Design Details

And

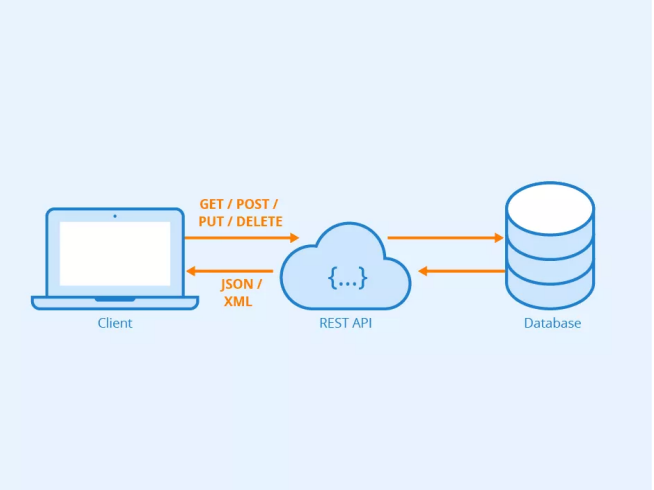
Management Information Report

Design Details

Given the data source http://outbreaks.globalincidentmap.com/ we plan to develop a Restful API aimed to fetch information about the diseases in each country including number of people infected, death rate, and more to update relevant information every 24 hours.

Steps That Will Be Taken to Develop The API Module:

1) We first intend to collect data from "http://outbreaks.globalincidentmap.com/" by using scrapy and selenium.
2) After collecting the base data, we will account for newly added data by creating scrapers to detect new entries on the website.
3) The data collected will be stored in the form of JSON using mongodb, specifically using Atlas cloud storage.
4) We will then create a Restful API using NodeJS and ExpressJS, that will integrate mongoose to retrieve data from the MongoDB library.
5) We then intend to launch both the scraper and Restful to AWS in order to automate these processes, whereby the scrapers will run once a day to keep the data up to date.
6) Furthermore, the Restful API will be available for public use, and will be documented using Stop Light.



Why Python:

Although we have chosen Javascript/NodeJS as our backend, Python provides a rich set of libraries for scraping , cleaning and preparing data to be stored in our database of choice (MongoDB Atlas).

Scrapers with python:

1. Scrapy:
    a. Is extremely efficient and fast.
    b. However, it has a steep learning curve.
    c. Hence, should be used to scrape the base website that is http://outbreaks.globalincidentmap.com/
2. Beautiful Soup 4:
    a. Is slower than Scrapy.
    b. However it is simple and easy to use and allows for query generalisation for different web-articles.

    c. Hence suitable to scrape external links in the “outbreaks” site, it will not require case by case analysis of a web-article’s Html source code.
3. Selenium:
    a. Is extremely slow compared to the previous scrapers.
    b. The value lies in its ability to interact with Javascript, i.e. to navigate sites.
    c. Hence, by replacing the drivers of Scrapy and BS4 with Selenium’s, we are able to both scrape websites efficiently and collect previously inaccessible information.

Scrapers in detail:

Spiders:

1. Spiders are scrapy scrapers that could be run concurrently.
2. We will have two Scrapy Spiders:
    a. OutBreaksSpider.py will gather:
        i. Disease,
        ii. Time/Date,
        iii. Country,
        iv. City,
        v. Description,
        vi. Internal links on the site.
    b. OutBreaksSpiderDetails.py: Using the internal links collected in the previous spider, it gathers a disease case’s
        i. Latitude,
        ii. Longitude,
        iii. external links to web articles.

BS4 or Newspaper3k:

- Now that article links have been collected, we need to access these links and scrape the data without creating case by case queries.
- We will utilize python modules such as Newspaper3k, which can identify informational parts of news articles and collect the data. We then add this new found information to the JSON data. Hence, utilizing the right tools for the tasks.

Deployment and Updating Database:

- Scraper will be deployed to AWS. A check will be run daily to check the “Last-Modified” parameter in the header of the “outbreak” site. If the parameter has changed, the website will be scrapped for new diseases by looking at each category of diseases and retrieving ones with newer dates.

Database:

- The JSONs collected will be stored using MongoDB. We have selected MongoDB Atlas to store and retrieve data. Atlas integrates well with AWS, and hence is suitable

to write and retrieve data in a well tested and documented environment where we do not have to reinvent the wheel. Furthermore, MongoDB has NodeJS libraries and Python libraries which will simplify the process of storing and querying data.

Amazon Web Service:

- We have chosen AWS to deploy our API since it is:
    - Comprehensive:
        - The interface is user friendly, and easy to use.
        - It provides documentation, and videos for people to use.
    - Adaptable:
        - AWS has assets to assist in enhancing its IT framework.
        - Enables users to spin-up clones in multiple regions for different environments within a few minutes, eliminating the need to repeat the set-up steps every time.
    - Secure:
        - Prevents security leaks.
        - In the face of most failure modes, including natural disasters or system failures multiple geographic regions and Availability Zones allow you to remain resilient.
        - Ability to configure built-in firewall rules from totally public to completely private or somewhere in between to control access to instances.

Summary of overall API architecture model:

- A collection of python scrapers launched to AWS will feed and update cloud storage of MongoDB Atlas. Express and NodeJs will provide a backend to access the cloud storage through endpoints and request data.

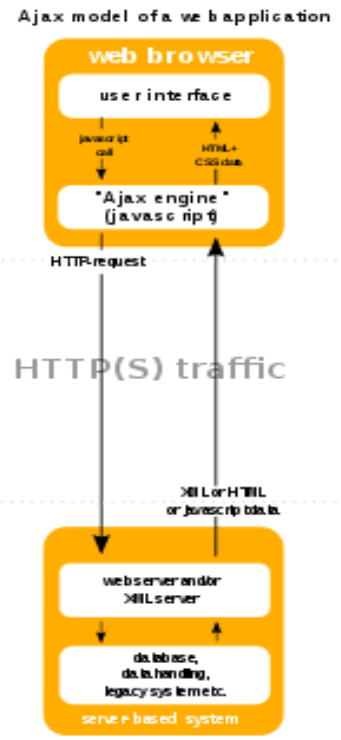
Reasons behind choosing Stoplight for documentation:

The most valuable advantage of Stoplight is it being free of charge compared to Swagger, and thus has allowed us to collaborate and create beautiful documentation. Furthermore:

1. It allows for collaboration, and hence division of tasks.
2. Integrates nicely with github, where the advantages of version control can be seen.
3. Provides both Code based and Form based documentation creation, whereby we did not have to learn OpenAPI to start writing documentation and hence saved time.
4. The documentation can be uploaded and shared to allow external users to familiarise themselves with the API.
5. Reusability, objects such as parameters can be reused in many different parts easily.

Interactions with API modules using sample GET requests:

- The three main areas of input for the API are:
    - Time period of interest (start date and time, end date and time)
    - Key search terms (disease name or multiple disease names)
    - Location (including city, country, coordinates).



The API design uses user-friendly endpoint names, and has relevant status codes and error messages in accordance with the OpenAPI design guidelines.

The api will make GET requests to the scraped data (will be scraped every 24 hours) that will be stored and updated in a JSON format on MongoDB Atlas. Each GET request made to the data will collect corresponding results about disease reports based on the input. The front-end will be using jQuery methods (possibly with the Cheerio Library), that the API will then use in the corresponding request as an input parameter.

Example Parameters:

1) Time: GET {{api_link}}/dates/

- Parameters for the url:

Name (Key)	Type	Description
Date	Datetime	yyyy-MM-dd format. Date is the only input parameter.
Datetime	Datetime	yyyy-MM-ddTHH:mm:ss format. Date and time are input parameters.

For example if you'd like to get disease reports sorted by dates, GET {{api_link}}/dates?date={{user_input}} will give you the result.

- Sample JSON response:

```
{
  "date": {{date}},
  "Diseases": [
    {
      "r1": "report 1"
    },
    {
      "r2": "report 2"
    }
  ]
}
```

```
{
  "time_period" : {{start_date}} - {{end_date}}
  "Diseases": [
    {
      "r1": "report 1"
    },
    {
      "r2": "report 2"
    }
  ]
}
```

2) Diseases: GET {{api_link}}/diseases/

- Parameters for the url:

Name	Type	Description
Names	String	Comma-separated string of diseases to be searched.

For example if you'd like to get disease reports with multiple disease names or a single name in the search, GET {{api_link}}/diseases?names={{user_input}} will give you the result.

- Sample JSON response:

```
{
  "key_terms" : {{tags}},
  "Diseases": [
    {
      "r1": "report 1"
    },
    {
      "r2": "report 2"
    }
  ]
}
```

Here the “tag” can be anything related to the disease, like name, news articles etc.

3) Location: GET {{api_link}}/locale/

- Parameters for the url:

Name	Type	Description
Country	String	Name of a country to be searched.
City	String	Name of a city to be searched.
Coordinates	Float	Name of a location (and surrounding radius) to be searched.

For example if you’d like to get disease reports within a country and city,

GET {{api_link}}/locale?country={{user_input}}&city={{user_input}} will give you the result.

- Sample JSON responses:

```
{
  "city": {city},
  "Diseases": [
    {
      "r1": "report 1"
    },
    {
      "r2": "report 2"
    }
  ]
}
```

```
{
  "country": {{country}},
  "Diseases": [
    {
      "r1": "report 1"
    },
    {
      "r2": "report 2"
    }
  ]
}
```

4) Articles: GET {{api_link}}/articles/

- Parameters for the url:

Name	Type	Description
Keyword/s	String	Comma-separated string of keywords to be searched to find articles.

For example if you'd like to get disease reports with multiple disease names or a single name in the search, GET {{api_link}}/diseases?names={{user_input}} will give you the result.

- Sample JSON response:

```
{
  "latitude" : {{latitude}},
  "longitude" : {{longitude}},
  "range" : {{radius}}
  "Diseases" : [
  {
    "r1" : "report 1"
  },
  {
    "r2" : "report 2"
  }
  ]
}
```

```
{
  "key_terms" : {{tags}},
  "Diseases": [
    {
      "r1": "report 1"
    },
    {
      "r2": "report 2"
    }
  ]
}
```

The tag can be any keyword related to the article to be found, like name, deaths, disease, symptoms, cases, treatments, etc.

- In line with microservice architecture, we split our locale endpoint to search by country, city and coordinates separately (it was initially one endpoint which handled all the inputs). This promotes loose coupling of the endpoints which allows for more precise return results. We applied this approach to designing all our endpoints.

The API documentation and more details on GET requests can be found here on Stoplight:

https://crackedpepper.stoplight.io/docs/seng3011-cracked-pepper/openapi.yml

- The python web scraper we built has collected all the necessary data from the website and this data is stored in a JSON format as shown in the image below:


```
[
  {
    "Disease": "Salmonella Outbreak (Suspected or Confirmed)",
    "DataArrays": [
      [
        "2020-10-26 04:18:00",
        "eventdetail.php?ID=37085",
        "India",
        "New Delhi",
        "INDIA - Indias Covid-19 Tally Reaches 7909959 - Death Toll 119014"
      ],
      [
        "2017-07-24 16:49:00",
        "eventdetail.php?ID=27396",
        "United States",
        "Atlanta, GA",
        "UNITED STATES :: Papaya-linked Salmonella outbreak sickens 47 in 12 states"
      ],
```

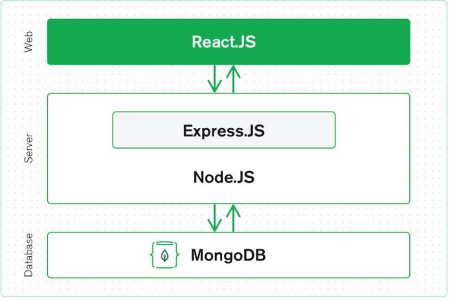

Snippet of the data we scraped.

Languages

MERN:

MERN is a full-stack, following the traditional 3-tier architectural pattern, including the front-end display tier (React.js), application tier (Express.js and Node.js), and database tier (MongoDB)

Our reasons for picking a 3-tier software architecture:

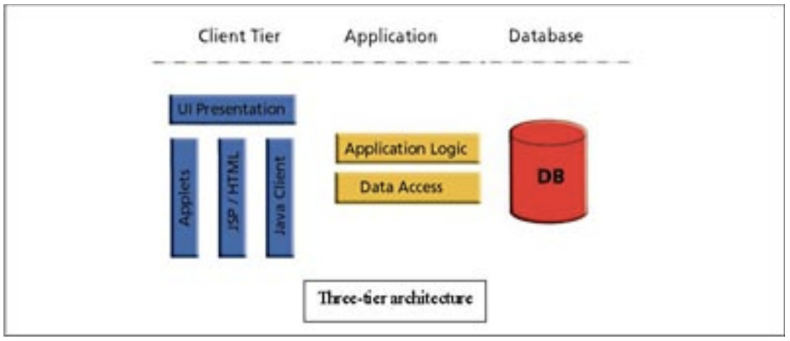


- We have the ability to update the technology stack of one tier, without impacting other areas of the whole app meaning that it will be very adaptable and scalable.
- Allows for specialization in developer teams.
  Developers can pick tiers to work on which means that there are less dependencies between the layers.
- The application has the flexibility to be scaled up or down depending on development goals since layers are independent of each other.
- Instead of directly accessing the data layer, the presentation layer only connects with the business logic layer, increasing the overall app security.

Our reasons for picking MERN:

- React.js is based on the single-page application (SPA) principle which involves having a web application accessed from a single web page. This avoids loading a new page with each action, making for a faster and more fluid user experience.

- Uses the REST API, which acts as a 'middleware' and is reusable for any application
  - The REST API allows connection between applications and



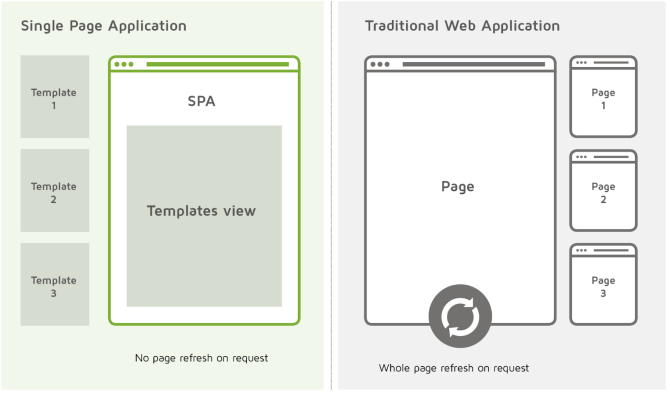
are based on HTTP and imitate web communication styles

- The use of Node.js means that the website performance will be faster compared to websites constructed in other popular environments in today's market. This is due to Node.js being an asynchronous language meaning that actions can be executed on the client side without blocking other operations.

Frontend:

React:

React is an open-source, front end, JavaScript library for building user interfaces.



Learnability:

- Built on top of vanilla JS meaning that less time is needed by developers to acquaint themselves with new syntax.
- Official React documentation is very comprehensive and walks through all the features and concepts of the framework at an understandable pace.

Reusability:

- Has the ability to reuse system components so in the event of updates, changes in one component won't affect others meaning that re-employing the same digital object makes for smoother workflow
- Uses a component-based structure that enables you to start with small components and progress to large components meaning that the development process progresses from one level of wrapper component to a higher level until you achieve one root component.

Developer tools:

- Offers a great toolkit for developers. React Developer Tools is great for inspecting react components within their hierarchy and also great for observing their current props and states.
- Redux developer tools allow developers to observe dispatched actions, current store states and watch changes on stores. It is also possible to dispatch actions or modify stores and see changes reflected to a view instantly.

Virtual DOM:

- With many fast client platforms and JavaScript engines, extensive DOM manipulation can be a performance bottle-neck and result in a subpar user experience.
- React solves this by using a virtual representation of the DOM. Any new view changes are first performed on the virtual DOM, which lives in memory and not on your screen. This guarantees a minimum update time to the real DOM, providing higher performance and a cleaner user experience all around.

Code Stability:

- 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.

Bootstrap4:

- Bootstrap is a CSS and HTML based frontend framework that is used for creating dynamic websites and web applications.

Reusability:

- Bootstrap is a convenient framework to use since it has pre-built blocks of code therefore reusability is more possible and code complexity is reduced.

Responsiveness:

- It's gridding system is responsive therefore the display can fit in any screen resolution. Its 'mobile-first' principle is important for scalability and works greatly with React native.

Consistency:

- Since it uses pre-built code, Bootstrap4 can be worked on by different developers and still be mutually understandable owing to the consistency of its functions.
- Eradicates the use of libraries that differ from developer to developer.

Customization:

- Has a large library of functions which will give us a lot of elements to work with for our frontend design.

Learnability:

- Furthermore, the documentation for Bootstrap is very comprehensive and allows us to grasp key concepts and utilities in a limited timeframe.

Backend:

Node.js:

- Node.js is a server-side JavaScript run-time environment.

Learnability:

- Can be used with a rudimentary knowledge of javascript and backend programming principles

Scalability:

- Easy to scale the applications in horizontal as well as the vertical directions. The applications can be scaled in horizontal manner by the addition of additional nodes to the existing system.
- Offers the option of adding extra resources to the single nodes during the vertical scaling of the application.

Performance:

- Interprets the JavaScript code via Google's V8 JavaScript engine. This engine compiles the JavaScript code directly into the machine code making it easier and faster to implement the code.
- The speed of the code execution is also enhanced by the runtime environment as it supports the non-blocking I/O operations.

Effective Caching:

- Node's runtime environment provides the facility of caching single modules. Whenever there is any request for the first module, it gets cached in the application memory.

Concurrent request handling:

- Process several requests concurrently since Node provides non-blocking I/O systems

- Since the incoming requests get lined up and are executed quickly and systematically, it can perform concurrent request handling better than Ruby or Python.

Extensibility:

- Allows the use of JSON to exchange data between the web server and the client.
- Includes built-in APIs for developing HTTP, TCP, and DNS servers.

MongoDB:

MongoDB is a document database built on a scale-out architecture.

Learnability :

- Stores data in collections with no enforced schema. This flexible approach to storing data makes it suitable for beginner-level backend developers who want to use a database to support the development of their applications.
- Not a relational database (like SQL databases) so there is no need to understand the principles of normalization, referential integrity, and relational database design.

Scalability:

- Extremely easy to scale. Configuring a sharded cluster (data distributed across many servers) allows a portion of the database, called a shard, to also be configured as a replica set. A replica set is the replication of a group of MongoDB servers that hold the same data, ensuring high availability and disaster recovery.
- This highly flexible approach allows MongoDB to horizontally scale both read and write performance to cater to applications of any scale.
- SQL databases have vertical scalability, or adding read replicas. Vertical scalability involves adding more resources to the existing database server but has a certain limit. Read replication involves adding read-only copies of the database to other servers. This can cause issues with applications that are either write-heavy, or write and read regularly for the database

Performance:

- MongoDB documents tend to follow a hierarchical data model and keep most of the data in one document, therefore eliminating the need for joins across multiple documents.
- Optimized for write performance, and features a specific API for rapidly inserting data, prioritizing speed over transaction safety wherein SQL data needs to be inserted row by row.

Flexibility:

- The schemaless design of MongoDB documents makes it extremely easy to build and enhance applications over time, without needing to run complex and expensive schema migration processes as you would with a relational database.
- In MongoDB, there are more dynamic options for updating the schema of a collection, such as creating new fields based on an aggregation pipeline or updating nested array fields. In contrast, large SQL databases are slower to migrate schemas and stored procedures that can be dependent on the updated schemas

Express JS:

- Express.js is a framework made for running Node.js and is used to organize web applications on the server-side into a more organized MVC architecture.

Usability/Learnability:

- Express's minimalism provides many benefits to developers such as simplifying routing for requests made by clients and middleware that is responsible for making decisions to give the correct responses for the requests made by the client.
- Since Express is built on top of JS, there is significantly reduced learning involved and a lot of scalability since web apps and mobile apps can be developed as a result.

Fast server-side development:

- Provides many commonly used features of Node.js in the form of functions that can be readily used anywhere in the program. This removes the need to code for several hours.

Templating:

- Provides templating engines that allow the developer to build dynamic content on the web pages by building HTML templates on the server-side.

References

Abhinav-Asthana (2018). *How Postman Can Improve Your API Documentation*. [online] ProgrammableWeb. Available at: https://www.programmableweb.com/news/how-postman-can-improve-your-api-documentation/sponsored-content/2018/07/09.

Anon, (n.d.). *The Benefits of REST API – Forte Blog*. [online] Available at: https://blog.forte.net/benefits-rest-api/.

Besant Technologies. (2020). *What is Express.js? | Why should use Express.js? | Features of Express.js*. [online] Available at: https://www.besanttechnologies.com/what-is-expressjs#:~:text=It%20is%20used%20for%20designing [Accessed 5 Mar. 2021].

Chakray. (2017). *What are the advantages of a REST API? - Chakray*. [online] Available at: https://www.chakray.com/advantages-of-rest-api/.

Default Admin User (2017). *Top 10 Advantages of Using React.js | DA-14*. [online] Da-14.com. Available at: https://da-14.com/blog/its-high-time-reactjs-ten-reasons-give-it-try.

Dhrumil Bhatt (2019). *Benefits of ReactJS: Top 10 Reasons to Choose It | CustomerThink*. [online] Customerthink.com. Available at: https://customerthink.com/benefits-of-reactjs-top-10-reasons-to-choose-it/.

MindInventory. (2019). *The Advantages and Disadvantages of Node.js Web App Development*. [online] Available at: https://www.mindinventory.com/blog/pros-and-cons-of-node-js-web-app-development/.

MongoDB. (2019). *MongoDB and MySQL Compared*. [online] Available at: https://www.mongodb.com/compare/mongodb-mysql.

Naeem, T. (2020). *REST API: Definition, Working, Benefits, and Design Principles*. [online] Astera. Available at: https://www.astera.com/type/blog/rest-api-definition/.

Office Timeline. (n.d.). *Free Timeline Templates for Professionals*. [online] Available at: https://www.officetimeline.com/timeline-template.

Tech Blogs by TechAffinity. (2020). *Why Should you Use Bootstrap? | Responsive Front-end Development*. [online] Available at: https://techaffinity.com/blog/why-use-bootstrap-for-frontend-design/ [Accessed 5 Mar. 2021].

Willoughby, J. (2017). *The Top 5 Benefits of React that Make Life Better*. [online] Telerik Blogs. Available at: https://www.telerik.com/blogs/5-benefits-of-reactjs-to-brighten-a-cloudy-day.