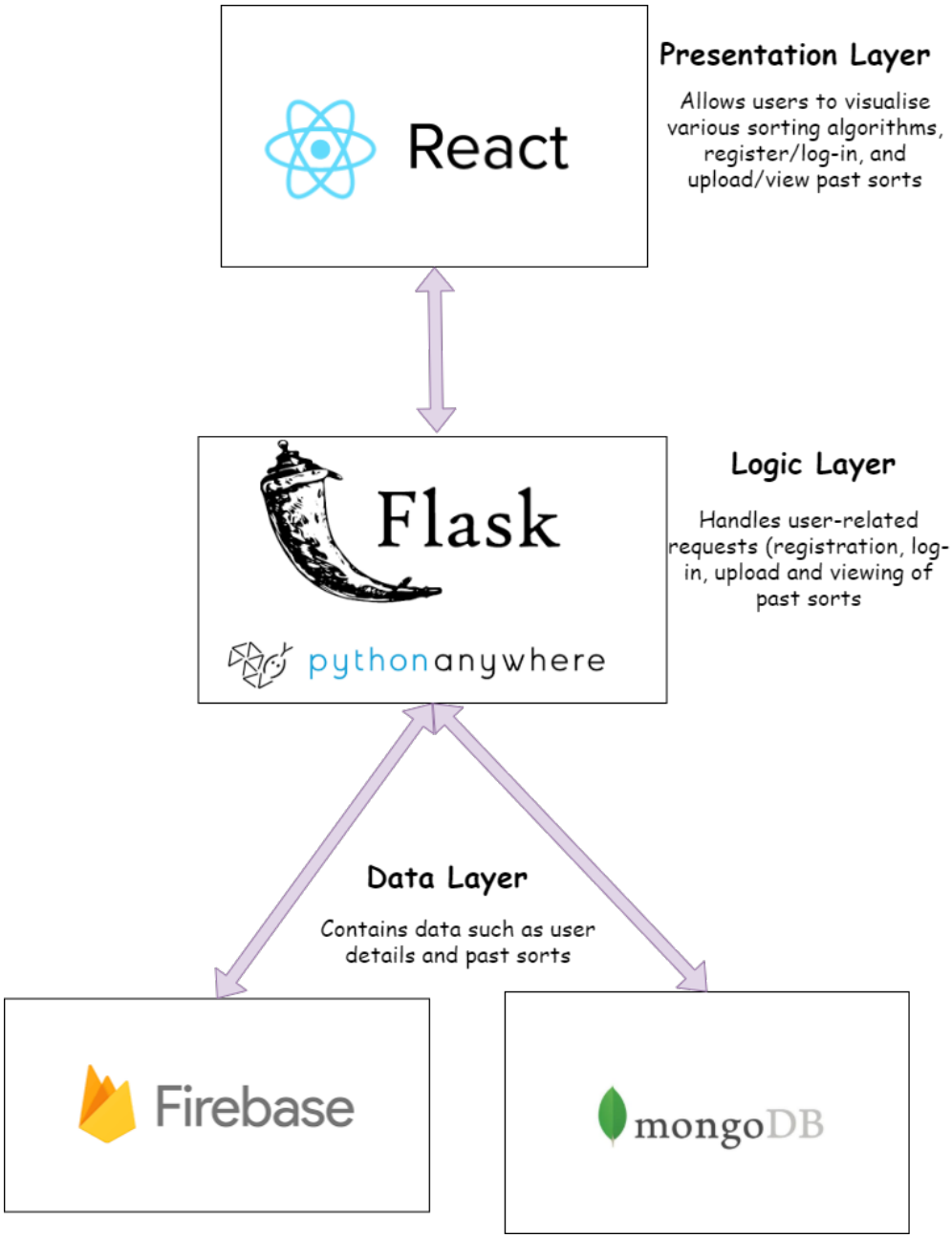
System Design

This chapter discusses the system design, analysing the various aspects of the project and purpose of each component such as the visualisation, the sorting algorithms, user authentication, the server, etc.

4.1 Overview



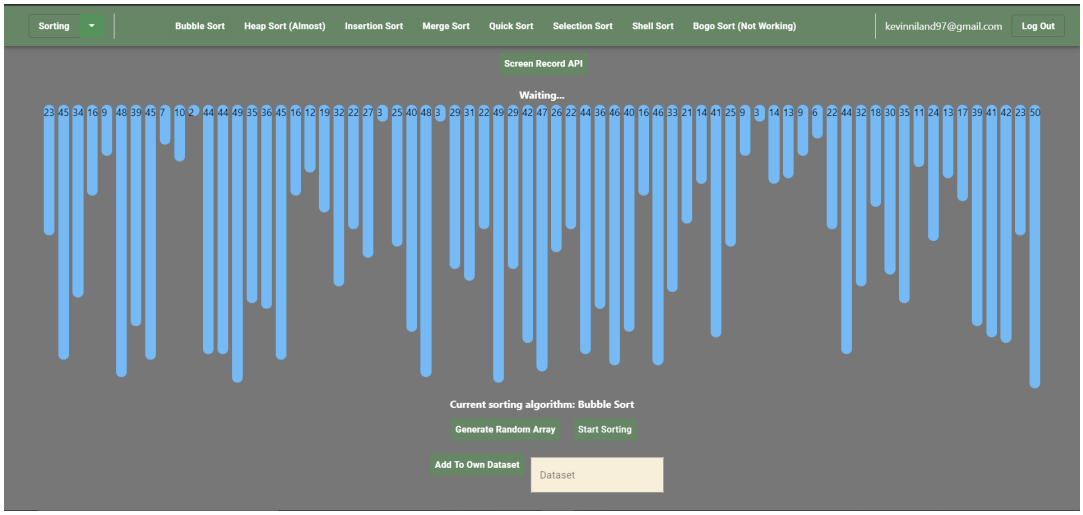
This project utilises the multi-tier architecture (often referred to as n-tier architecture) platform. Multi-tier architecture or multilayered architecture is a client–server architecture in which presentation, application processing, and data management functions are physically separated. The most widespread use of multi-tier architecture is the three-tier architecture. This project consists of three main components, all working together: The web application, which represents the presentation layer, allows users to visualise various sorting algorithms, register and login, upload and view past sorts from other users. The Flask server, which represents the logic tier, handles requests such as account registration and log in, and uploading and retrieving of sorts by users. The databases: the MongoDB database which stores user details and the Firebase database which stores all previous sorts.


React
Presentation Layer
Allows users to visualise various sorting algorithms, register/log-in, and upload/view past sorts
Flask
pythonanywhere
Logic Layer
Handles user-related requests (registration, log-in, upload and viewing of past sorts
Data Layer
Contains data such as user details and past sorts
Firebase
mongoDB

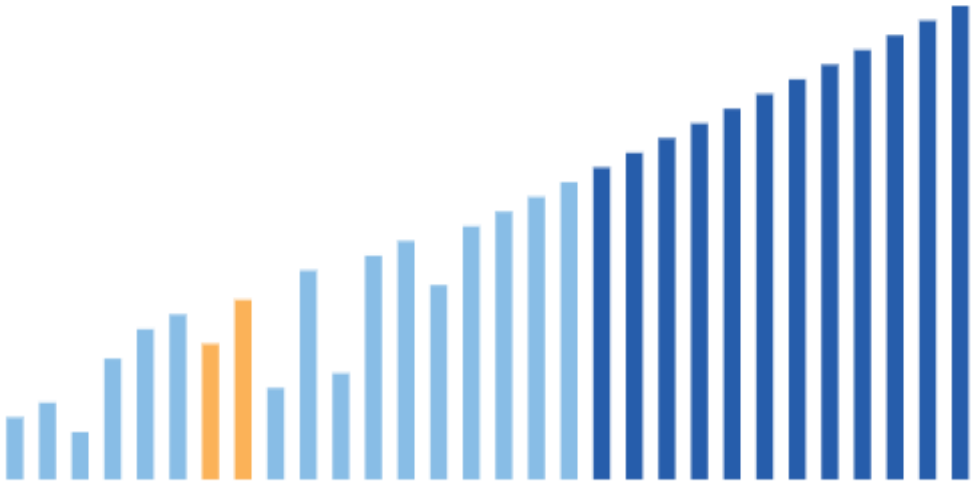
4.2 Web Application

The web application was designed and developed first as this is the main aspect of the project itself. The main objective of the web application was to allow a user to choose a sorting algorithm and be able to visualise it. Secondary objectives, which were developed at a later point, was to allow users to register and log in to an account, and have the ability to upload and view previous sorts. This was greatly aided by the decision to use ReactJS as discussed in Chapter 3 The web application consists of four main pages:

- **Sorting Page** - The sorting page, as shown in Figure 4.2.1, is the main page of the application. Allows a user to choose one of several sorting algorithms to visualise, generate a random array of elements to sort, utilize the ScreenFlow API and specify their own dataset to sort.
- **Login Page** - The login page handles user authentication and will login a user to the application if that user exists within the database. The user will then be redirected to the sorting page, with new functionality available such as accessing the ScreenFlow API and uploading one's own dataset.
- **Register Page** - The register page handles user authentication and will register a user with the application. The user will then be redirected to the login page, where they can then attempt to login.
- **Sorts Page** - The sorts page displays all previous sorts recorded and saved by past users.



4.2.1 Sorting



There are currently seven algorithms that can be visualised within the application. Currently, all sorting algorithms have been written using JavaScript [31]. They are as follows:

- Bubble Sort [3]
- Heap Sort
- Insertion Sort
- Merge Sort
- Quick Sort
- Selection Sort
- Shell Sort

Bubble Sort

```
export default class BubbleSort {
    /**
     * Performs the bubble sort algorithm
     *
     * Bubble sort, sometimes referred to as sinking sort, is a simple sorting algorithm that repeatedly
     * steps through the list, compares adjacent elements and swaps them if they are in the wrong order.
     * The pass through the list is repeated until the list is sorted. This algorithm is not suitable for large data
     * sets as its average and worst case complexities are of O(n2), where n is the number of items.
     *
     * @param {*} array - Array of items to be sorted
     * @param {*} sortHistory - Previous items that have been sorted
     * @param {*} highlightHistory - Previous items that have been...
     */
    static bubbleSort(array, sortHistory, highlightHistory) {
        this.clearArray(sortHistory);
        this.clearArray(highlightHistory);

        let counter = 0;

        for (let i = 0; i < array.length - 1; i++) {
            for (let j = 0; j < array.length - (i + 1); j++) {
                sortHistory.push(array.slice());
                highlightHistory.push([j + 1, j]);

                counter++;

                // If index j in the array is larger than index j + 1 in the array, swap them
                if (array[j] > array[j + 1]) {
                    this.swap(array, j, j + 1);
                }
            }
        }
    }
```

Bubble sort, sometimes referred to as sinking sort, is a simple sorting algorithm that repeatedly steps through the list, compares adjacent elements and swaps them if they are in the wrong order [7]. The pass through the list is repeated until the list is sorted. The algorithm, which is a comparison sort, is named for the way smaller or larger elements "bubble" to the top of the list.

This simple algorithm performs poorly in real world use and is used primarily as an educational tool. More efficient algorithms such as Tim Sort, or Merge Sort are used by the sorting libraries built into popular programming languages such as Python and Java.

How it works in the context of the application Bubble sort works by comparing adjacent pairs of objects in the array using nested for loops [4]. Sorted elements are pushed into `sortedElements` as a new array object. Elements that need to be sorted are pushed into `selectedElements`. If the objects are not in the correct ordered, they are swapped so that the largest of the two moves up.

Insertion Sort

```
export default class InsertionSort {
    /**
     * Performs the insertion sort algorithm
     *
     * Insertion sort is a simple sorting algorithm that builds the final sorted array (or list) one item at a time.
     * It is much less efficient on large lists than more advanced algorithms such as quicksort, heapsort, or merge sort.
     * This algorithm is not suitable for large data sets as its average and worst case complexities are of O(n2), where n is the
     * number of items.
     *
     * @param {*} array - Array of items to be sorted
     * @param {*} sortHistory - Elements that have been sorted
     * @param {*} selectedHistory - Elements that have been previously selected for sorting
     */
    static insertionSort(array, sortHistory, selectedHistory) {
        this.clearArray(sortHistory);
        this.clearArray(selectedHistory);

        let leftCol = -1;

        // Iterate through entire array
        for (let i = 0; i < array.length; i++) {
            leftCol = i;
            let temp = array[i];
            let j;

            for (j = i - 1; j >= 0 && array[j] > temp; j--) {
                array[j + 1] = array[j];

                sortHistory.push(array.slice());
                selectedHistory.push([j, leftCol]);
            }

            array[j + 1] = temp;
        }

        sortHistory.push(array.slice());
        selectedHistory.push([0, array.length - 1]);
    }
```

Insertion sort is a simple sorting algorithm that builds the final sorted array (or list) one item at a time [14]. It is much less efficient on large lists than more advanced algorithms such as Quick Sort, Heap Sort, or Merge Sort. However, insertion sort provides several advantages:

- **Efficient** - Efficient for (quite) small data sets, much like other quadratic sorting algorithms
- **Stable** - Does not change the relative order of elements with equal keys
- **Online** - Can sort a list as it receives it
- **Adaptive** - Efficient for data sets that are already substantially sorted:

When people manually sort cards in a bridge hand, most use a method that is similar to insertion sort.

How it works in the context of the application Insertion sort iterates, consuming one input element each repetition, and growing a sorted output list [5]. At each iteration, insertion sort removes one element from the input data, finds the location it belongs within the sorted list, and inserts it there. It repeats until no input elements remain.

Merge Sort

```
export default class MergeSort {
    /**
     * Performs the merge sort algorithm on one half of array
     *
     * Like QuickSort, Merge Sort is a Divide and Conquer algorithm. It divides input array in
     * two halves, calls itself for the two halves and then merges the two sorted halves.
     *
     * @param {*} array - Array to be sorted
     * @param {*} sortHistory - Elements that have been sorted
     * @param {*} selectedHistory - Elements that have been previously selected for sorting
     */
    static mergeSort(array, sortHistory, selectedHistory) {
        let step = 1;

        // While still able to step through the array
        while (step < array.length) {
            let left = 0;

            while (left + step < array.length) {
                this.mergeBottomUp(array, left, step, sortHistory, selectedHistory);

                left += step * 2;
            }

            step *= 2;
        }
    }
```

```
    /**
     * Performs merge sort on remaining half of array
     *
     * @param {*} array - Array to be sorted
     * @param {*} left - Starting index
     * @param {*} step - Amount to step forward in array
     * @param {*} sortHistory - Elements that have been sorted
     * @param {*} selectedHistory - Elements that have been previously selected for sorting
     */
    static mergeBottomUp(array, left, step, sortHistory, selectedHistory) {
        let right = left + step;
        let end = Math.min(left + step * 2 - 1, array.length - 1);
        let stepLeft = left;
        let stepRight = right;
        let temp = [];

        for (let i = left; i <= end; i++) {
            if ((array[stepLeft] <= array[stepRight] || stepRight > end) && stepLeft < right) {
                temp[i] = array[stepLeft];

                stepLeft++;
            } else {
                temp[i] = array[stepRight];

                stepRight++;
            }
        }

        for (let j = left; j <= end; j++) {
            array[j] = temp[j];
        }

        sortHistory.push(array.slice());
        selectedHistory.push([-1]);
    }
```

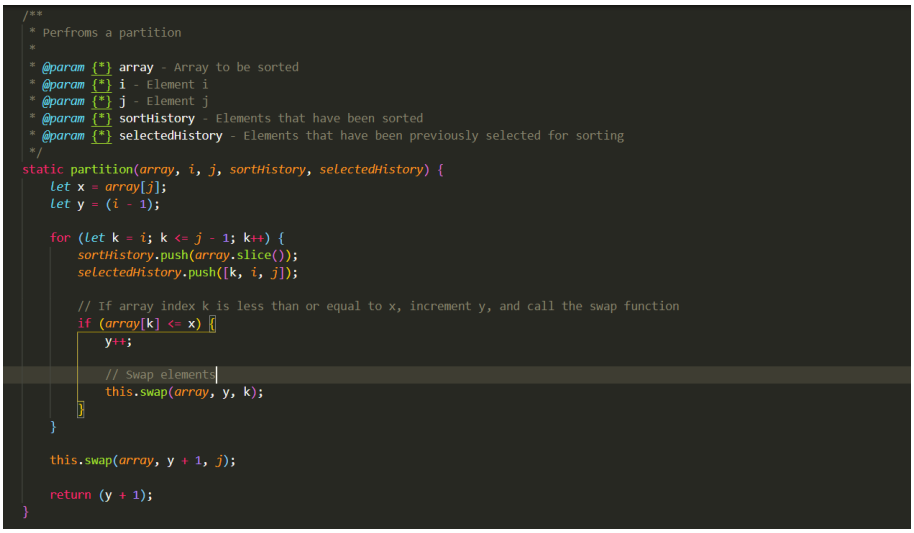
Merge Sort is an efficient, general-purpose, comparison-based sorting algorithm [18]. Most implementations produce a stable sort, which means that the order of equal elements is the same in the input and output. It divides input array in two halves, calls itself for the two halves and then merges the two sorted halves.

How it works in the context of the application A middle point is found and the array is divided in two halves. The sub-arrays are then recursively sorted in each of the two sub-problems created by the divide step. That is, recursively sort the sub-array `array[p..q]` and recursively sort the sub-array `array[q+1..r]`. The two sorted sub-arrays are then merged back into a single sorted array [17].

Quick Sort

```
export default class QuickSort {
    /**
     * Performs the quick sort algorithm
     *
     * Like Merge Sort, QuickSort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given
     * array around the picked pivot. There are many different versions of quickSort that pick pivot in different ways. This implementation
     * of quick sort is a recursive implementation of quick sort. This algorithm is not suitable for large data sets as its average and
     * worst case complexities are of O(n2), where n is the number of items.
     *
     * @param {*} array - Array to be sorted
     * @param {*} sortHistory - Elements that have been sorted
     * @param {*} selectedHistory - Elements that have been previously selected for sorting
     */
    static quickSort(array, sortHistory, selectedHistory) {
        this.clearArray(sortHistory);
        this.clearArray(selectedHistory);

        this.quickSortIterative(array, 0, array.length - 1, sortHistory, selectedHistory);
    }
```

```
    /**
     * Quick sort iterative
     *
     * @param {*} array - Array to be sorted
     * @param {*} i - Element i
     * @param {*} j - Element j
     * @param {*} sortHistory - Elements that have been sorted
     * @param {*} selectedHistory - Elements that have been previously selected for sorting
     */
    static quickSortIterative(array, i, j, sortHistory, selectedHistory) {
        // Stack of elements
        let stack = [];

        // Top of stack
        let top = -1;

        stack[++top] = i;
        stack[++top] = j;

        // While top of stack is greater than or equal to 0
        while (top >= 0) {
            j = stack[top--]; // Take item of stack and store in j
            i = stack[top--]; // Take item of stack and store in i

            // Partition array
            let partition = this.partition(array, i, j, sortHistory, selectedHistory);

            if (partition - 1 > i) {
                stack[++top] = i;
                stack[++top] = partition - 1;
            }

            if (partition + 1 < j) {
                stack[++top] = partition + 1;
                stack[++top] = j;
            }
        }
    }
```



```
/**
 * Perfroms a partition
 *
 * @param {*} array - Array to be sorted
 * @param {*} i - Element i
 * @param {*} j - Element j
 * @param {*} sortHistory - Elements that have been sorted
 * @param {*} selectedHistory - Elements that have been previously selected for sorting
 */
static partition(array, i, j, sortHistory, selectedHistory) {
    let x = array[j];
    let y = (i - 1);

    for (let k = i; k <= j - 1; k++) {
        sortHistory.push(array.slice());
        selectedHistory.push([k, i, j]);

        // If array index k is less than or equal to x, increment y, and call the swap function
        if (array[k] <= x) {
            y++;

            // Swap elements
            this.swap(array, y, k);
        }
    }

    this.swap(array, y + 1, j);

    return (y + 1);
}
```

Quick Sort is an efficient sorting algorithm [25]. Quick Sort is a divide-and-conquer algorithm. It works by selecting a 'pivot' element from the array and partitioning the other elements into two sub-arrays, according to whether they are less than or greater than the pivot. The sub-arrays are then sorted recursively. This can be done in-place, requiring small additional amounts of memory to perform the sorting.

Quick Sort is a comparison sort, meaning that it can sort items of any type for which a "less-than" relation (formally, a total order) is defined. Efficient implementations of Quick Sort are not a stable sort, meaning that the relative order of equal sort items is not preserved.

Mathematical analysis of Quick Sort shows that, on average, the algorithm takes O(n log n) comparisons to sort n items. In the worst case, it makes O(n2) comparisons, though this behavior is rare.

How it works in the context of the application A "pivot" item is found in the array. A left and right pointer is started in the first and last item in the array, respectively. While the value at the left pointer in the array is less than the pivot value, move the left pointer to the right (add 1). This continues until the value at the left pointer is greater than or equal to the pivot value. While the value at the right pointer in the array is greater than the pivot value, move the right pointer to the left (subtract 1). This continues until the value at the right pointer is less than or equal to the pivot value. If the left pointer is less than or equal to the right pointer, the values at these locations in the array are swapped. The left pointer is moved to the right by one and the right pointer is moved to the left by one. If the left pointer and right pointer don't meet, the process is started again [24].

Selection Sort

```
export default class SelectionSort {
    /**
     * Performs the selection sort algorithm
     *
     * Selection sort is a simple sorting algorithm. This sorting algorithm is an in-place comparison-based
     * algorithm in which the list is divided into two parts, the sorted part at the left end and the unsorted
     * part at the right end. Initially, the sorted part is empty and the unsorted part is the entire list. The
     * smallest element is selected from the unsorted array and swapped with the leftmost element, and that element
     * becomes a part of the sorted array. This process continues moving unsorted array boundary by one element to
     * the right. This algorithm is not suitable for large data sets as its average and worst case complexities
     * are of O(n2), where n is the number of items.
     *
     * @param {*} array
     * @param {*} sortHistory
     * @param {*} highlightHistory
     */
    static selectionSort(array, sortHistory, highlightHistory) {
        this.clearArray(sortHistory);
        this.clearArray(highlightHistory);

        let leftCol = -1;

        /**
         * Iterate through the array
         */
        for (let i = 0; i < array.length; i++) {
            let min = i;
            leftCol = min;

            // Iterate through the array, starting at the index immediately after index i
            for (let j = i + 1; j < array.length; j++) {

                // If array index j is less than the minimum, j is the new minimum
                if (array[j] < array[min]) {
                    min = j;
                }

                sortHistory.push(array.slice());
                highlightHistory.push([j, leftCol, min]);
            }

            // If i doesn't equal the minimum, swap i and min
            if (i !== min) {
                this.swap(array, i, min);
            }
        }

        sortHistory.push(array.slice());
        highlightHistory.push([-1, array.length - 1]);
    }
```

Selection sort is an in-place comparison sorting algorithm [34]. It has an O(n2) time complexity, which makes it inefficient on large lists, and generally performs worse than the similar insertion sort. Selection sort is noted for its simplicity and has performance advantages over more complicated algorithms in certain situations, particularly where auxiliary memory is limited.

The algorithm divides the input list into two parts: a sorted sublist of items which is built up from left to right at the front (left) of the list and a sublist of the remaining unsorted items that occupy the rest of the list. Initially, the sorted sublist is empty and the unsorted sublist is the entire input list. The algorithm proceeds by finding the smallest (or largest, depending on sorting order) element in the unsorted sublist, exchanging (swapping) it with the leftmost unsorted element (putting it in sorted order), and moving the sublist boundaries one element to the right.

The time efficiency of selection sort is quadratic, so there are a number of sorting techniques which have better time complexity than selection sort. One thing which distinguishes selection sort from other sorting algorithms is that it makes the minimum possible number of swaps, n 1 in the worst case.

How it works in the context of the application For the first position in the sorted list, the whole list is scanned sequentially. The whole array is sorted to find the lowest value. After one iteration, the minimum value is stored in the first position of the sorted list. For the second position, the rest of the list is scanned in a linear manner. The same process is applied to the rest of the items in the array [33].

Shell Sort

```
 *
 * @param {*} array - Array to be sorted
 * @param {*} sortedElements - Elements that have been sorted
 * @param {*} selectedElements - Elements that have been previously selected for sorting
 */
static shellSort(array, sortedElements, selectedElements) {
    // Clear sortedElements and selectedElements
    this.clearArray(sortedElements);
    this.clearArray(selectedElements);

    for (let h = array.length; h > 0; h = parseInt(h / 2, 10)) {
        for (let i = h; i < array.length; i++) {
            /**
             * The slice() method returns a shallow copy of a portion of an array into a
             * new array object selected from begin to end (end not included) where begin and
             * end represent the index of items in that array. The original array will not be
             * modified. Push this portion into sortedElements
             */
            sortedElements.push(array.slice());
            selectedElements.push([i + 1, i]);

            let k = array[i];

            for (var j = i; j >= h && k < array[j - h]; j -= h)
            array[j] = array[j - h];
            array[j] = k;
        }
    }
}
```

Shell Sort is an in-place comparison sort [37]. It can be seen as either a generalization of sorting by exchange (bubble sort) or sorting by insertion (insertion sort). The method starts by sorting pairs of elements far apart from each other, then progressively reducing the gap between elements to be compared. Starting with far apart elements, it can move some out-of-place elements into position faster than a simple nearest neighbor exchange. The running time of Shell Sort is heavily dependent on the gap sequence it uses. For many practical variants, determining their time complexity remains an open problem.

How it works in the context of the application The idea of Shell Sort is to allow exchange of far items. In Shell Sort, we make the array h-sorted for a

large value of h. We keep reducing the value of h until it becomes 1. An array is said to be h-sorted if all sublists of every h'th element is sorted [36].

Bogo Sort and Heap Sort have been programmed as well, however, they are not fully working as of now and as such, they are not covered in this section.

4.2.2 Visualization

```
function ArrayElement(props) {
    const arrayElement = {
      element: {
        backgroundColor: props.color,
        borderBottomLeftRadius: 22,
        borderBottomRightRadius: 22,
        borderTopLeftRadius: 22,
        borderTopRightRadius: 22,
        color: 'green',
        display: 'inline-block',
        height: props.size * 9,
        margin: 3,
        width: 17
      },
      text: {
        color: 'black',
        display: 'inline-block',
      }
    }

    // Renders a bar for element in the array
    return (
      <div className='array-element' style={arrayElement.element}>
        <h7 style={arrayElement.text}>{ arrayElement.element.height / 9}</h7>
      </div>
    );
  }

export default ArrayElement;
```

```
<div className="bar-wrapper">
  {this.state.array.map((item, index) => <ArrayElement key={index} size={item} color={this.setColor(this.state.isSelected, index)}/>)}
</div>
```

The ArrayElement renders a bar that will represent each element in the array. In MainPage, each element and it's index will be mapped and then visually represented by the bar generated using ArrayElement [12].

4.3 Flask Server

The Flask server, which is hosted on PythonAnywhere, is the middle-man of the entire application and allows the web application to communicate with the database. The server was developed and integrated into the web application secondly alongside the database. It handles requests, such as user login requests, user registration requests, uploading saved sorts, etc. made by the web application and performs the appropriate action. The database is then read and/or updated based on this.

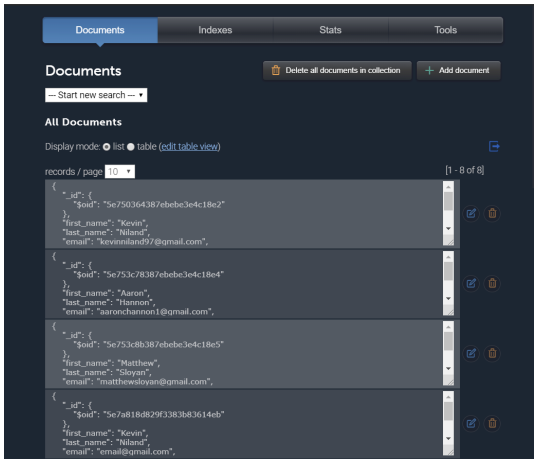
```
42    },
43    "proxy": "http://kniland97.eu.pythonanywhere.com"
44  }
```

4.4 Databases

This application utilizes two databases: a MongoDB database, which is used to store user details and a Firebase database, which stores all previous sorts.

4.4.1 MongoDB

MongoDB was chosen as the database to store user details as MongoDB stores data records as BSON documents. BSON is a binary representation of JSON documents, though it contains more data types than JSON. I thought this would be the best way to save user details since there are several libraries, such as bcrypt, that could work hand-in-hand. The value of a field can be any of the BSON data types, including other documents, arrays, and arrays of documents.



4.4.2 Firebase

Firebase was chosen as the database to store previous sorts as it provides an easy and simple way to store files. There are a number of React libraries, such as React File Uploader, that have been specifically developed to work with Firebase which provides a simple and easy way to upload a file to Firebase. Firebase itself also provides a number of methods, such as getMetadata() which will retrieve the file type and getDownloadURL() which will retrieve the download URL for each file, that enable an application to retrieve these files easily.

Firebase Storage was designed specifically to allow users to upload files, such as images and videos. Data is stored in a Google Cloud Storage bucket, an exabyte scale object storage solution with high availability and global redundancy.

Chapter 5

System Evaluation

This chapter evaluates the system, analysing the various aspects of the project to ensure the requirements of the project were met and discusses the various types of testing that was performed throughout the software development cycle to ensure each of the requirements was met and the software was of high quality.

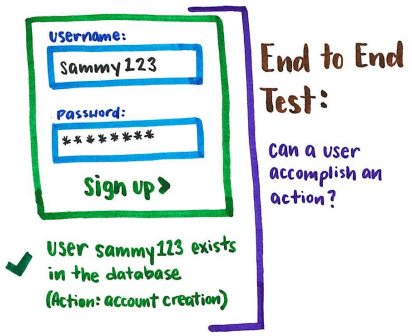
5.1 Overview

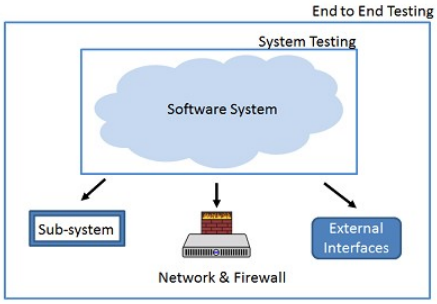
The system was designed and developed using the Test Driven Development (TDD) methodology. As each unit of code was written and before each new component was integrated into the system, appropriate tests were carried out and adjusted if needed. Because of this, a lot of bugs were caught early on, allowing them to be fixed with minimal disruption to the system as a whole. This also allowed the system to be redesigned early on when necessary.

The types of testing carried out were as follows:

- End-to-End Testing
- Exploratory Testing
- Functional Testing
- Graphical User Interface (GUI) Testing
- Integration Testing
- Unit Testing
- System Testing

5.2 End-to-End Testing

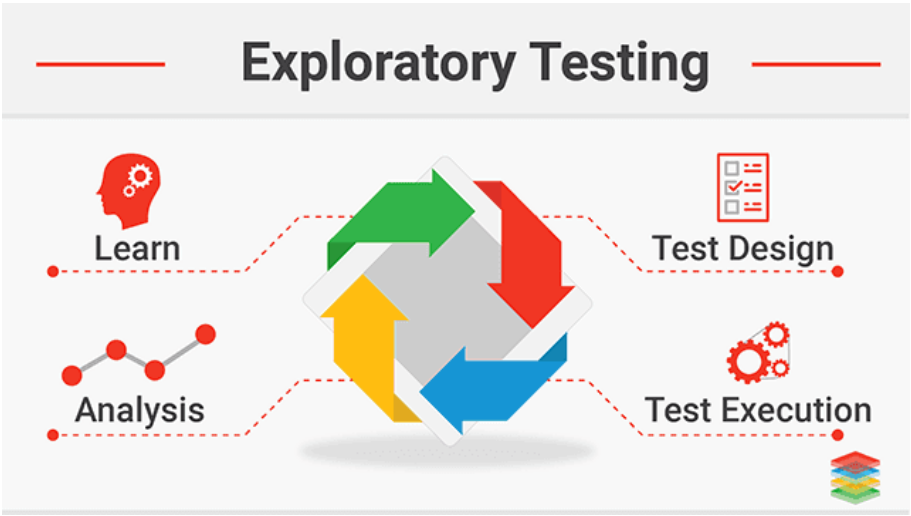




End-to-end testing is a technique used to test whether the flow of an application right from start to finish is behaving as expected. The purpose of performing end-to-end testing is to identify system dependencies and to ensure that the data integrity is maintained between various system components and systems.

The entire application is tested for critical functionalities such as communicating with the other systems, interfaces, database, network, and other applications. This type of testing was carried out at the end, when all features were fully implemented. To perform 'test cases', the system was used in a way that a user would typically use the system. All features were tested to see if the appropriate behaviour happened, such as visualizing an algorithm, registering for an account, logging into an account, recording a sort, uploading a sort, and viewing past sorts.

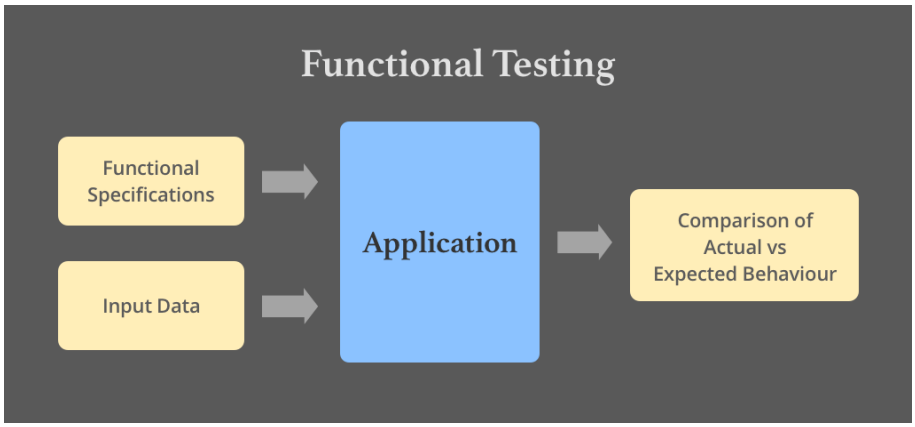
5.3 Exploratory Testing



Exploratory testing is a technique or a method whose aim is to discover the flaws during the process of software development. Exploratory Testing is widely used in Agile models and is all about discovery, investigation, and learning. It emphasizes personal freedom and responsibility of the individual tester.

This technique of testing is concerned about the qualitative assurance of the software. It is used to discover the anonymous issues during the process of development of software. Exploratory testing was carried out throughout development and helped prevent massive system flaws from remaining in the application and causing other bugs and/or having to fix these issues down the line, where it mightn't be possible to fix them or having to perform a complete overhaul of the system. One such flaw was in regards to an early design. Initially, there would be a 'master' navigation bar, with each of the application's pages appearing and being able to be navigated to from here. This caused the visualization aspect to not work properly, as the user would not be able to select an algorithm to visualize. This resulted in the system being redesigned to what it is now.

5.4 Functional Testing

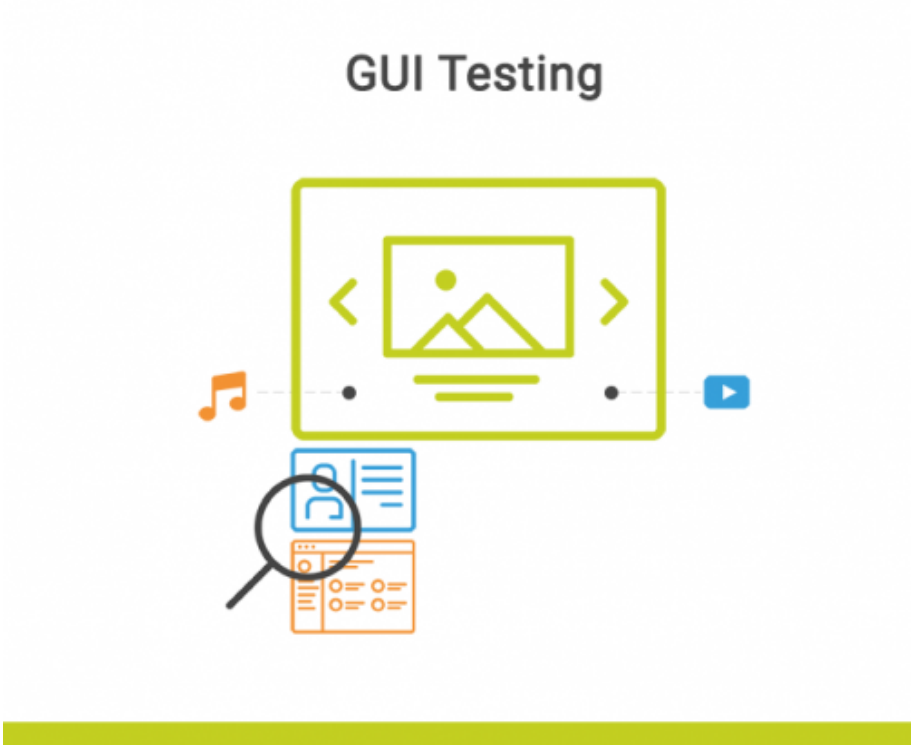


Functional Testing is a type of software testing whereby the system is tested against the functional requirements/specifications.

Functions (or features) are tested by feeding them input and examining the output. Functional testing ensures that the requirements are properly satisfied by the application. This type of testing is not concerned with how processing occurs, but rather, with the results of processing. It simulates actual system usage but does not make any system structure assumptions.

During functional testing, Black Box Testing technique is used in which the internal logic of the system being tested is not known to the tester.

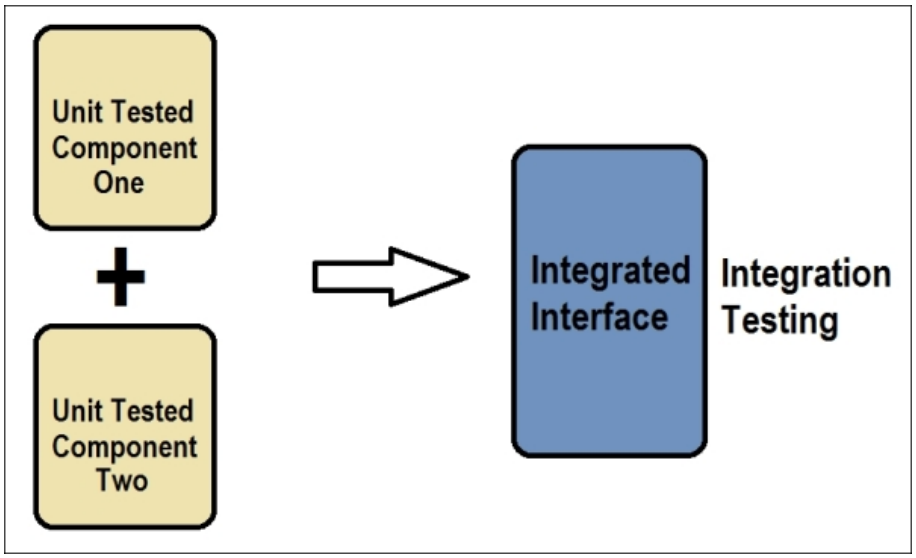
5.5 Graphical User Interface (GUI) Testing



Graphical User Interface (GUI) Testing is a software testing type that checks the Graphical User Interface of the application under test. GUI testing involves checking the screens with the controls like menus, buttons, icons, and all types of bars - toolbar, menu bar, dialog boxes, and windows, etc. The purpose of GUI Testing is to ensure UI functionality works as per the specification.

This type of testing, like exploratory testing, was carried out in conjunction with development and as such, aided in discovering bugs with the system. As discussed above, the system needed to be redesigned due a bug discovered, which GUI testing aided in as it was discovered in a previous design that the user was not able to visualize an algorithm due to an unresponsive GUI.

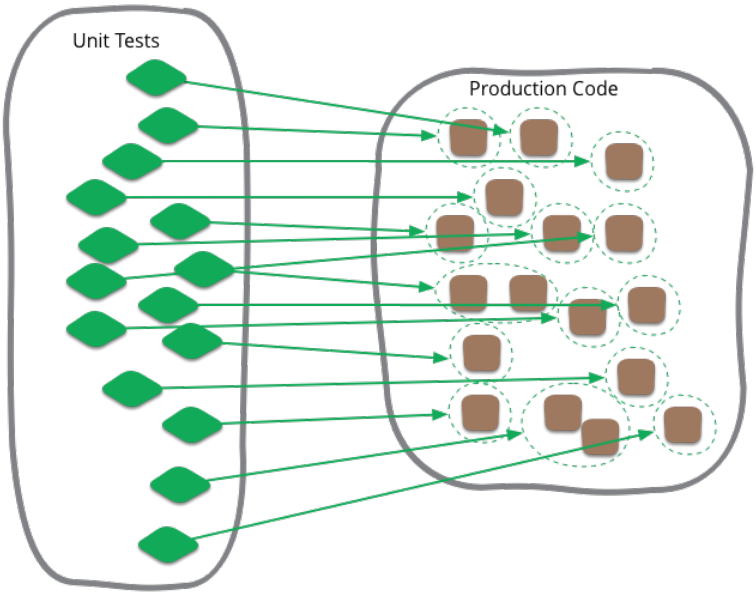
5.6 Integration Testing



Integration testing (sometimes called integration and testing, abbreviated IT) is the phase in software testing in which individual software modules are combined and tested as a group. Integration testing is conducted to evaluate the compliance of a system or component with specified functional requirements.

Integration testing was carried out throughout development and was integral in ensuring components worked with already integrated components without breaking the system. When a new component was added to the application, the entire application was tested in full before committing the changes to GitHub to make sure the new component didn't negatively affect or break the application.

5.7 Unit Testing



Unit Testing is a level of software testing where individual units/components of a software are tested. The purpose is to validate that each unit of the software performs as designed. A unit is the smallest testable part of any software. It usually has one or a few inputs and usually a single output.

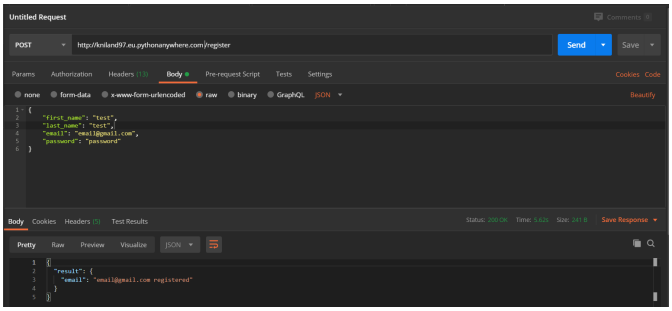
Unit testing was carried out on individual components before they were integrated into the system. Several different pieces of software were used to help test the application.

5.7.1 Postman

To test the user authentication aspect of the application, Postman was used to test requests made to the database to see if a user could successfully register for an account and log into an account. Requests could be made to the database

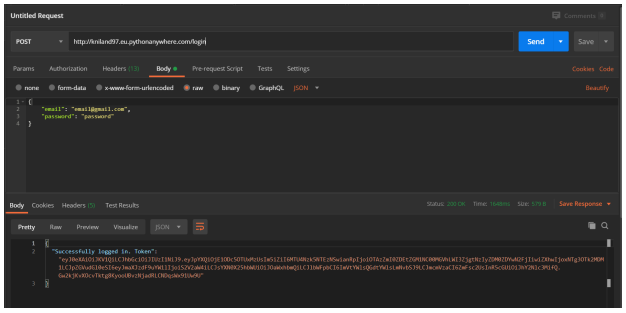
via the Flask Server (hosted on PythonAnywhere), which would then return a response.

Register



Using the POST method, a request is made to see if a user with the above details is able to successfully register an account with the application.

Login

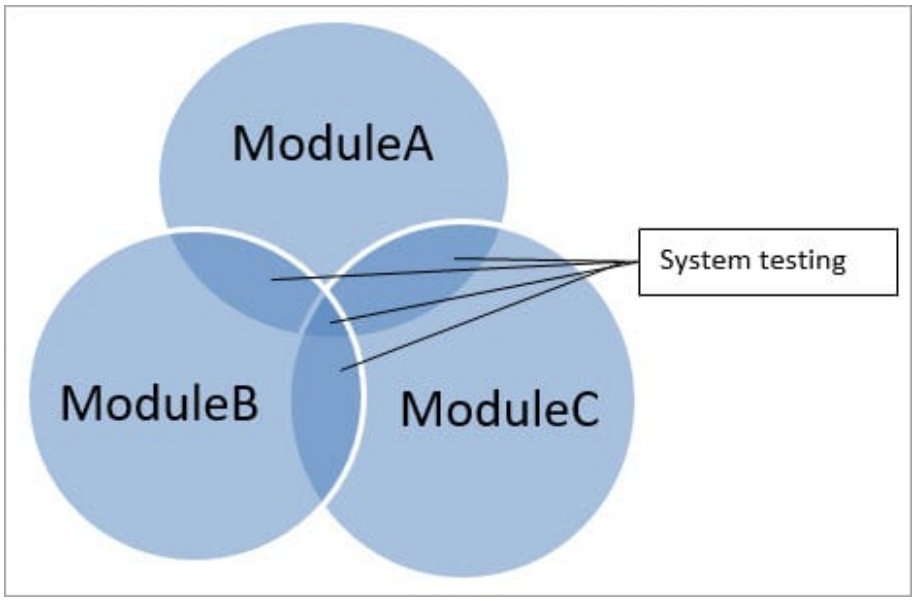


Using the POST method, a request is made to see if a user with the above details is able to successfully login to the application.

5.7.2 Sorting Algorithms

Before each sorting algorithm was implemented and added into the application, they were first written and tested on sample inputs and arrays.

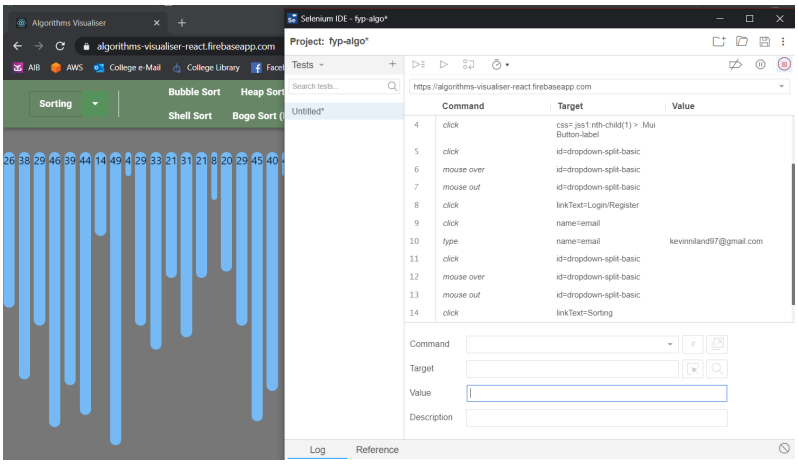
5.8 System Testing



System Testing is a level of testing that validates the complete and fully integrated software product. The purpose of a system test is to evaluate the end-to-end system specifications. Usually, the software is only one element of a larger computer-based system. Ultimately, the software is interfaced with other software/hardware systems.

5.8.1 Selenium

Selenium was used to test the system as a whole. Selenium is a portable framework for testing web applications. It also provides a test domain-specific language (Selenese) to write tests in a number of popular programming languages, including C, Groovy, Java, Perl, PHP, Python, Ruby and Scala. The tests can then run against most modern web browsers.



Selenium provides the option to record, edit, and debugging of functional tests. As can be seen, Selenium allows one to test various functionality of a web application such as button clicks, mouse over, mouse out, typing, etc. This functionality can be tested on the fly (as in, you can specify the URL of the web application and test the functionality of the web application yourself) or define a test case and then run it (such as clicking and verifying a specific button or entering text into a textbox).

Chapter 6

Conclusion

This chapter serves as a conclusion to the project. In it, the objectives of the project are analysed again briefly to evaluate whether or not they were met. It also analyses any improvements and/or changes that could be added/made to the project if it was to be developed again.

6.1 Objectives

One of the objectives of this project was to make an application that had the capabilities to visualize several different sorting algorithms, from start to finish. This application is intended and can be useful in an educational context, where users can use it to see how the programmed algorithms perform a sort of a provided array of elements. This was achieved by highlighting the elements currently chosen to be sorted and repeating this until the entire array is sorted.

Another one of the objectives of the project was to provide users of a system with the ability to register for an account, log in, and have the ability to access a screen record API and have the ability to upload the recording of the sort to a database, which can then be viewed by all users. The user registration/log-in system was achieved through a combination of Python and MongoDB. Creating a screen recording was done through the use of an external API, Screen Flow. Firebase was subsequently used to store these screen recordings.

6.2 Reflection

Overall, the objectives of the project were met and done to a standard that I was happy with. However, as with all software projects, there would be things I would do differently, incorporate different technologies to make a certain feature more robust, etc.

6.2.1 Downfalls and Improvements

If the project was to be repeated, a number of changes and improvements could be made. While user authentication does play a role, this could be further enhanced in several different ways such as further enhancing the viewing of all sorts aspect of the project. The user could have the option of uploading a dataset in addition to uploading a saved sort. The uploading of sorts could have been done in a better way. For instance, the Screen Record API used could have been integrated with the application more. This was attempted but due to the way the API was programmed, it does not seem like integrating it with this application in a more cohesive manner was possible at the time.

Another improvement could have been to include more sorting algorithms. The application could also have shown a visualization of the code itself. For example, when a certain sorting algorithm is picked, a window would appear with the appropriate code for the selected sorting algorithm. When the array of elements is being sorted, the appropriate logic in the code could be highlighted (such as the logic for swapping elements, comparison of different elements, etc.).

The application could also have included a way to pause the sort. To do achieve this, the sorting could be considered "finished". The half-sorted array would have to be saved in some way and then the currently selected sorting algorithm would have to "start over" again, now taking the half-sorted array as the new array to be sorted. A downfall of the application could be the user is unable to randomly generate an array of a certain size. The user is also unable to control the speed of the sort itself. A future improvement could be to add sliders that would alter the size of the array and speed of the sort.

6.2.2 Additions

As this is application that visualizes algorithms, a viable addition to the application could be to incorporate pathfinding algorithms. Pathfinding algorithms were recently covered on the course in the Artificial Intelligence module. This would come with it's own challenges, however. One challenge would be programming the pathfinding algorithms to work with the grid. If the application was to support the ability to dynamically generate a wall/maze inside the grid, these would have to be taken into account during runtime to make sure the pathfinding algorithms perform as intended.

6.2.3 Overall

Overall, this was a enjoyable project to work on and provided great experience in many different areas, such as undertaking and developing a project over a prolonged period of time, working with new technologies, and creating an application that can be deployed and used by multiple users. It also gave a more in-depth look at developing a project over a prolonged period of time,

with weekly meetings with the project supervisor to discuss the progress of the project and any problems associated with it.

Chapter 7

Appendices

This chapter contains any and all important resources related to the project, such as the source code,

7.1 Source code

The source code for this project is located on GitHub, at https://github.com/kevinniland97/Applied-Project-and-Minor-Dissertation. The repository contains the following main directories and files:

- backend - Directory containing the code for the Flask Server.
- dissertation - Contains files for the dissertation.
- public - The public folder contains the HTML file for tweaking minor aspects of the application, such as the title.
- src - Contains all code used to deve++lop the application.

7.2 Installation and Usage

Instructions on how to download, install and/or access this application can be found in the README.md of the GitHub repository. The README also contains a video that gives an overview of the repository and the project itself. The video can be found here: https://youtu.be/2GvhoUd2-po

7.3 Hosting

This project is hosted using Firebase Hosting and the project is also Dockerized.

7.3.1 Web Application

The application can be found at the following link: https://algorithms-visualiser-react.firebaseapp.com. This version of the application was deployed using Firebase Hosting.

The application is also dockerized and can also be found at localhost:3000, when run with Docker.

7.3.2 Flask Server

The Flask server used for user authentication can be found here: http://kniland97.eu.pythonanywhere.com.

References

[1] Sanchit Aggarwal. "Modern Web-Development using ReactJS". In: *International Journal of Recent Research Aspects* 5.1 (Mar. 2018), pp. 133–137.

[2] ArjunAranetaCodes. *React, Flask, and MongoDB Login and Registration.* URL: https://github.com/ArjunAranetaCodes/MoreCodes-Youtube/tree/master/react-flask-mongodb-login-reg.

[3] Owen Astrachan. "Bubble sort: an archaeological algorithmic analysis". In: *ACM SIGCSE Bulletin* (Jan. 2003).

[4] *Bubble Sort Implementation.* URL: https://www.geeksforgeeks.org/bubble-sort/.

[5] *Bubble Sort Implementation.* URL: https://www.geeksforgeeks.org/insertion-sort/.

[6] *Bubble Sort Implementation.* URL: https://www.geeksforgeeks.org/heap-sort/.

[7] *Bubble Sort Wiki.* URL: https://en.wikipedia.org/wiki/Bubble_sort.

[8] *Building a Sort Algorithm with React and RxJS.* URL: https://medium.com/@fiffty/building-a-sort-algorithm-visualizer-with-react-and-rxjs-9799f91e541b.

[9] *Firebase Documentation.* URL: https://firebase.google.com/docs/?gclid=CjwKCAjwnIr1BRAWEiwA6GpwNTqfEidzPDw4DoKA1aRz-4Vl1CzYzObF9aLjgHEaZuHQYqN-LSDraRoCTIoQAvD_BwE.

[10] *Flask Documentation.* URL: https://flask.palletsprojects.com/en/1.1.x/.

[11] *Heap Sort Wiki.* URL: https://en.wikipedia.org/wiki/Heapsort.

[12] *How To Build A Bar Graph With React.* URL: https://medium.com/@ItsMeDannyZ/how-to-build-a-bar-graph-with-react-458a19ef0ba0.

[13] *How to create a sorting visualizer.* URL: https://www.reddit.com/r/learnprogramming/comments/d0vdia/how_to_create_a_sorting_visualizer/.

[14] *Insertion Sort Wiki.* URL: https://en.wikipedia.org/wiki/Insertion_sort.

[15] Anurag Kumar and Ravi Kumar Singh. "Comparative Analysis of AngularJS and ReactJS". In: *International Journal of Latest Trends in Engineering and Technology* 7.4 ().

[16] *Listing Files from Firebase Storage*. URL: https://firebase.google.com/docs/storage/web/list-files.

[17] *Merge Sort Implementation*. URL: https://www.geeksforgeeks.org/merge-sort/.

[18] *Merge Sort Wiki*. URL: https://en.wikipedia.org/wiki/Merge_sort.

[19] Clement Mihailescu. *Clement Mihailescu's Sorting Visualizer*. URL: https://clementmihailescu.github.io/Sorting-Visualizer/.

[20] *MLab Documentation*. URL: https://docs.mlab.com/.

[21] *MongoDB Documentation*. URL: https://docs.mongodb.com/.

[22] José M. Pérez. *Poor Man's ScreenFlow*. URL: https://github.com/JMPerez/screenflow.

[23] *Python Documentation*. URL: https://docs.python.org/3/.

[24] *Quick Sort Implementation*. URL: https://www.geeksforgeeks.org/quick-sort/.

[25] *Quick Sort Wiki*. URL: https://en.wikipedia.org/wiki/Quicksort.

[26] *React Documentation*. URL: https://reactjs.org/.

[27] *React Firebase File Uploader*. URL: https://www.npmjs.com/package/react-firebase-file-uploader.

[28] *React HTML Parser*. URL: https://github.com/wrakky/react-html-parser.

[29] *React Wikipedia*. URL: https://en.wikipedia.org/wiki/React_(web_framework).

[30] *ReactJS + Flask + MongoDB User Authentication*. URL: https://www.youtube.com/watch?v=jIbx9yOvJ08.

[31] Rajat S. *A Guide To Sorting Algorithms in JavaScript*. URL: https://blog.bitsrc.io/a-guide-to-sorting-algorithms-in-javascript-5b32da4eae1e.

[32] University of San Francisco. *University of San Francisco's Sorting Visualizer*. URL: https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html.

[33] *Selection Sort Implementation*. URL: https://www.geeksforgeeks.org/selection-sort/.

[34] *Selection Sort Wiki*. URL: https://en.wikipedia.org/wiki/Selection_sort.

[35] *Setting up Flask applications on PythonAnywhere*. URL: https://help.pythonanywhere.com/pages/Flask/.

[36] *Shell Sort Implementation*. URL: https://www.geeksforgeeks.org/shellsort/.

[37] *Shell Sort Wiki*. URL: https://en.wikipedia.org/wiki/Shellsort.

[38] *Sorting Algorithm*. URL: https://en.wikipedia.org/wiki/Sorting_algorithm.

[39] VisuAlgo.net. *VisuAlgo's Sorting Visualizer*. URL: https://visualgo.net/bn/sorting?slide=1.

[26] [23] [10] [9] [21] [20] [19] [39] [32] [13] [2] [31] [16] [8] [35] [28] [22] [30] [12] [27] [7] [4] [11] [6] [14] [6] [18] [17] [25] [24] [34] [33] [37] [36] [1] [15] [3] [38]