|  |  |  |
| --- | --- | --- |
| **Lab 1 Report** | | |
|  | | |
| **Calculator using NodeJS and ReactJS** | | |
|  | | |
| **Introduction** | | |
|  | | |
| **Goal:** | | |
| -The goal of the system is getting used to the basic working of ReactJS to control the frontend using the most basic part of ReactJS knows as components and NodeJS to control the backend using routes.  -Getting the overview of RESTful API services. | | |
|  | | |
| **Purpose:** | | |
| - As the name suggests, this calculator will help user solve basic mathematical operations like addition, subtraction, multiplication and division.  - The system will allow user to open and close the calculator and will also notify user of invalid inputs. | | |
|  | | |
| **Backend Routes** | | |
|  |  |  |
| **Route** | **Method** | **Functionality** |
| /solve | POST | - posts the equation to be solved and returns the evaluation to the client side. |
|  |  |  |

|  |
| --- |
| **Results:** |
|  |
| **The Start Page** |
|  |
| **Explanation:** |
| - “/” will show the navigation bar component with 2 buttons viz. Start Calculator and Stop Calculator.  - “Start Calculator” link will allow user to interact with the calculator UI to solve basic mathematical operations.  -“Stop Calculator” link will hide/stop the UI from the home page. |
|  |

|  |
| --- |
| **Calculator Started** |
|  |
| **Explanation:** |
| - This is the basic calculator UI with number buttons and buttons for clearing the input field and the answer sign denoted b “=”, which will send the request to the backend running on port 3001, to solve the operation mentioned in the input field.  - The input field is disabled for input and the user will only be able to enter the data using the buttons to avoid any haphazard situations like “having an alphabet or unwanted symbol” in the input field. |
|  |

|  |
| --- |
| **A normal equation** |
|  |
| **Explanation** |
| - The above screenshots shows the normal equation which will output the result when user will click on the “=” button.  - The output can be seen in the second screenshot. |
| **Some examples of Invalid Input** |
|  |
|  |
| **Explanation:** |
| - This calculator also handles exceptions like “operations doing a division by zero” and “not entering the full equation”.  - On seeing division by zero, the backend throws a syntax error which is caught by the program and the user is shown that the inputted value was invalid. |
|  |
|  |
|  |
| **Explanation:** |
| - Another example of invalid input is, not entering the complete equation.  - Over here, the “/” sign has only one operand to work with, which is invalid and the output satisfies the same. |
|  |

|  |  |  |
| --- | --- | --- |
| **Performance:** | | |
| 1. **Invoke 100 concurrent users with 1,000 calls each to calculator on randomly selected tasks.** 2. **Graph after invoking 5,000 calculator calls on randomly selected tasks.** 3. **Graph after invoking 1,000 calculator calls on randomly selected tasks.** | | |
|  | | |
|  | | |
| **Conclusion:** | | |
| - This was a pretty straightforward application which made use of simple ReactJS concepts and the basic NodeJS concept.  - This forms a pretty solid foundation to create complicated apps with the help of ReactJS and NodeJS like “A prototype of HomeAway.” | | |
|  | | |
|  | | |
|  | | |
|  | | |
|  | | |
|  | | |
|  | | |
|  | | |
|  | | |
|  | | |
|  | | |
|  | | |
|  | | |
|  | | |
|  | | |
| **HomeAway using NodeJS, ReactJS and MySQL** | | |
|  | | |
| **Introduction:** | | |
|  | | |
| **Goals of the system:** | | |
| - The goal of this system is to widen the understanding of ReactJS and NodeJS to develop enterprise-wide applications which takes care of various operations like managing the information of users, managing the information of owners, managing the information of properties listed by different owners and also taking care of booking done b different travelers.  - In order to develop this application, we will get a wide understanding of different concepts of ReactJS like props, states, Redirect features, form validations and many more.  - This application will also help us in widening the backend i.e NodeJS concepts because of different operations like communication with MySQL, managing session, encrypting passwords using bcrypt, uploading images and many more. | | |
|  | | |
| **Purpose of the system:** | | |
| - The purpose of the system is to connect different property listers and the travelers who will access this listed properties.  - The system will allow different owners of properties around the globe to rent their properties on the system and in turn help them to earn through it.  - For travelers, this system will allow them to search for different properties all around the world and booking the one they want to rent.  - This system will take care of booking and listing of properties with proper session management. | | |
|  | | |
| **Backend Routes** | | |
|  |  |  |
| **Routes** | **Method** | **Functionality** |
| **Travel** **Routes** | | |
| /login | POST | - Sends the parameters like email and password and the route authenticates the user. If the credentials are correct, the user is logged in. |
| /signup | POST | - Sign’s up new user and send an acknowledgement on successful creation. |
| /:travelid | GET | - Get the details of the travel user having id :travelid |
| /:travelid | PUT | - Update the details of the travel user having id :travelid |
| /:travelid/editpassword | PUT | - Edits the password of the user having id :travelid |
| /:travelid/bookingdetails | GET | - Gets the booking details that the user having :travelid has done. |
| /upload | POST | - Updates the profile pic of the traveler |
|  | | |
| **Owner** **Routes** | | |
| /login | POST | - Sends the parameters like email and password and the route authenticates the owner. If the credentials are correct, the owner is logged in. |
| /signup | POST | - Sign’s up the new owner and send an acknowledgement on successful creation. |
| /:ownerid | GET | - Get the details of the owner user having id :ownerid |
| /:ownerid | PUT | - Updates the detail of the owner having id :ownerid. |
| /:ownerid/editpassword | PUT | - Edits the password of the user having id :ownerid |
| /:ownerid/property | GET | - Gives all the properties listed by the owner having id :ownerid |
| /:ownerid/dashboard | GET | - Gives the details about who has booked the properties listed by the owner having id :ownerid |
| **Property Routes** | | |
| / | POST | - Adds a new property |
| /:propertyid | GET | - Gets the details of the property having id :propertyid |
| /:propertid | PUT | - Updates the details of the property. |
| /search | POST | - Searches for properties with filters provided in the body. |
| /:propertyid/book | POST | - Books the property having id :propertyid |
|  |  |  |
| **Results:** | | |
| We’ll consider different cases and will visit them one b one. The use cases that I’ll be discussing are as follows: | | |
| 1. **Creating a new owner account.** 2. **Logging in with the newly created owner account.** 3. **Listing a new property from this owner account.** 4. **Displaying the properties listed by the owner and adding photos to the newly created property.** 5. **Editing the owner’s detail like billing address, contact no and other things** 6. **Logging off from the owner account.** 7. **Creating a new travel account.** 8. **Logging in with the newly created travel account.** 9. **Searching a property based on “city, arrival, departure and # of guests”.** 10. **Displaying the result of the above search** 11. **Selecting a property from search result and booking it with the travel account.** 12. **Editing the basic details of travel account and also looking at the recent trips that I have booked.** 13. **Logging off.** | | |
| We’ll also consider different use cases which will have invalid inputs in form like arrival date is after the departure date and many more. Different use cases that we’ll consider for validations are as follows: | | |
| 1. **Keeping the email/password field empty** 2. **Logging in with incorrect credentials** 3. **Searching the property without location city or arrival date or departure date or guests.** 4. **Booking the property without selecting any date.** 5. **Editing the information and keeping required fields like name, contact no blank.** | | |
|  | | |
| **Use Cases’ Results:** | | |
|  | | |
| **Creating a new owner account** | | |
|  | | |
|  | | |
|  | | |
| **Explanation:** | | |
| - A new owner will be created named “Paresh Kapadia” and login credentials of [paresh@gmail.com](mailto:paresh@gmail.com) and the password encrypted with bcrypt.  - As one can see in the output of the terminal, the post request has been successfully completed and the password too has been stored after being encrypted with bcrypt and the salt cycle running for 10 rounds. | | |
|  | | |

|  |
| --- |
| **Logging in with the newly created owner account** |
|  |
| **Explanation:** |
| - The owner logged in using his login credentials.  - Notice the navigation bar, the owner has still not logged in and thus getting the option of “Login” only at the top.  - Please note the change once we login. |
|  |

|  |
| --- |
| **Listing a new property from this owner account** |
|  |
| **Explanation:** |
| - This is the home screen of the owner.  - As you can see on the navigation bar, the “login” option has changed and the owner now gets access to options like “Add new property, profile settings and many more.”  - This screen usually shows the bookings that the travelers have done on the listings that that owner has listed for the rent.  - We will be able to see the booking over here after we will create a new traveler who will book this owner’s properties. |
|  |
|  |
|  |
|  |
|  |
|  |
| **Explanation:** |
| - These screenshot shows the property that we just created in this owner’s account.  - This property will be available from 10th October,2018 to 1st January,2019.  - The property is located in “Surat” city. |
|  |

|  |
| --- |
| **Displaying the properties listed by the owner and adding photos to the newly created property** |
|  |
|  |
| **Explanation:** |
| - This page shows all the properties that this owner has listed.  - As there were no images uploaded, the next step here is to upload the images of the properties so that the travelers can know better about these properties. |
|  |
|  |
|  |
| **Explanation:** |
| - These screenshots show the steps to upload the property photos to the public folder on the server.  - These static files are located at server side in ‘public/uploads/property-{property\_id}-filename’. |
|  |

|  |
| --- |
| **Editing the owner’s detail like billing address, contact no and other things** |
|  |
|  |
|  |
|  |
| **Explanation:** |
| - These steps show how to change the details after creating an account.  - As you can see in the 2nd screenshot, the details which were already provided during the signup are pre-loaded using componentDidMount() lifecycle method. |
|  |

|  |
| --- |
| **Logging off from the owner account** |
|  |
| **Explanation:** |
| - Once the owner logs out, the owner will be directed to the login page again and the options are gone too. |
|  |
| **Creating a new travel account** |
|  |
| **Explanation:** |
| - We created a new travel account with email id = [rupal@gmail.com](mailto:rupal@gmail.com) and password=rupal which has been encrypted using bcrypt.  - The database output shows that the password is encrypted. |
|  |
| **Logging in with new travel account** |
|  |
|  |
| **Explanation:** |
| - We have logged in using the login credentials that we just used for signing up. |
|  |
| Searching a property based on “city, arrival, departure and # of guests” |
|  |
|  |
|  |
| **Explanation:** |
| - As you can see in the first screenshot, because we haven’t entered all data into the form, the search button is disabled and will not let us access the search button.  - After we provide all the details(as done in the 3rd screenshot), the search button will get enabled and we will get our results.  - The console output shows the query which is being used to search the properties, as, we have to take care of all the details provided and plus, we have to make sure that the dates provided are not booked already.  - After we click on search, we will be directed to a page which will show the search results. But, notice how the inline form has the search parameters intact even after we have directed to the next page. This was done using props. |
|  |
| **Selecting a property from search result and booking it with the travel account** |
|  |
|  |
|  |
|  |
| **Explanation:** |
| - The above screenshots show the details of the property that we just selected i.e ”The champaklal gopaldas home for old.”  - On the side, we can re-confirm our booking details and after clicking on book, we book that property for that date and for that user.  - The console output will certify that thing. |
|  |

|  |
| --- |
| **Editing the basic details of travel account and also looking at the recent trips that one has booked** |
|  |
|  |
|  |
|  |
| **Explanation:** |
| - This section shows how we updated the profile picture of the travel user.  - The profile pic is stored on the server side at the location /public/uploads/profile-{travel\_id}.jpg  - After that, one screenshot shows the recent trips that the user has booked. Our results show the property we just booked in the previous section. |
|  |

|  |
| --- |
| **Logging off** |
|  |
|  |
| **Explanation:** |
| - Demonstrates a simple functioning of logging off. |
|  |
| **Validation Use-Cases’ Results:** |
|  |
| **Keeping the email/password field empty** |
|  |
|  |
| Logging in with incorrect credentials |
|  |

|  |
| --- |
| **Searching the property without location city or arrival date or departure date or guests** |
|  |
| **Explanation**: |
| - First screenshot shows that the arrival date is before today’s date and will show an error to the user saying “available\_from is invalid”.  - Second screenshot shows that the guests cannot be negative. We selected -1 as our guests which is never possible and thus it gets reflected.  - In both the cases, please notice the search button. It is disabled throughout. It will only get enabled if all the details are passing the validations. |
|  |
| **Booking the property without selecting any date** |
|  |
| **Explanation:** |
| - Over here, the user just searched about the property from the recent trips. Because it was directed from the recent trips, the user don’t want to book the property.  - Thus, because we won’t have any dates to play with, the “book now” button will be disabled. For the user to book it, the user might have to go to home page and search for it along with the details. |
|  |

|  |
| --- |
| **Editing the information and keeping required fields like name blank and contact no not of 10 digits** |
|  |
| **Explanation:** |
| - As it shows, one cannot have empty name in the database and one cannot have more than 10 digits in the contact no.  - Because these validations were proved wrong, the “Save Changes” button is disabled and the user is shown the error above the form. |
|  |
|  |

|  |
| --- |
| **Recent Bookings Done by Travel User** |
| **Shows the bookings done by the travel user** |
|  |
| **Owner properties booked by travel users** |
| **Shows which properties of the owner’s was booked by whom** |
| Question and Answers: |
|  |
| **Q: Explain the encryption algorithm used in your application. Mention different encryption algorithms available and the reason for your selection of the algorithm used?** |
| Ans:  1) **SHA256:** A cryptographic hash (sometimes called ‘digest’) is a kind of ‘signature’ for a text or a data file. SHA-256 generates an almost-unique 256-bit (32-byte) signature for a text.  2) **SHA512:** SHA512 is a cryptographic hash function that when applied to the provided input results in a 128-digit hexadecimal number that is highly unlikely to match the value produced for a different input. SHA512 is a variant of the SHA-2 set of cryptographic hash functions.  3)**MD5:** The MD5 message-digest algorithm is a widely used hash function producing a 128-bit hash value. Although MD5 was initially designed to be used as a cryptographic hash function, it has been found to suffer from extensive vulnerabilities.  4) **Bcrypt:** bcrypt is a password hashing function designed by Niels Provos and David Mazières, based on the Blowfish cipher, and presented at USENIX in 1999.[1] Besides incorporating a salt to protect against rainbow table attacks, bcrypt is an adaptive function: over time, the iteration count can be increased to make it slower, so it remains resistant to brute-force search attacks even with increasing computation power.  The algorithm that **I used was bcrypt** and the reasons are:  - General-purpose cryptographic hash functions like the (now-broken) MD5, SHA512, SHA256, etc. are designed to be **computationally easy, ie. fast**.  - Salting protects against rainbow tables [1], but it doesn't change the fact that computing a SHA256 hash is fast.  -Password hash functions like bcrypt, scrypt are designed to be computationally expensive, to make a **password-cracking endeavor take even longer.** |
|  |
| **Q: What is SQL caching? What all types of SQL caching are available, and which suits your code the most. You don’t need to implement the caching, write pseudocode or explain in detail.** |
| Ans:  **SQL Caching:**  Caching in general is storing the results of a process that is resource intensive so that the resources won’t have to be spent again.  In SQL; it is always specific to the vendor technology. MS-SQL, Oracle, MySQL and PostGres all have different choices for what they cache. The results, the query, even the execution plan for the query can all be stored for use by later attempts to process the same information.  In the event that the underlying data is changed the system usually has a mechanism to indicate that the cached information is “dirty” and no longer fully usable; so the query can be run again by the system.  **Two Techniques are used:**   1. Upfront population 2. Lazy population   Upfront population means that you populate the cache with all needed values when the system keeping the cache is starting up. Being able to do so requires that you know what data to populate the cache with. You may not always know what data should be inserted into the cache at system startup time.  Lazy evaluation means that you populate the cache the first time a certain piece of data is needed. First you check the cache to see if the data is already there. If not, you read the data from the remote system and insert into the cache. |
| **Q: Is your session strategy horizontally scalable? If YES, explain your session handling strategy. If NO, then explain how you can achieve it.** |
| Ans:  No, my session strategy is not horizontally scalable. I have used cookies to handle my sessions, which is not a way to horizontally scale the sessions.  Steps to achieve horizontally scalable session:  The first is to make your application stateless on the server side as much as possible. Any time your application has to rely on server-side tracking of what it’s doing at a given moment, that user session is tied inextricably to that particular server. If, on the other hand, all session-related specifics are stored browser-side, that session can be passed seamlessly across literally hundreds of servers. The ability to hand a single session (or thousands or millions of single sessions) across servers interchangeably is the very epitome of horizontal scaling.  The second goal to keep square in your sights is to develop your app with a service-oriented architecture. The more your app is comprised of self-contained but interacting logical blocks, the more you’ll be able to scale each of those blocks independently as your use load demands. Be sure to develop your app with independent web, application, caching and database tiers. This is critical for realizing cost savings – because without this microservice architecture, you’re going to have to scale up each component of your app to the demand levels of the services tier getting hit the hardest. |
|  |
|  |
| **Commit History** |
|  |
|  |
|  |
|  |
|  |
| **Github Link: https://github.com/Anuragis/CMPE273-36.git** |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |