# Data Visualization Generator

SRS Document

Doofenshmirtz Evil Inc

COS 301 - 2020

Marco Lombaard u18026975
Elna Pistorius u18010319
Phillip Schulze u18171185
Byron Tomkinson u18042717
Gian Uys u18052569

# Contents

# 1 Introduction

Huge amounts of structured and unstructured data is being stored and processed at a very high rate. This is where the term **'Big data'** comes from. Data is captured to help detect problems and to make better decisions. It is much easier for us as humans to gain insight from data patterns if it is visually represented using charts.

These representations take a lot of time to create manually for each data set, especially when we want to create powerful tools like dashboards and drill-downs for end-users.

## 1.1 Definitions

- **Big data:** It is a field that focuses on extracting information and ways to analyze data sets that are very large or complex to deal with. [1]
- **Charts:** It is a large group of methods for presenting information in the form of graphs, diagrams, or tables.
- **IGA:** (Interactive Genetic Algorithm), is an effective method in solving optimization problems with implicit or fuzzy indices. These types of algorithms combine evolutionary mechanisms with the user's intelligent evaluation and individual fitness. [2]
- **Drill Downs:** This provides the user with the capability to view a more specialized representation of the data. [3]
- **Dashboards:** This is an information management tool used to visually track, analyze, and display key performance indicators (KPI). Dashboards can be customized to meet the specific requirements of the end-user. [4]

## 1.2 Vision

**Data Visualization Generator** is a progressive web application used to simplify and enhance the way **'Big data'** is displayed on charts. It takes time to make visualizations manually, instead of building these visualizations from scratch the Data Visualization Generator system will make it easier and faster for the end-user, by providing an intelligent list of auto-generated visualizations as suggestions. These suggestions would be based on data as is or after it is preprocessed through other Artificial Intelligence (AI) algorithms.

An **Interactive Genetic Algorithm (IGA)** will be used to suggest visualizations for dashboards and to provide drill-down of these suggestions. These visualizations will help the end-user to make adequate decisions and to make more accurate predictions.

## 1.3 Objectives

The key objectives that need to be fulfilled so that the Data Visualization Generator system can be used effectively as a tool to visualize data are the following:

- Enable the user to **select a specific data source and trigger visualization generation**. This would be used to provide the user of a visual suggestion of the data. An IGA is used for suggestion generation, the user can then trigger a next-generation calculation if a better suggestion is desired.
- Enable the user to **view and add these suggestions easily to dashboards**. The user can then filter all the visualizations on the dashboard, making it easier to track, analyze and display key performance indicators.
- The Data Visualization Generator system aims to mitigate the time and effort needed for manually creating visual representations of data provided.

## 1.4 Business Needs

Data Visualization Generator enables companies, governments, and health care providers to generate customized reports and interact with data in an unmediated way.

Data Visualization Generator helps these users increase their **productivity** by presenting a visual summary of data, making it easier to **identify patterns and trends**. This could help users spend less time analyzing data but providing a platform where these users can make **informed decisions** based on the visual representations provided. Users can then customize these representations of data to eliminate what they do not require and drill down into more important details.

## 1.5 Scope

Data Visualization Generator will be a progressive web application, paired with a backend system, this will be accessed through the web interface. Controlled altering and viewing of the data stored on the backend database will be allowed through the backend system and web application.

The database will be set up using a database management system like **PostgreSQL**, which is an appropriate relational database management system **(RDBMS)**.To ensure access to this database is secure, a **web API** will be used, this will be implemented using a server-side language such as **NodeJS**.

The web application will be developed using a framework such as **React**. End-users will be able to select specific data sources and trigger visualization suggestion generations via the web application. This would send a request to an endpoint on the server that would trigger the IGA to suggest representations. The user must then be able to view and add these suggestions easily to a dashboard.

The data sources would be stored in the database on the server and each data entity required from the database would be retrieved via an API. Only administrators would have access to the backend system to be able to alter the database and do any server-side actions.

## 1.6 Domain Model

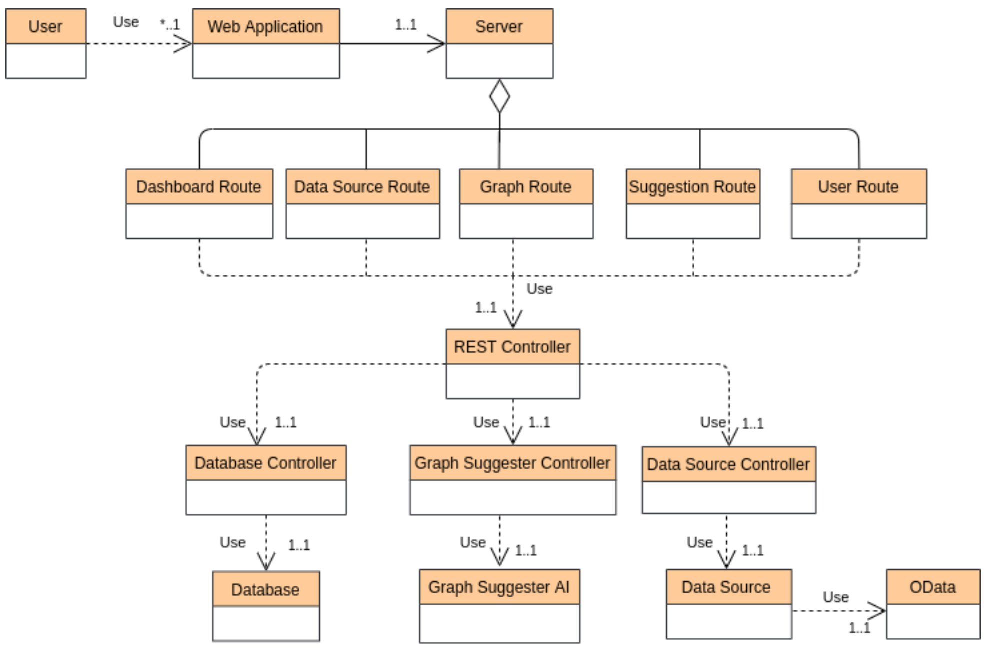


***Figure 1:*** Domain model showing the system components and their relationship with one another

This domain model helps to document the architecture of the Data Visualization Generator system and enhances the understanding of the system as a whole.

- The **User** will use the **Web Application** for editing, saving, or creating dashboards (please refer to all use cases to see what actions a user is able to carry out).
- The **Web Application** has a server and hence makes any necessary HTTP requests to the Server.

- The **Server** serves as a root that is composed out of a **Dashboard Route**, **Data Source Route**, **Chart Route**, **Suggestion Route** and **User Route**. These Routes forward the supported requests (and any information encoded in request URLs) to the **REST Controller**.
- The **REST Controller** handles any requests coming from the routes. This entails delegating parts of the request to the **Data Source Controller**, the **Chart Suggester Controller**, or the **Database Controller**.
- The **Database Controller** will handle requests that have to do with the database. These types of requests are any type of CRUD operations that need to be performed to the database itself. The **Database Controller** uses the **Database** to accomplish these CRUD operations.
- The **Chart Suggester Controller** will handle any requests that have to do with the IGA. These types of requests will be when the IGA is triggered for example when a user requests chart suggestions. The **Chart Suggester Controller** uses the **Chart Suggester AI** to make suggestions.
- The **Data Source Controller** will handle requests that have to do with requesting OData from external sources.
- The **Database** is a PostgreSQL database that will perform accurate CRUD operations.
- The **Chart Suggester AI** is a Genetic Algorithm that generates chart suggestions based on data that it receives from the **REST Controller**, which is requested from the **Data Source Controller**.
- The **Data Source** is an external data source that supplies clustered **OData** that is used to make chart suggestion.

## 1.7 User Characteristics

These users will use the Data Visualization Generator system by interacting with the web application. These users will mainly use the system to use the suggested visualizations generated by the IGA.

1. **Guest User**
   These type of users will use the system without having to register or log in to the system, these users will not have access to the full functionality of the Data-Visualization system.
2. **Registered User**
   These type of users will use the system by logging in, these users will have access to the full functionality of the Data-Visualization system.

## 1.8 User Stories

A user story is accompanied by lots of other documentation – these user stories are just a visual representation used for planning and to help setting up the product backlog.

### 1.8.1 Guest User

1. **As a** guest, I want to be able to select a data source **so that** I can view the entities that belong to that data source and select the entities that I am interested in.
2. **As a** guest, I want to be able to add a connection by adding a data source URL of my preference **so that** I can view the entities that belong to that data source and select the entities that I am interested in.
3. **As a** guest, I want to be able to filter by chart type (select a chart type) **so that** I can see more chart suggestions that are of that type.
4. **As a** guest, I want to be able to filter by fields (select a charts fields) **so that** I can see more chart suggestion that include those fields.
5. **As a** guest, I want to be able to view chart suggestions **so that** I can see what the data represents.
6. **As a** guest, I want to be able to drill in to suggestions **so that** I can view fitter suggestions.
7. **As a** guest, I want to be able to export charts to csv or json format **so that** I can view the charts data in excel or a similar application as excel.
8. **As a** guest, I want to be able to create charts **so that** I can view charts of my own creation.

### 1.8.2 Registered User

9. **As a** registered user I want to be able to log in **so that** I can retrieved all my dashboards and charts (for example saved dashboards, chart suggestions etc.)
10. **As a** registered user, I want to be able to select a data source **so that** I can view the entities that belong to that data source and select the entities that I am interested in.
11. **As a** registered user, I want to be able to add a connection by adding a data source URL of my preference **so that** I can view the entities that belong to that data source and select the entities that I am interested in..
12. **As a** registered user, I want to be able to create multiple dashboards **so that** I can add chart suggestions to that dashboard.
13. **As a** registered user, I want to be able to view multiple saved dashboards **so that** I can save my work and retrieve it at a later stage.
14. **As a** registered user, I want to be able to personalize a dashboard for example changing the dashboards name, description and colour **so that** I can organize my dashboards to my preference.
15. **As a** registered user, I want to be able to filter a dashboard by searching for a chart's name **so that** I can find charts easier on a dashboard.
16. **As a** registered user, I want to be able to filter by chart type (select a chart type) **so that** I can see more chart suggestions that are of that type.
17. **As a** registered user, I want to be able to filter by fields (select a charts fields) **so that** I can see more chart suggestion that include those fields.
18. **As a** registered user, I want to be able to view chart suggestions **so that** I can see what the data represents.
19. **As a** registered user, I want to be able to drill in to suggestions **so that** I can view fitter suggestions.
20. **As a** registered user, I want to be able to customize charts **so that** I can emphasize properties of the data.
21. **As a** registered user, I want to be able to restore deleted charts **so that** I can fix erroneous deletions.
22. **As a** registered user, I want to be able to export charts to csv or json format **so that** I can view the charts data in excel or a similar application as excel.
23. **As a** registered user, I want to be able to create charts **so that** I can view charts of my own creation.

## 1.9 Product Backlog

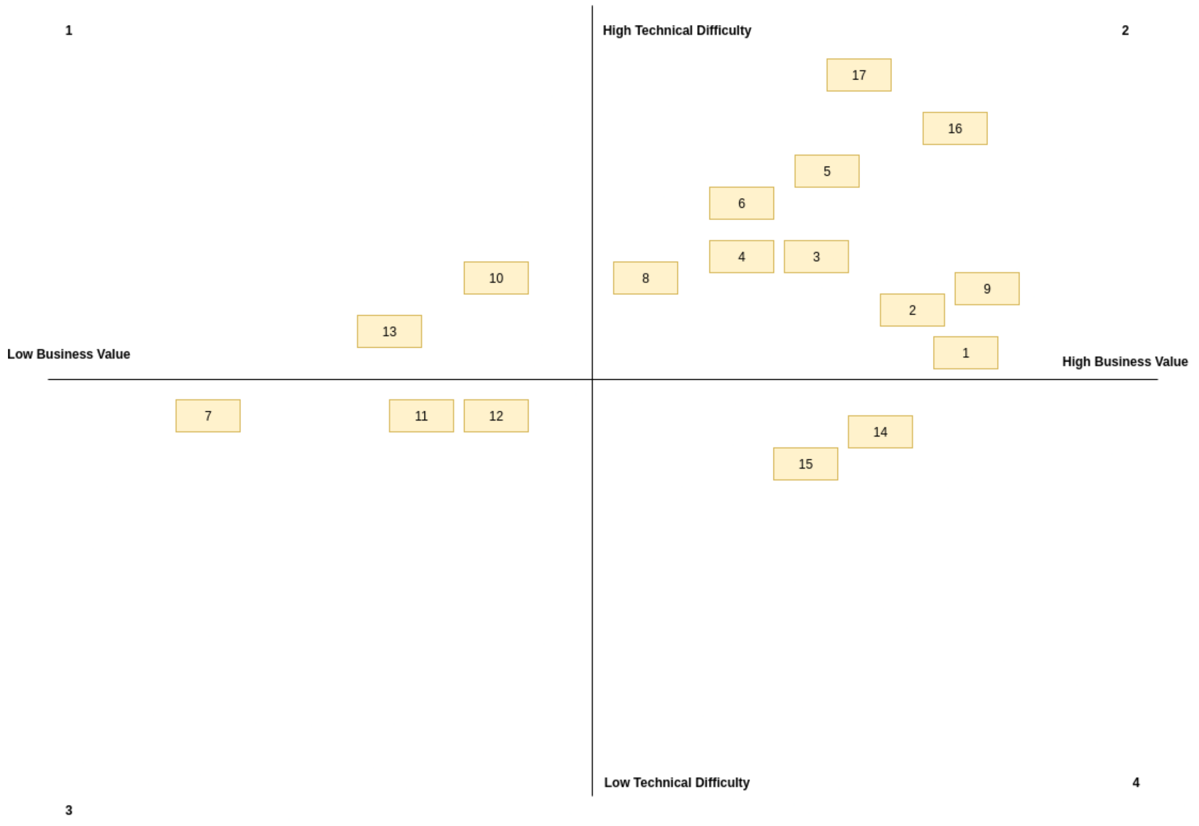


***Figure 2:*** Product Backlog diagram showing the user stories

# 2 Functional Requirements

## 2.1 Use cases

Actors - Registered User, Guest User or when referred to User we refer to both a Registered User and Guest User

**User Interaction Subsystem**

**UC01** Register users to the Data Visualization Generator system. (Actor: Registered User, System: Web Interface )

**UC02** Log in to the Data Visualization Generator system. (Actor: Registered User, System: Web Interface )

**UC03** Select a data source connection that should be used to generate suggestions. (Actor: User, System: Web Interface)

- **UC03.1** Select an existing data source connection. (Actor: User, System: Web Interface)
- **UC03.2** Add a new data source connection, by providing the system with an URL to this data source. (Actor: User, System: Web Interface)

**UC04** Create multiple dashboards. (Actor: Registered User, System: Web Interface)

**UC05** View existing dashboards. (Actor: Registered User, System: Web Interface)

- **UC05.1** View charts that were added to the dashboards (Actor: Registered User, System: Web Interface)

**UC06** Personalize dashboards. (Actor: Registered User, System: Web Interface)

- **UC06.1** Add a name to dashboards. (Actor: Registered User, System: Web Interface)
- **UC06.2** Add a description to dashboards. (Actor: Registered User, System: Web Interface )
- **UC06.3** Undo/Redo changes made to the dashboard like changing the name, description and colour of the dashboard. (Actor: Registered User, System: Web Interface)
- **UC06.4** Add chart suggestions to dashboards. (Actor: Registered User, System: Web Interface)

**UC07** Filter dashboards by searching for a charts name on the dashboard. (Actor: Registered User, System: Web Interface)

**UC08** Export chart suggestions. (Actor: User, System: Web Interface)

- **UC08.1** Export chart suggestions to json format. (Actor: User, System: Web Interface)
- **UC08.2** Export chart suggestions to csv format. (Actor: User, System: Web Interface)

**UC09** Filter chart suggestions. (Actor: User, System: Web Interface)

- **UC09.1** Filter through suggestions by selecting the type of charts that should be used to generate new suggestions of that type. (Actor: User, System: Web Interface)
- **UC09.2** Filter through suggestions by selecting the type of fields that should be used to generate new suggestions that include those fields. (Actor: User, System: Web Interface)
- **UC10** Be able to manually create charts based of off data provided by the system.

**Interactive Genetic Algorithm**

**UC10** Provide suggestions based off of the data source chosen. (Actor: User, System: IGA)

**UC11** Filter charts based off of given input by the user. (Actor: User, System: IGA)

- **UC11.1** Filter charts based off of selected fields, by generating suggestions that consider the selected fields. (Actor: User, System: IGA)
- **UC11.2** Filters charts based off of charts types, by generating suggestions that consider the selected chart types. (Actor: User, System: IGA)

**UC12** Drill in to suggestions. (Actor: User, System: IGA)

- **UC12.1** When a user interacts with a chart for example selecting it, the IGA will treat this as a fitter individual and trigger a next generation of suggestions. These suggestions will then indicate some kind of resemblance or property from the fittest suggestions of the previous generation.(Actor: User,System: IGA)

## 2.2 Use Case Diagram

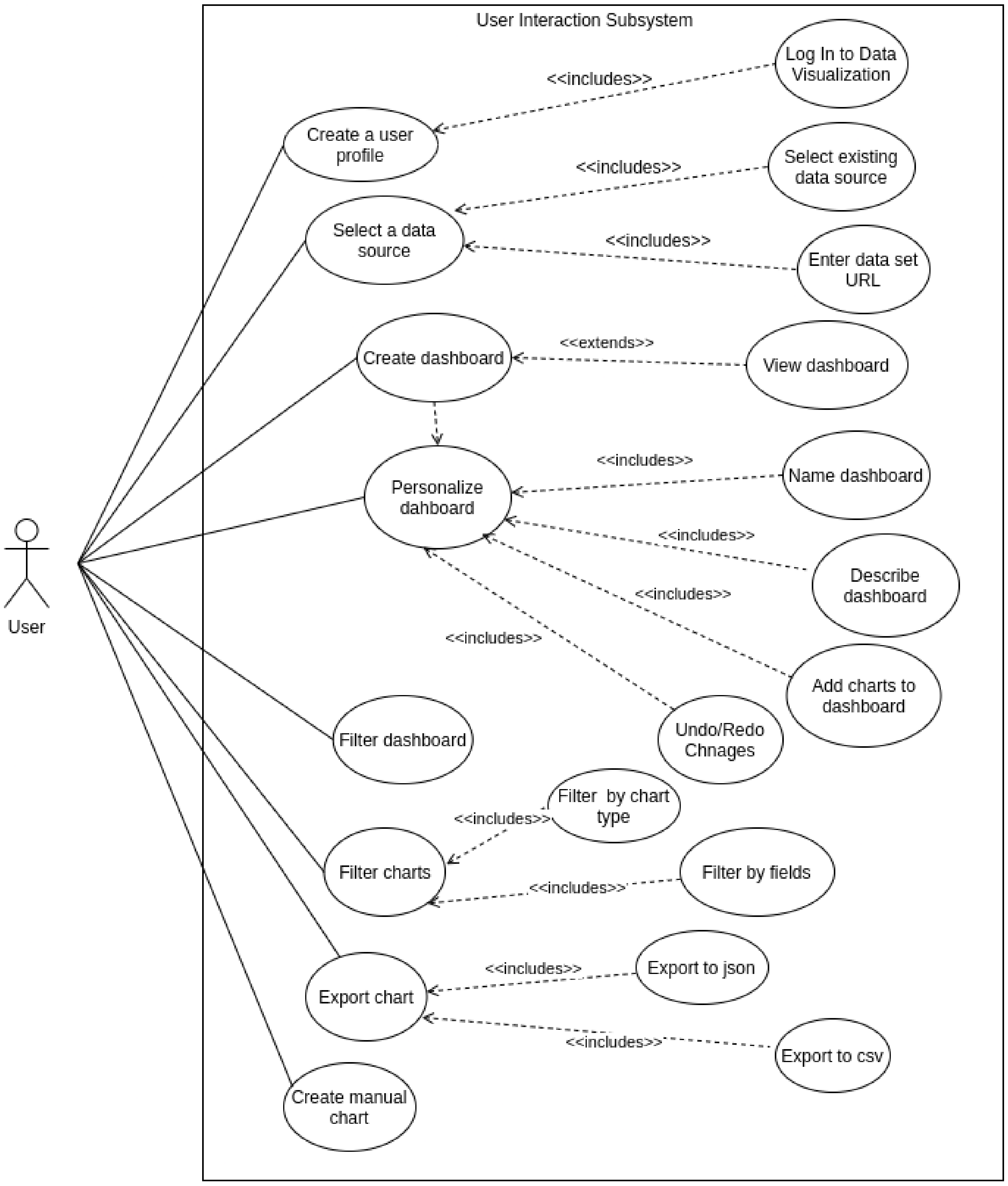


***Figure 3:*** Use Case diagram showing the use cases surrounding the **User Interaction** Subsystem

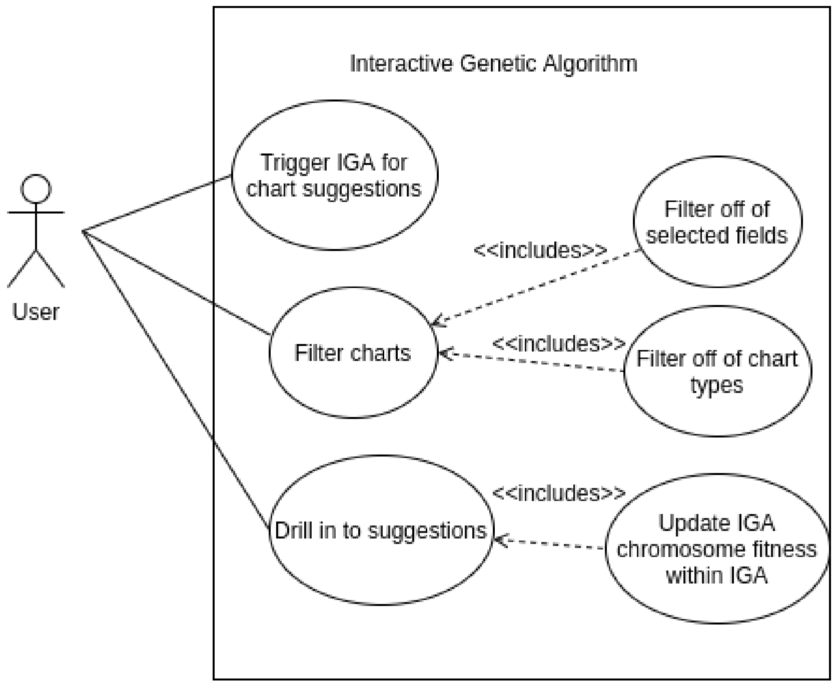

Interactive Genetic Algorithm
User
Trigger IGA for chart suggestions
Filter off of selected fields
<<includes>>
Filter charts
<<includes>>
Filter off of chart types
<<includes>>
Drill in to suggestions
Update IGA chromosome fitness within IGA

***Figure 4:*** Use Case diagram showing the use cases surrounding the **Interactive Genetic Algorithm** Subsystem

## 2.3 Functional Requirements

**R1** The Data Visualization Generator system must be able to let users create an account.

**R1.1** The system must be able to retrieve all of the user's dashboards that they created or saved.

**R1.2** The system must be able to let users retrieve all their preferences, such as chart types and filters selected (if any is present).

**R1.3** The system must be able to retrieve template dashboards that is provided by the system.

**R2** The Data Visualization Generator system must be able to let users select specific data sources that trigger a visualization suggestion generator.

**R2.1** The system must be able to provide the user with suggestions of that specific data source in the form of a visual representation via a chart etc.

**R2.2** An **IGA** is used for the suggestion generation. When drilling into these suggestions it uses the fittest individual and triggers the next generation of suggestions. These suggestions are used to indicate some kind of resemblance/property from the fittest suggestion of the previous generation.

**R2.3** The system must be able to let a user view and add these suggestions easily to a dashboard.

**R2.4** The system will also allow users to specify what type of fields they want in the chart and this will then provide a chart that reflects these fields selected.

**R3** The Data Visualization Generator system must be able to let users create/save multiple dashboards.

**R3.1** The system will allow users to give the dashboard a specific name and also a description of what the dashboard is about.

**R3.2** When the user views the suggestions provided by the system, the user can add these suggestions to dashboards. These dashboards are then customized to the specific user to visually track, analyze, and display key performance indicators.

**R3.3** Users can look at suggested dashboards provided by the system and may save these dashboards for use in the future.

**R4** The Data Visualization Generator system must be able to let the user selects a specific chart type.

**R4.1** When the user selects specific chart types then only that type needs to be used to create suggestions, otherwise, all types must be considered.

**R5** The Data Visualization Generator system must be able to let the user select different fields.

**R5.1** When the user selects different fields, then only these fields need to be used by suggestions, otherwise, all the fields and their data types must be considered.

**R6** The user needs to be able to add filters and when used they should filter all the visualizations simultaneously on the dashboard.
**R6.1** When the user adds a filter on the data set for example choosing a certain time frame, then all the visualizations should update simultaneously.

**R7** The user needs to be able to export suggested charts.
**R7.1** The user needs to be able to export suggested charts to csv format.
**R7.2** The user needs to be able to export suggested charts to json format.

**R8** The users needs to be able to create manual charts, based off of data that is provided by the system.

**R9** The progressive web applications interface should be responsive.
**R9.1** The interface should adjust to different screen sizes, including mobile phones.

## 2.4 Subsystems

### S1. REST Controller Subsystem

***Description***
This subsystem is responsible for handling requests from the Progressive Web Application Subsystem. This entails delegating parts of the request to the Data Source Subsystem, the Interactive Genetic Algorithm Subsystem, or the Data Storage and Organisation Subsystem.

- R1.1
- R1.2
- R1.3
- R2.1
- R2.2
- R2.3
- R2.4
- R3.1
- R3.2
- R3.3
- R4.1
- R5.1
- R6.1
- R7.1
- R9.1

### S2. Data Storage and Organisation Subsystem

***Description***

This subsystem is responsible for maintaining the database and performing **CRUD** operations on the database. It will the interface for database operations to all other subsystems.
***Related Functional Requirements:***

- R1.1
- R1.2
- R1.3
- R3.1
- R3.3

## S3. Interactive Genetic Algorithm Subsystem

***Description***

This subsystem is responsible for generating suggestions for charts when it is triggered. This is triggered via a user using the web interface and must generate chart suggestions based off the given data source and fields.
***Related Functional Requirements:***

- R2.1
- R2.2
- R3.2
- R4.1
- R5.1

## S4. Data Source Subsystem

***Description***

This subsystem is responsible for managing requests to external data sources and caching the responses of such requests.
***Related Functional Requirements:***

- R2.3
- R2.4
- R3.2
- R3.3
- R6.1

## S5. Progressive Web Application Subsystem

***Description***
This subsystem is responsible for the web interface, this will therefore handle any interaction between the user and the system as the web interface is the main means for a user to use the Data Visualisation system.
***Related Functional Requirements:***

- R2.1
- R2.3
- R2.4
- R3.1
- R3.2
- R3.3

- R4.1
- R5.1
- R6.1
- R7.1
- R8
- R9.1

## 3 Quality Requirements

The requirements in this section provide information about the quality of the application and what the application should be able to achieve.

**Q1. Performance**

- Data Visualization Generator system must be able to respond to an initial request in under 10 seconds but this will depend on the internet connection strength and location of the server.
- It must be able to handle 180 user requests per second.

  This will ensure that the user will not get frustrated and wait for a long period of time for their request to be handled.

**Q2. Reliability and Availability**
The Data Visualization Generator system will be hosted on an external server with a contractual agreement with the service provider.

1. **Reliability:**
   - The Data Visualization Generator system must be tested before deployment and the system must behave the same in deployment as it did in testing.

     This will ensure that the system behaves in an expected manner, the application must behave in the same way as it did in the development stage for every user, as it did when the product owners conducted tests on the application.
2. **Availability:**
   - The Data Visualization Generator system must available for 98.9% throughout its lifetime.
   - The Data Visualization Generator system must have access to the databases for 98.9% throughout its lifetime.

     This will ensure that the user will trust the application as the user will be able to use the application as much as possible.

**Q3. Scalability**

- The Data Visualization Generator system must be designed using appropriate design patterns to allow for easy scalability of the functionality of the system.

  This ensures that the system is easily maintainable and easy to fix. This will provide that the application can be extended with more functionality without having to change a lot of modules.

**Q4. Security and Maintainability**

1. **Security:**
   - **Data Storage:**
     - The data of users must be stored in a secure manner and must have controlled access.

  - All data that conform to the Customer Laws need to be logged for the required amount of time and must be deleted after a certain amount of time.
  - Audit logs must be stored and must only be accessible to the product owners.
  - Client passwords must be hashed and salted before storage.

  This will ensure that all personal information that will be stored on the database would not be used in a malicious way. It is also important to log the required information about people by law.

- **Data Transfer:**
  - Data sent over the internet must be encrypted and securely transferred between different locations.
  - When sending data, a user must be prompted to transfer the data or deny the transfer of data.

  This will ensure that all personal information would be securely transferred and cannot be obtained by any *'man in the middle attacks'*. This will be achieved by encryption and ensures the privacy of user data.

- **Data Access:**
  - All data must have clearance levels associated with it, which will give controlled access to data.
  - All data logs must have controlled access and can only be accessed through an interface (not API), only users with desired clearance levels may access the data.
  - The owner of the product must be able to add privileges or remove privileges from a client.

  This will ensure that data will be kept private and cannot be accessed by people without the necessary clearance levels,

2. **Maintainability:**

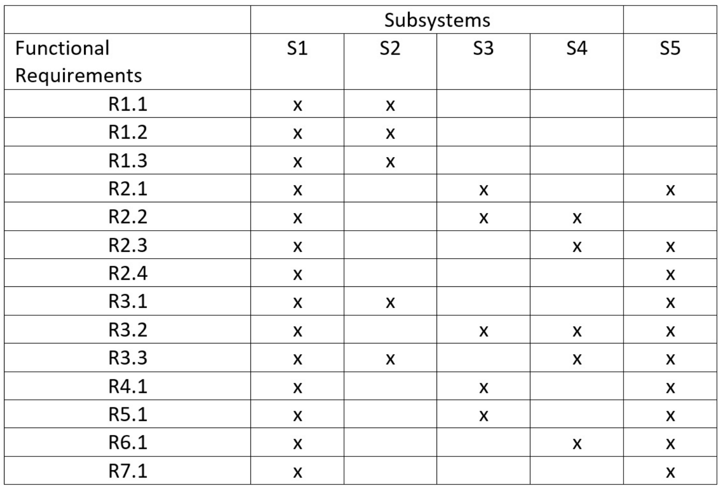
   - To ensure our system is easily maintained, a modular approach would be followed. This approach is know as **Modularization**, this principle facilitates the functionality of low coupling and high cohesion. This means the system must be sub-divided in to smaller systems that make maintenance and updates to the system easier.

**Q5. Usability**

- The Data Visualization Generator system has an easy to navigate user interface to allow all users to understand the application. Data Visualization Generator system has been designed in a vertical approach rather than a horizontal design to make navigating through the application easier.

The user-interface must provide the user with a good user experience **(UX)**, a user-friendly product is in general more successful and users will find the system more reliable if the system is self-explanatory. This is an ideal situation as this means our **goal of developing a successful product is met**.

## 4 Traceability matrix



| | Subsystems | | | | |
|---|---|---|---|---|---|
| Functional Requirements | S1 | S2 | S3 | S4 | S5 |
| R1.1 | x | x | | | |
| R1.2 | x | x | | | |
| R1.3 | x | x | | | |
| R2.1 | x | | x | | x |
| R2.2 | x | | x | x | |
| R2.3 | x | | | x | x |
| R2.4 | x | | | | x |
| R3.1 | x | x | | | x |
| R3.2 | x | | x | x | x |
| R3.3 | x | x | | x | x |
| R4.1 | x | | x | | x |
| R5.1 | x | | x | | x |
| R6.1 | x | | | x | x |
| R7.1 | x | | | | x |

***Figure 5:*** Traceability matrix

## 5 Architectural Design

### 5.1 Objective

The Data Visualization Generator System is a N-tier architecture that is encapsulated by a Client-Server architecture. With the N-tier architecture, the presentation (**Client Layer**), business logic (**Application Layer**), **Network Layer** and data management (**Data layer**) are both logically and physically separated. These function can run on separate machines or separate clusters so that each function is able to provide the services at top capacity since no resource sharing would occur. The separation makes managing each function separately easier and doing work on one component does not affect the other. N-tier architecture is also known as multi-tier architecture. With Client Server Architecture, the **server** hosts, delivers and manages most of the resources and services to be **client** by the client. This type of architecture has one or more client computers connected to a central server over a network or internet connection.

### 5.2 Reasoning

The reason why the N-tier architecture was chosen is because it perfectly aligns with our requirements and suites our needs. The following reasons are why the Data Visualization Generator System would benefit from using the N-tier architecture.

1. **Better Security:** This architecture enforces security for each tier. The architecture gives you full control of security at each layer. For example, the business logic layer and data layer needs more security than the presentation layer.

2. **Scalability:** Allows each tier to scale as needed, meaning the application will grow with our systems needs. For example, the database can be scaled by database clustering without touching the other tiers.

3. **Easy to maintain:** It is easier to maintain, different people can manage different tiers without the risk of touching other tiers. For example, the front-end designer should work on the presentation layer and not the other layers.

4. **Easily enhanced:** Allows you to easily upgrade or modify applications with each layer being upgraded individually. For example, additional databases can be added if the system expands or new business rules can be added to the application layer.

## 5.3 Architectural Design Diagram

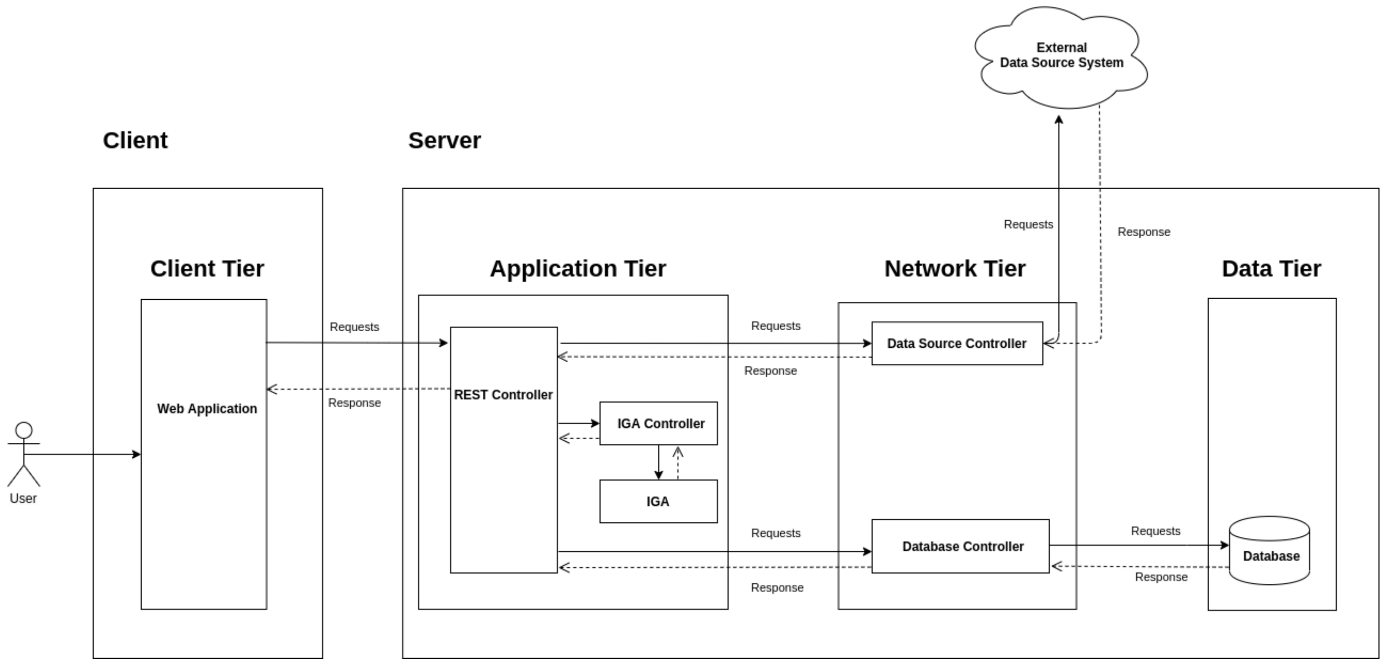


***Figure 6:*** Architectural Diagram of the complete application.

### Description

The system consist of a four-tier architectural style.

1. The first tier represents the front-end of the web application (**Client Tier**). This would be the layer that the user interacts with and would be responsive depending on the platform and/or device type the web application is viewed on. All system processes start here, with the client sending user requests to the server, which triggers the server to perform the necessary operations, and respond to that request once it finishes.
2. The second tier is the (**Application Tier**) and contains the REST Controller that is responsible for delegating tasks to the Network Tier and also contains the IGA Controller and IGA for performing AI calculations in our system. Any requests from the client are received by the REST Controller before being delegated to the next tier.
3. The third tier is the (**Network Tier**) and consists of the Data Source Controller that will handle communication with a remote data source system as well as perform caching, so that less requests need to be made to the data source and thus speeding up requests for data. It also includes the Database Controller, which controls access to the Database. All communication with the Database takes place through the Database Controller.
4. The last tier is the (**Data Tier**) which contains the Database. The Database stores all user-related data, such as accounts, saved dashboards, etc. Access to the Database is controlled through the Database Controller.

## 5.4 Deployment Model

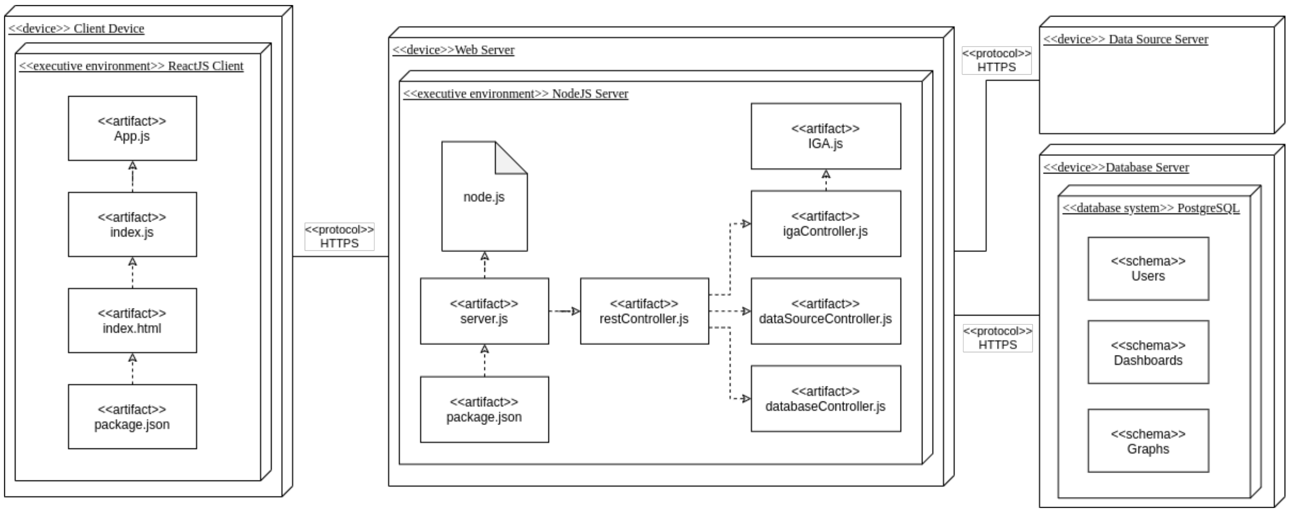

<<device>> Client Device
<<executive environment>> ReactJS Client
<<artifact>> App.js
<<artifact>> index.js
<<artifact>> index.html
<<artifact>> package.json
<<protocol>> HTTPS
<<device>>Web Server
<<executive environment>> NodeJS Server
node.js
<<artifact>> server.js
<<artifact>> package.json
<<artifact>> restController.js
<<artifact>> IGA.js
<<artifact>> igaController.js
<<artifact>> dataSourceController.js
<<artifact>> databaseController.js
<<protocol>> HTTPS
<<device>> Data Source Server
<<protocol>> HTTPS
<<device>>Database Server
<<database system>> PostgreSQL
<<schema>> Users
<<schema>> Dashboards
<<schema>> Graphs

***Figure 7:*** Deployment Model of the complete application.

**Deployment Model Description**

- The **Client Device** sub-model is the client's web browser. App.js is the main React component that contains all the sub-components and manages the state and logic of the Progressive Web Application. index.js renders the App component onto the index.html files, which is rendered on the web browser. This package.json contains all the packages and dependencies that are used by the Progressive Web Application. This Application communicates with the Web Server through HTTP requests.
- The **Web Server** sub-model contains the Node.js server environment. server.js is an Express.js server that delegates HTTP requests from the Client Application to restController.js which handles them.
- **restController.js** is a JavaScript module that contains all the logic for handling a variety of requests. Depending on the request, other requests are made to databaseController.js, dataSourceController.js, and igaController.js.
- **databaseController.js** is a JavaScript module that serves as an API for communicating with the PostgreSQL database. This module communicates with the database through HTTP requests.
- **dataSourceController.js** is a JavaScript module that serves as an API for retrieving data from external data sources.This module communicates with external data sources through HTTP requests.
- **igaController.js** is a JavaScript module that serves as an API for communicating the IGA.js.
- **IGA.js** is an Interactive Genetic Algorithm that is encapsulated in a JavaScript module. This module generates chart suggestions based on the data it receives from the restController.js, which the restController.js gets from the dataSourceController.js
- The **Database Server** sub-model contains the PostgreSQL database server. The database contains three collections:
  - Users
  - Dashboards
  - Charts
- The **Data Source** sub-model refers to any external data source that the dataSourceController.js can communicate with. These data sources may vary in the way that their data is structured and how their API's work.

## 6 Constraints

Our architecture consists of four controllers that control access to individual subsystems. Multiple controllers increases integration complexity and can produce increased communication overhead. However, separation of concerns decreased coupling. **Constraints regarding controllers include:**

- All communication to the web server is to be done through the REST Controller as it serves as an API.
- All communication to external data sources is to be done through the Data source Controller to handle different variants of data and data caching.
- All communication to the IGA must be done through the IGA Controller. This simplifies requests and responses to the IGA component.
- All access to data in the Database must be sent to a Database Controller, no direct database access is permitted.

**Other constrains:**

- Format of the data received from external sources is required (for now) to be structured or at the least semi-structured as the system does not have the functionality to process Big Data yet.

## 7 Technology requirements and decisions

- Version control: **GitHub**
  GitHub achieves all we will require in terms of version control. It enables is to track changes of who did what and where.
- Project management: **ZenHub**
  ZenHub works well with agile project development. It integrates seamlessly with GitHub and allows us to have our project management tool in the same location as our repository. The interface is user-friendly making communicating on the progress of each task very simple and efficient.
- Continuous Integration: **CircleCI**
  CircleCI was our choice for continuous integration as it is GitHub friendly, has good support and we can ensure that our tests get a fresh run because each build runs in a clean LXC Container.
- Testing: **JEST**
  JEST is an open source testing framework that is easy to set up and works well with React (our front-end framework). This testing frameworks allow us to easily execute tests locally during development in order to ensure our code is working and producing the correct outputs.
- Linting: **ESLint**
  ESLint makes it easier to keep code consistent and standard helping us to avoid coding bugs.
- Front End Framework: **React**
  We have considered using Vue, Angular or React as each of them are more or less suitable for our needs. React was our final choice as React is based on single-direction data flow and has lots of flexibility. Angular has a bit of a steeper learning curve and Vue has a very small community for support.
- Hosting provider: **Heroku**
  We make use of Heroku as it is a free hosting service that is startup-friendly. It offers you a ready-to-use environment that allows you to deploy your code fast whereas the deployment process of other services such as Amazon Web Services is quite complicated.
- Server-side Framework: **NodeJS Express**
  Express.js simplifies development and makes it simpler to write secure, modular and fast applications. For example, a lot of the code to conduct HTTP requests is already done for us. NodeJS offers event-driven environment that allows us to easily handle asynchronous calls and performance is better than most other languages like python. It was chosen so that we have the ability to keep data in native JSON (object notation) format in your database should we need to.

- Database Management System (DBMS): **PostgreSQL**
  We have chosen to use a SQL database instead of something like Hadoop because our data that we receive is expected to be structured (As specified by the client). Should we at a stage be required to receive semi-structured data such as JSON, PostgreSQL allows you to use this data to transform PostgreSQL into a NoSQL database with the data types being index which provides speed and data integrity.

## References

[1] Min Chen, Shiwen Mao, and Yunhao Liu. Big data: A survey. *Mobile networks and applications*, 19(2):171–209, 2014.

[2] Dun-wei Gong, Xiao-yan Sun, and Jie Yuan. Interactive genetic algorithms with individual's uncertain fitness. *Evolutionary Computation*, pages 21–44, 2009.

[3] Jacob Andrew Taylor, Majed Itani, Ajay Gupta, Andrew Wu, Joseph Parsons, Roger Smith, and Chris Nojima. Crm system and method having drilldowns, acls, shared folders, a tracker and a module builder, March 12 2009. US Patent App. 12/200,301.

[4] Lidong Wang, Guanghui Wang, and Cheryl Ann Alexander. Big data and visualization: methods, challenges and technology progress. *Digital Technologies*, 1(1):33–38, 2015.