Granify - Assignment Part 2

1. **Why did you use the design/framework (if any) you chose?**
    - **React -** provided a slick workflow for displaying my view components
    - **CreateJS -** event and event dispatcher classes used by the model class to dispatch events and will notify when changes occur
    - **Bootstrap-** helps with the style and responsiveness for the page
    - **CSS3 -** was also used to style the site further
    - **Reactstrap -** easy to use react components set - reactstrap.github.io
    - **Jquery + Ajax -** used for loading external data which is used to build layout and site content
    - The decision to use these tools for this exercise is a long one but to keep it short and simple, It was really easy to get the initial layout working and allowed me to focus on other areas that might be more time consuming. Also it was a “self imposed” challenge to uses these tools.
2. **Are there better frameworks for this task which you did not choose for some reason (e.g. example you are not familiar with them and did not have time to learn them)?**
    - This question is a bit difficult to answer. In a short form I would respond with “it depends”. The degree in which frameworks are judge should be subjective to each project. In this particular case I felt that going with React would serve my purpose just fine. I am sure that other developers experienced in other frameworks like angular would have gone with that technology over React.
    - Personally I feel a developer should weigh several factors when taking on a project. Knowing the tech is a big part of it but in the project he or she might need to investigate unfamiliar tech.
    - In short, I am sure there might be ‘better’ frameworks, that all depends on who is judging and what qualifies as better.
3. **When you added testing, why did you select those particular tests?**
    - I installed **Enzyme** for unit testing because it’s billed as a JS testing utility for React components and is quite popular. Unfortunately time was a factor and couldn’t build a proper test harness in that time.
    - There is one unit test and can be tested by running : `npm test`
    - However I used other forms of testing this application:

- **Functionality Testing:**
  - Testing loading site data (pulling for external url)
  - Testing all internal links (nav buttons, form buttons etc)
  - Testing for wrong input (adding records)
  - Check validating data (adding records)
    - Adding record form filled out completely
    - Phone number can only contain numeric values
  - Testing for deleting multiple records
  - Testing selection of records
  - Testing newly entered records have a unique 10 digit id
  - Testing form buttons are enabled/disabled when needed
  - Tested all calculation are correct:
    - Average records added per hour
    - Average records deleted per hour
    - Ratio of records added per hour versus deleted per hour.
    - Total records added
    - Total records deleted
    - Total time on page

Please Note: Average is calculated assuming that 0 - 60 mins falls within the first hour, meaning that if one add 15 files within 0 - 60 mins the average will be 15 files per hours, if the hour goes past 61 mins the average will fall to 7.5 files per hours, since 15 files / 2 hours = 7.5 AVG per hour.

Testing activity ratio can be done by setting a Debug flag within constructor function of Stats.js

- **Usability Testing**
  - Navigation is easy to understand (menu items are clearly marked)
  - Forms are straight forward
  - Menu links work as expected
  - Design consistent throughout site
- **Compatibility Testing**
  - Site tested on different browsers
  - Simulate mobile browser via chrome developer tools

1. **How can the API calls be made secure so that not anyone with the API can call them?**

    - OAuth can be implemented to secure the API calls. For this to work you'd need to store a "secret key" on the server that the client runs on. This server can make server to server calls using that key to obtain a temporary session token. This token will expire preventing people from using the api.
2. **Explain how bad input should be handled on the back end if it gets past the front end checks.**
    - The server should always valid input from the client side application. If the input isn't valid the server should return an error and surfaced to the user. Client should only display the error results
    - Client should have some basic validation that could filter bad data before request is made to server.
3. **Which JavaScript best practices do you find valuable and adhere to?**
    - Using === instead of ==
        - A sure way of comparing both value and types
    - Never shorthand function braces
    - Avoid "digging-deep" into object properties
        - Ex: foo.bar.foo1.doSomething();
    - Declare variable outside of loops
    - Comment code
    - when creating object, arrays or String by using {} and avoiding `new` keyword
        - Ex: let arr = [] vs let arr = new Array()
    - Use 'Strict Mode' when using JS, helps prevent certain actions from being taken and throws more exceptions. With strict mode, you can not, for example, use undeclared variables. By default Strict mode checks are run in development mode only
    - **PropTypes is** a good practice to have on React projects
    - Using clear and descriptive names for methods and properties within a class
    - Constructing functions do only one thing and not multiple tasks
    - Functions should be made somewhat generic so that they might be used by multiple sources
    - Loading script with <body> tag to ensure they are ready