DMHP PROJECT REPORT

May 14, 2021

Divyanshu Sharma - MT2020061
Neeraj Jetha - MT2020079
Gourav Sachdev - MT2020109

Contents

1 Introduction

The District Mental Health Programme (DMHP), started under the National Mental Health Programme (NMHP), decentralizes the mental health services and provides mental health services at the community level by integrating mental health with the general healthcare delivery system. Every month many patients with different mental disorders visit this programme. The National Institute of Mental Health and Neurosciences (NIMHANS) provides training for medical officers and health professionals to empower them in diagnosing and treating mental health disorders.

Every month data from 31 districts of the Karnataka state and their talukas is being collected. This data includes the count of patients with its type, expense spend across various sections, and training conducted for various departments. At the end of every month, a brief report is manually created.Currently, DMHP has live web application which is used to visualize and analyze the dump data.

The main **objective** of this project is:

1. Revamped the Table
2. To create a generic SQL query so their is no need for multiple SQL queries visualization of any new data.
3. Instead of multiple API their should be single API call.

Our seniors made the DMHP live web application but there were multiple API's and each API's has static SQL queries ,future analytics and visualization require writing of multiple API's and SQL queries ,so our work was to optimize the backend part to convert multiple API's call to generic API call interacting with the database we have also generalize the SQL queries.

2 Links

- Stored Procedure : Click here to visit
- Revamped Table: Click here to visit

- Ordering: Click here to visit
- Project: Click here to visit

3 Technology Stack

Web Development Framework	Angular v8.3.22
JavaScript Visualisation Library	D3 v5.15.1
Backend	NodeJS v12.14.0
Database	MySQL Server

4 Challenges faced

4.0.1 Revamped the Table

The Table tbl_reportdata on which the website is live has too many columns due to which the number of parameters increases in dynamic SQL queries so we have created a new table name ReportData which makes the queries easy and reduces the complexity

4.0.2 Dynamic Stored Procedure

Dynamic stored procedure create the SQL queries on the runtime .It take the input in parameters and generate the string at the end prepared statement and execute statement is called to execute the query. Dynamic query take 12 input parameters.

4.0.3 Single API call

Their are multiple API calls in the backend since doing optimization in the SQL ,their become possibility to make the API generic ,so we have made the generic JSON format and calling the stored procedure from the single API

5 Revamped Table

1. In table tbl_reportdata has disease columns in the format visittype_disease_gender but now we break the column name into 3 parts in new table ReportData i.e Visit-Type, Disease, Gender

2. By breaking the column number of rows increases and number of columns decreases. For each row in old table corresponding to that their are 4 more rows added.

3. For each gender i.e {male ,female} their are 2 combination of visit type i.e {old, new} that make total of 4 rows and additional row for which gender is not specified and visit type for that is also not specified that makes total of 5 rows.

4. Overall columns names in ReportData are **[ReportId, StateId, DistrictId, TalukaId, FacilityTypeId, FacilityId, ReportingMonthYear, VisitType, Gender, ConsultType, SMD, CMD, SuicideAttempts, Epilepsy, AlcoholSubstanceAbuse, Dementia, DevelopmentDisorders, BehaviouralDisorders , EmotionalDisorders, PsychiatricDisorders, Others, Referred, TotalPaitents]**.

6 Dynamic Stored Procedure

Dynamic SQL is a programming technique that enables you to build SQL statements dynamically at runtime. You can create more general purpose, flexible applications by using dynamic SQL because the full text of a SQL statement may be unknown at compilation. We basically have used the dynamic SQL in stored procedure.

When to use Dynamic SQL?

Dynamic SQL programs can handle changes in data definition information, because the SQL statements can change "on the fly" at runtime. Therefore, dynamic SQL is much more flexible than static SQL. Dynamic SQL enables you to write application code that is reusable because the code defines a process that is independent of the specific SQL statements used. With static SQL, all of the data definition information, such as table definitions, referenced by the SQL statements in your program must be known at compilation. If the data definition changes, you must change and recompile the program.

In addition, dynamic SQL lets you execute SQL statements that are not supported in static SQL programs, such as data definition language (DDL) statements.

6.1 Structure of stored procedure

```
CREATE PROCEDURE procedure_name [[IN | OUT | INOUT] parameter_name datatype(size)]
BEGIN
    Declaration_section
```

```
    Executable_section
END
```

6.2 Parameter Explanation

Parameter Explanation	
Parameter Name	Description
1.procedure_name	It represents the name of the stored procedure.
2.parameter	It represents the number of parameters. It can be one or more than one.
3.Declaration_section	It represents the declarations of all variables.
4.Executable_section	It represents the code for the function execution.

6.3 Parameters used in Dynamic stored procedure

1. **display**: This parameter will contain the list of the column name of ReportData that you want to display generally it is subset of the group_by parameter

2. **disease**: This parameter will include the list of the Diseases column name whose data you want to see

3. **start_date**: This parameter will include the starting date in the format **YYYY-MM-DD**

4. **end_date**: This parameter will include the ending date in the format **YYYY-MM-DD**

5. **visit_type**: This parameter will include the list of values in the column name VisitType like new, old, N (Notspecified) on which values you need to filter the rows.

6. **gender_string**: This parameter will include the list of values in the column name Gender Type like **M (Male),F (Female),O (Others),N (NotSpecified)** on which values you need to filter the rows.

7. **district_list**: This parameter will contain the list of values which is numerical in column DistrictId on which filtering of rows will take place.If nothing is specified than it will do filtering on all the distinct values in DistrictId

8. **taluka_list**: This parameter will contain the list of values of Column TalukaId on which filtering of rows will take place.If nothing is specified than it will do filtering on all the distinct values in TalukaId.

9. **group_by**: This parameter will contain list of column names on which you want to apply groupby

10. **timeperiod_type**: This parameter will contain whether you want to display data on what type means annually ,quaterlly , monthly

11. **year_type**: This parameter will take the value either **c(Calendar), f(financial) on** what calendar type data should be displayed

6.4 How to provide values into the Parameter

1. **display : 'ReportingMonthyear,DistrictId'** column name should be seperated by commas and all the values of column name are closed within single quotes bcoz we are passing a list of values as a string

2. **disease : 'SMD,CMD,SuicideAttempts,Referred'** column name should be seperated by commas and all the values of column name are closed within single quotes bcoz we are passing a list of values as a string

3. **start_date : '2017-08-01'** It should be passed like this

4. **visit_type : "'new','old'"** It contains the values of columnVisitType.Its format of passing the value in parameter format is different bcoz we are not passing the column name here we are passing the values in the column name whose type isvarchar

5. **gender_string : "'M','F'"** It contains the values of column Gender .If we want don't provide the value here means" "it will do filtering on all the distinct values in the gender column.

6. **facilitytype_list** It contains the numerical value 1,2,3,8 which case be passed as string like **'1,2,3,8'**

7. **district_list : '3,43,45'** here we are passing the number in the form of the string if we don't provide any value in parameter means '' it will do filtering on all the distinct values of DistrictId.

8. **group_by : 'ReportingMonthyear,DistrictId'** It will contains the column name you can write the column name in any order

9. **timeperiod_type : 'quaterlly'/ 'annually'/'monthly'** It can take any one of the value

10. **year_type : 'c'/'f'** Either the calendar type will be financial or Calendar but can't be both

11. **timeperiodtype** is the procedure name

Code Snippet

```
call DMHP.timeperiodtype('DistrictId', 'SMD,CMD,SuicideAttempts,Referred',
'2017-08-01', '2018-12-01', "'new'", "",'1,2,3,8','3,43,45','','ReportingMonthyear,
DistrictId','annually','c');
```

The above query display the disease data of whose DistrictId is 3,43 and 45 where visit type is new and gender can be anything where groupby done on column Reporting-MonthYear and DistrictId on the basis of calendar year between the date 2017-08-01 to 2018-12-01.

6.5 Values parameters can hold

1. **display** {DistrictId,TalukaId,ReportingMonthyear,ReportId}

2. **disease** {SMD,CMD,SuicideAttempts,Epilepsy,AlcoholSubstanceAbuse ,Dementia, DevelopmentalDisorders,BehaviouralDisorders, EmotionalDisorders, PsychiatricDisorders,Others, Referred}

3. **start_date** {YYYY-MM-DD}

4. **end_date** {YYYY-MM-DD}

5. **visit_type** {new,old,N

6. **gender_string** {M,F,N,O}

7. **facilitytype_string** {1,2,3,8}

8. **district_list** {Distinct values in column of DistrictId }
9. **taluka_list** {Distinct values in column of TalukaId }
10. **group_by** {DistrictId,TalukaId,ReportingMonthyear,ReportId}
11. **timeperiod_type** {annually,quaterlly,monthly}
12. **year_type** {c,f}

7 Single API call

Main purpose of single API call is to reduce the writing of SQL queries through which their is no need to write SQL queries for each API call. We can call generic stored procedure directly by passing the parameter.
Now ,we have converted the static query into dynamic query by passing the parameters using prepared statement. Earlier static queries were written which were passed as string to the database and data is fetched but now along we are calling the procedure name and passing the parameters whose values are coming from the json.

7.1 JSON format of single API call

```
{
"display_type" :  ["DistrictId"],
"diseases" :      ["SMD,CMD,SuicideAttempts,Referred"],
"start_date" :    "2020-01-01" ,
"end_date" :      "2020-12-01",
"visit_type" :    ["'new','old'"],
"gender" :        ["'M','F'"],
"district_id" :   [""],
"taluka_id" :     [""],
"group_by" :      ["DistrictId"],
"time_period" :   "anually",
"calender_type" : "c"
}
```

Code snippet on next page shows the difference between the existing code and new code which is single API call.

>>>>>>>>>>>>>>>>>>Code before making changes<<<<<<<<<<<<<<<<<<<

```
app.post("/getDataAllDistrictAnnually", (req, res) => {
  var year = req.body.year;

  sql = ‘select m.DistrictId, d.District, d.Population,’ + cases +
    ‘from tbl_reportdata m, Districts d
    where year(ReportingMonthyear)=? and m.DistrictId = d.DistrictId
    group by m.DistrictId,d.Population, d.District
    order by d.District’;

  con.query(sql, [year], function (err, response) {
    // if (err) console.log(err);
    res.json(response);
  });
})
```

>>>>>>>>>>>>>>>>>>Code after making changes<<<<<<<<<<<<<<<<<<<

```
app.post("/getData",(req,res)=>{

var display_type=req.body.group_by;   //‘DistrictId’
var diseases=req.body.diseases;       // ‘SMD,CMD,SuicideAttempts,Referred’
var start_date=req.body.start_date;    // ’2020-01-01’
var end_date=req.body.end_date;        // ’2020-01-01’
var visit_type=req.body.visit_type;     //   "‘’"   or "‘new’,‘old’"
var gender=req.body.gender;              //   "‘M’"
var district_id=req.body.district_id;    // ‘3,30,45’
var taluka_id=req.body.taluka_id;        // ‘205,210,217’
var group_by=req.body.group_by;           //  ‘DistrictId,ReportingMonthYear’
var time_period=req.body.time_period;      //  ‘annually’
var calender_type=req.body.calender_type;  //  ‘c’
```

```
sql='call dhmp.timeperiodtype(?, ?,?, ?, ?, ?,?,?,?,?,?);';
```

con.query(sql,[display_type,diseases,start_date,end_date,visit_type,gender,district_id,taluka_id, group_by,time_period,calender_type],function (err, response)

```
if (response != null) {
res.json(response[0]);
console.log(response);
}
else
res.json(response);
});
})
```

8 Future Scope

As the project is developed keeping reusability of stored procedure in mind, makes it very easy to visualize new data just by changing some values in the parameters. In future following aspects can be added to the project:

1. **Integrating with machine learning**
   Using machine learning models to find trends and correlations in the data. This will help to analyze and predict future trends and will also help to prepare and take appropriate actions well in advance

2. **Adding more stored procedure**
   Different stored procedure can be made for different tables or we can add the parameter table name in the stored procedure. It's better to create new procedure

3. **Extending this project to different diseases/outbreaks or using this project for different hospital-cases or healthcare-projects**
   This project can be easily extended to other diseases like heart, cancer, lungs,COVID etc. or just like DMHP different healthcare projects like Integrated Disease Surveillance Programme (IDSP), Guinea Worm Eradication Programme (GWEP),etc. or even visualizing hospital specific data like AIIMS hospital, Safdarjung Hospital, etc.

9 References

- Dynamic Query inside Stored Procedure
- Examples of Dynamic Stored Procedure
- How to write Dynamic SQL Query in MySQL Stored Procedure
- How to pass an array to a MySQL stored procedure?
- Passing a list to a stored procedure
- Calling MySQL Stored Procedures from Node.js

Appendix

A1 Introduction

The District Mental Health Programme (DMHP), started under the National Mental Health Programme (NMHP), decentralizes the mental health services and provides mental health services at the community level by integrating mental health with the general healthcare delivery system. Every month many patients with different mental disorders visit this programme. The National Institute of Mental Health and Neurosciences (NIMHANS) provides training for medical officers and health professionals to empower them in diagnosing and treating mental health disorders.

Every month data from 31 districts of the Karnataka state and their talukas is being collected. This data includes the count of patients with its type, expense spend across various sections, and training conducted for various departments. At the end of every month, a brief report is manually created. Currently, DMHP has no tool which will visualize and analyze this dump of data. This makes it very difficult to understand the statistics/trends of the data and generate the report.

The main **objective** of this project is:

1. To create a project that will visualize the DMHP data.
2. To make each module of the project (especially charts and services) reusable, which can be reused in the visualization of any new data.

This will enable DMHP professionals to access important statistics and will eventually help to increase the overall performance.

The project includes two major dashboards -

1. **Operational Dashboard**
   Provides information on progress for a particular year - number of patients visited across districts and across time, and can be drill down to taluka level, expense spend across districts, total training conducted across districts These visualizations have features like:
   - Granularity - show data annually, month-wise and quarter-wise
   - Normalize with respect to the population
   - Sort data by rank, district name or population

- The data can also be viewed in the table and can be downloaded as a CSV file.

2. **Performance Analysis Dashboard**
   Provides a comparison between - training vs expense vs number of patients across the district and across time for a particular year.

All these visualisations can be downloaded as a picture using screenshot feature.

A2 Links

- GitHub : https://github.com/VipinRaiP/DMHP
- Click to view the demo on YouTube

A3 Technology Stack

Web Development Framework	**Angular v8.3.22**
JavaScript Visualisation Library	**D3 v5.15.1**
Backend	**NodeJS v12.14.0**
Database	**MySQL Server**

A4 Architecture and Block Diagrams

A4.1 System overview

A4.1.1 Operations Dashboard

1. Operations dashboard deals with data for a particular year. User can choose to view patient count(number of cases), training and expense data.

2. Summary section presents data of entire state for different features. Variation can be seen over months using line chart and multi line chart. Same data can be seen in table.

3. In the next section data can be viewed across districts. Granularity can be set as annual, quarter and monthly. Data is presented in both stacked bar chart and Karnataka map.

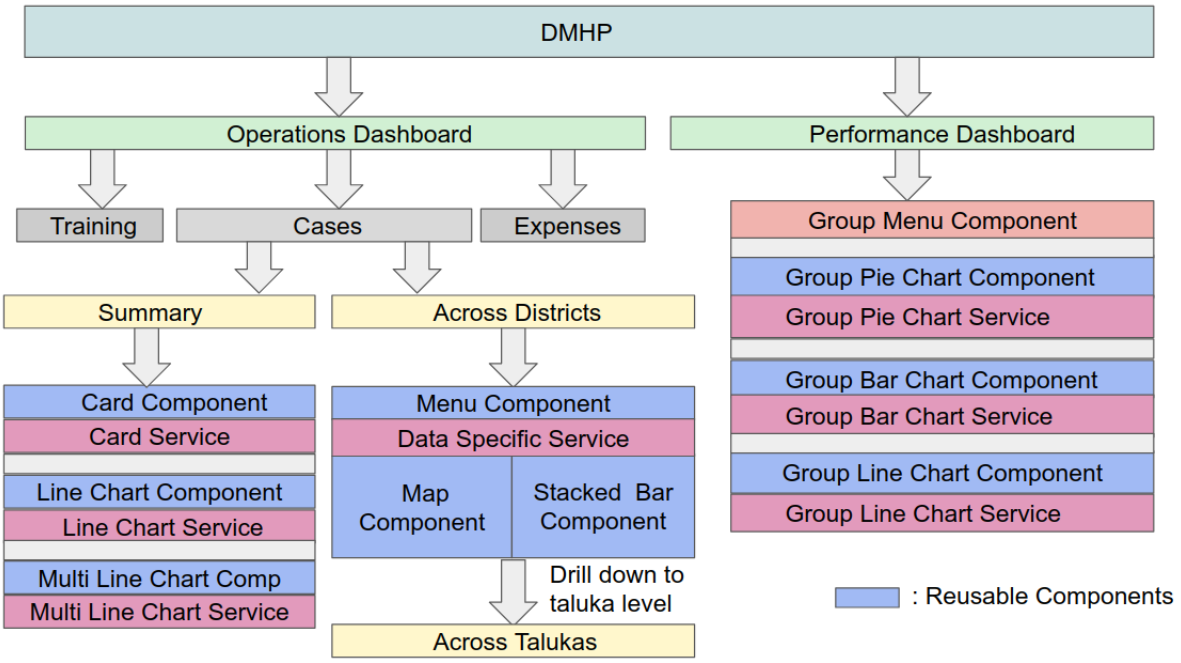


Figure 1: System overview

4. On double clicking a particular district it will drill down to a particular taluka view. Same controls as above are replicated.

5. Features represented in the charts can be filtered by toggling the legends (clicking over feature names in the legend).

6. Districts or Talukas can be ranked as per the data count or by name. Data can also be presented by normalising with respect to district population.

7. Options are provided to view the data in tabular form and to download in CSV format.

A4.1.2 Performance Dashboard

1. Performance dashboard is used to compare different features over a year. Entire state data is taken into consideration.

2. Data is presented using pie chart, bar chart and multi line chart. Different colors are used to indicate different features and are placed side by side for comparison.

3. In the bar cart, data can be sorted as per different features and also district names.

4. Features represented in the charts can be filtered by toggling the legends (clicking over feature names in the legend).

A4.2 Chart and Menu component interaction

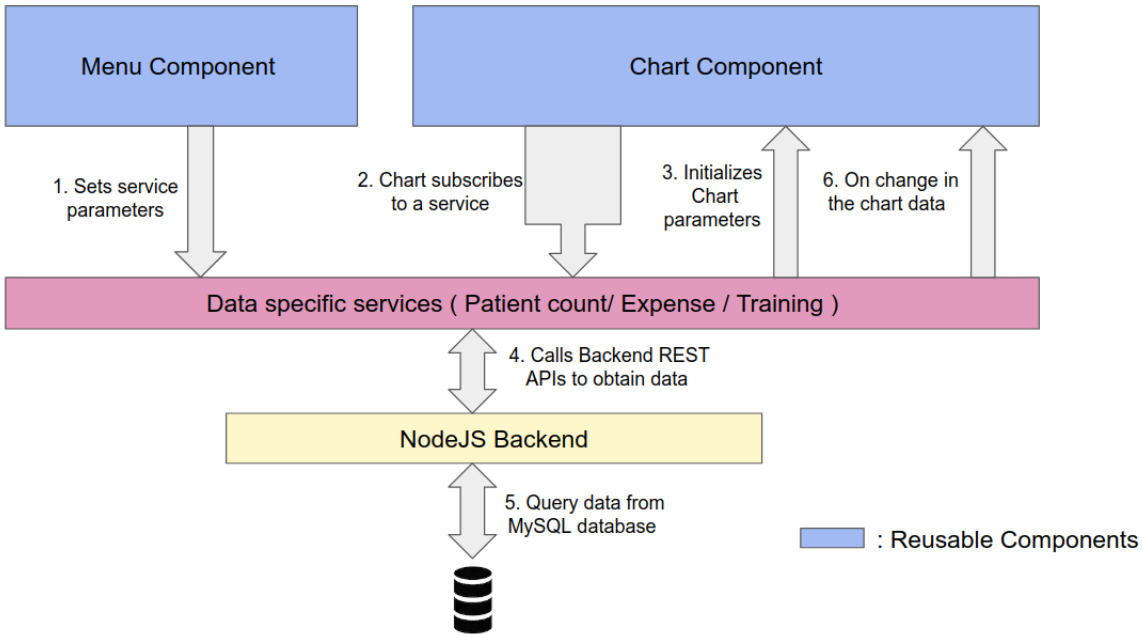


Figure 2: Menu and Chart component interaction

A4.2.1 Services

1. Service in Angular is a class which can be used by different components to communicate among themselves.

2. Service has methods to retrieve the data from backend. Methods calls the backend REST API to retrieve the data.

3. Service has attributes like granular(year, month ,quarter etc..) that define the data that is required by the chart. These attributes can be set from menu based on user selection.

4. Some of the services used :

    - Patient Count Service : Calls backend API to retrieve data related to number of patients.

- Training Service : Retrieves data from backend related to the trainings conducted.

A4.2.2 Menu Component

1. Menu component has controls to vary the granularity of the data that is presented in the chart.
2. It will set the parameters in the service. Thus data in the service can be filtered based on the controls set by menu.

A4.2.3 Chart Component

1. Chart component contains d3 specific codes related to a particular chart.
2. It will subscribe to a observable in the service to get that chart parameters like column name and data keys present.
3. Chart component will get data by subscribing to a observable in the service. If there is any change in the data present service a callback function in chart will update the chart data.

A4.2.4 NodeJS Backend

1. NodeJS backend has APIs which fetches data from the MySQL database. APIs deal with executing SQL queries and presenting data in a format required by frontend.
2. Data is retrieved and sent to angular in JSON format.
3. These APIs are called by angular services to get the data that is required for different charts.

A5 Adding Maps to the project

- **Components Used**
  - app/Operational_Dashboard/Charts/map
  - Map Info - app/Operational_Dashboard/Menu/map-info
- **Steps to add a Map in Project**

- Read the topojson file (getMap())
- Extract the features - topojson.feature(topology, object)
  - Returns the GeoJSON Feature or FeatureCollection for the specified object in the given topology. If the specified object is a string, it is treated as topology.objects[object]. Then, if the object is a GeometryCollection, a FeatureCollection is returned, and each geometry in the collection is mapped to a Feature. Otherwise, a Feature is returned. The returned feature is a shallow copy of the source object: they may share identifiers, bounding boxes, properties and coordinates.
- Create projection - d3.geoMercator();
  - As the size of actual map will be larger, we will create projection of the map so that it will fit into the desired area.
- Creating d3 path using the projection created - d3.geoPath(projection);
  - D3 path represents the line in the Map. It is used to draw the boundary of the map.
- Usind D3 plot the map -
  - this.svg.selectAll(".country") // country is the class of a particular path
  - .data(state.features) // features extracted
  - .enter()
  - .append("path") //path created
  - ...

A6 Reusing the existing charts

- Reusing Stacked-Bar Chart
  - Structure of Stacked-Bar Chart:
    - Input: ChartService , Whenever the stacked bar-chart component is used ChartService is passed to it as input.
    - For Example: <**app-stacked-bar-chart [chartService]="menuService"**> </**app-stacked-bar-chart**>

    - Subscriptions: Stacked Bar Chart has subscribed to **getParameterListener()** and **getDataListener()** methods of service which is passed as input to it.
    - **getParameterListener()** informs the stacked-bar chart component when there is change in parameters such as granularity, month, normalisation etc.
    - **getDataListener()** updates data needed for chart when any parameter in changed.
- Reusing the Stacked-bar Chart:
    - Create the service which will have required methods according to the functionality.
    - Let us take an example of Stacked Bar Chart in Expense field.
    - Here we have created a service **expense-count-district.service.ts** which has extended to Service **patient-count.service.ts**.
    - **patient-count.service.ts** contains all the methods which are required to update the data and parameter.
    - In **expense-count-district.service.ts** we have initialize data specific to expense field.
    - Now create an instance of this service into the component where you want to add stacked bar chart.
    - For ex : **public districtExpenseService: ExpenseCountDistrictService** and initialize the service with required data. (we have created expense service and initialised in district-expense-main-menu component)
    - After initializing pass the service instance to stacked bar chart component like <**app-stacked-bar-chart [chartService]="menuService"**> </**app-stacked-bar-chart**>
    - In our case we have passed the service instance to menu-component and from menu component we have called stacked bar chart component.

A7 Setting up the environment

Following steps are tested on system running Ubuntu 18.04 LTS

1. Install NodeJS version 12.14.0

```
$ apt upgrade
$ apt install curl
$ curl -o- https://raw.githubusercontent.com/creationix/nvm/↩
   v0.33.11/install.sh | bash
$ nvm install 12.14.0
```

2. Install Angular CLI version 8.3.22.

```
$ npm install -g @angular/cli:8.3.22
```

3. Clone the repository.

```
$ git clone https://github.com/VipinRaiP/DMHP
$ cd DMHP
```

4. Install dependencies.

```
$ npm install
```

5. Start the angular server.

```
$ ng serve
```

6. Start node server

```
$ node server.js
```

7. App can be accessed at 'localhost:4200' and backend is running at 'localhost:3000'.

```
username : dmhp
password : dmhp@2020
```

A8 Challenges faced

1. **Creating a MySQL database from CSV files**

   Data was given in the **CSV** format, it was then converted to **SQL**. Tables can be found in **DMH** schema. Database has following table

   - **Clinical_data :** It contains information about patient count for different cases like alcohol, suicide, SMD and others.
   - **District_Expense :** It has details about the money spent for various programmes and the salary paid for health workers.
   - **Districts_Training :** Training data was not available in the given csv files. We created a table and filled with dummy data for demonstration purpose.
   - **Districts :** It maps the district id with district name. It has other information like hospital name and sanctioned bed.
   - **Taluka :** It maps the taluka id with taluka name. It has taluka hospital name along with number of sanctioned beds.

2. **Added Population column to district table**

   Dashboard has an option to visualize data normalised with respect to district population. Population details were not present in the district table. Population data was obtained from an external source for the purpose of demonstration and added as a column in district table.

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

#	DistrictId	StateId	District	HospitalName	SancBeds	RegDate	CreatedDate	IsActive	CreatedBy	Population
1	1	17	Bagalkote	BAGALKOTE DISTRICT	100	2017-04-06	2017-11-06	1	admin user	1890826
2	2	17	Bangalore Rural	BANGALORE RURAL	100	2017-04-06	2017-11-06	1	admin user	987257
3	3	17	Bangalore Urban	BANGALORE URBAN	100	2017-04-06	2017-11-06	1	admin user	9588910
4	12	17	Belgaum	BELGAUM DISTRICT	100	2017-04-06	2017-11-06	1	admin user	4778439
5	13	17	Bellary	BELLARY DISTRICT	100	2017-04-06	2017-11-06	1	admin user	2532383
6	15	17	Bidar	BIDAR DISTRICT	100	2017-04-06	2017-11-06	1	admin user	1700018
7	16	17	Bijapur	BIJAPUR DISTRICT	100	2017-04-06	2017-11-06	1	admin user	2175102
8	17	17	Chamrajnagar	CHAMARAJNAGAR DISTRICT	100	2017-04-06	2017-11-06	1	admin user	1020962
9	18	17	Chikkaballapur	CHIKKABALLAPUR DISTRICT	100	2017-04-06	2017-11-06	1	admin user	1254377
10	19	17	Chikmagalur	CHICKMAGALUR DISTRICT	100	2017-04-06	2017-11-06	1	admin user	1137753
11	20	17	Chitradurga	CHITRADURGA DISTRICT	100	2017-04-06	2017-11-06	1	admin user	1660378
12	21	17	Dakshina Kan...	DAKSHIN KANNADA DISTRI...	100	2017-04-06	2017-11-06	1	admin user	2083625

Figure 3: District table

3. **Map format conversion from kml to topojson**

   Karnataka State map and taluka maps were obtained from **Karnataka Geographic Information System**. Data was present in the **kml** format which was then converted to **topojson**.

   Link : https://kgis.ksrsac.in/kgis/downloads.aspx

   Topojson files are stored in project folder **DMHP/src/assets**.

4. **Taluka name mismatch in map and database :**

   Taluka names provided in the database was not matching with names defined in the topojson file. Due to this issue data for that taluka was not getting represented. This issue is kept open and can be rectified by updating the taluka names appropriately.

Figure 4: Topojson defines as Hungund taluka

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

#	TalukaId	DistrictId	StateId	Taluka	HospitalName	SancBeds	RegDate	CreatedDate	IsActive	CreatedBy
1	101	1	17	Badami	Badami Taluka	50	2016-05-05	2016-05-05	1	admin
2	102	1	17	Bilagi	Bilagi Taluka	50	2016-05-05	2016-05-05	1	admin
3	103	1	17	Hunagund	Hunagund Taluka	50	2016-05-05	2016-05-05	1	admin
4	104	1	17	Jamakhandi	Jamakhandi Taluka	50	2016-05-05	2016-05-05	1	admin
5	105	1	17	Mudhole	Mudhole Taluka	50	2016-05-05	2016-05-05	1	admin
6	106	1	17	BagalkoteTaluka	Bagalkote Taluka	50	2016-05-05	2016-05-05	1	admin
7	107	2	17	Devanahalli	Devanahalli Taluka	50	2016-05-05	2016-05-05	1	admin
8	108	2	17	Doddaballapur	Doddaballapur Taluka	50	2016-05-05	2016-05-05	1	admin
9	109	2	17	Hosakote	Hosakote Taluka	50	2016-05-05	2016-05-05	1	admin
10	110	2	17	Nelamanagala	Nelamanagala Taluka	50	2016-05-05	2016-05-05	1	admin
11	111	2	17	BangaloreRur...	BangaloreRural Tal...	50	2016-05-05	2016-05-05	1	admin
12	112	3	17	Anekal	Anekal Taluka	50	2016-05-05	2016-05-05	1	admin

Figure 5: Database name is Hunagund

Figure 6: Data not represented due to mismatch

A9 Future Scope

As the project is developed keeping reusability of components in mind, makes it very easy to use the same charts and services to visualize new data just by changing some service parameters. In future following aspects can be added to the project:

1. **Integrating with machine learning**
   Using machine learning models to find trends and correlations in the data. This will help to analyze and predict future trends and will also help to prepare and take appropriate actions well in advance

2. **Adding more charts and dashboards for in-depth visualizations**
   Charts like ScatterPlot, Heatmap, BubbleChart, Treemap, WordCloud, Network-Flow, etc. can be added.

3. **Extending this project to different diseases/outbreaks or using this project for different hospital-cases or healthcare-projects**
   This project can be easily extended to other diseases like heart, cancer, lungs, COVID etc. or just like DMHP different healthcare projects like Integrated Disease Surveillance Programme (IDSP), Guinea Worm Eradication Programme (GWEP), etc. or even visualizing hospital-specific data like AIIMS hospital, Safdarjung Hospital, etc.

A10 Code Structure

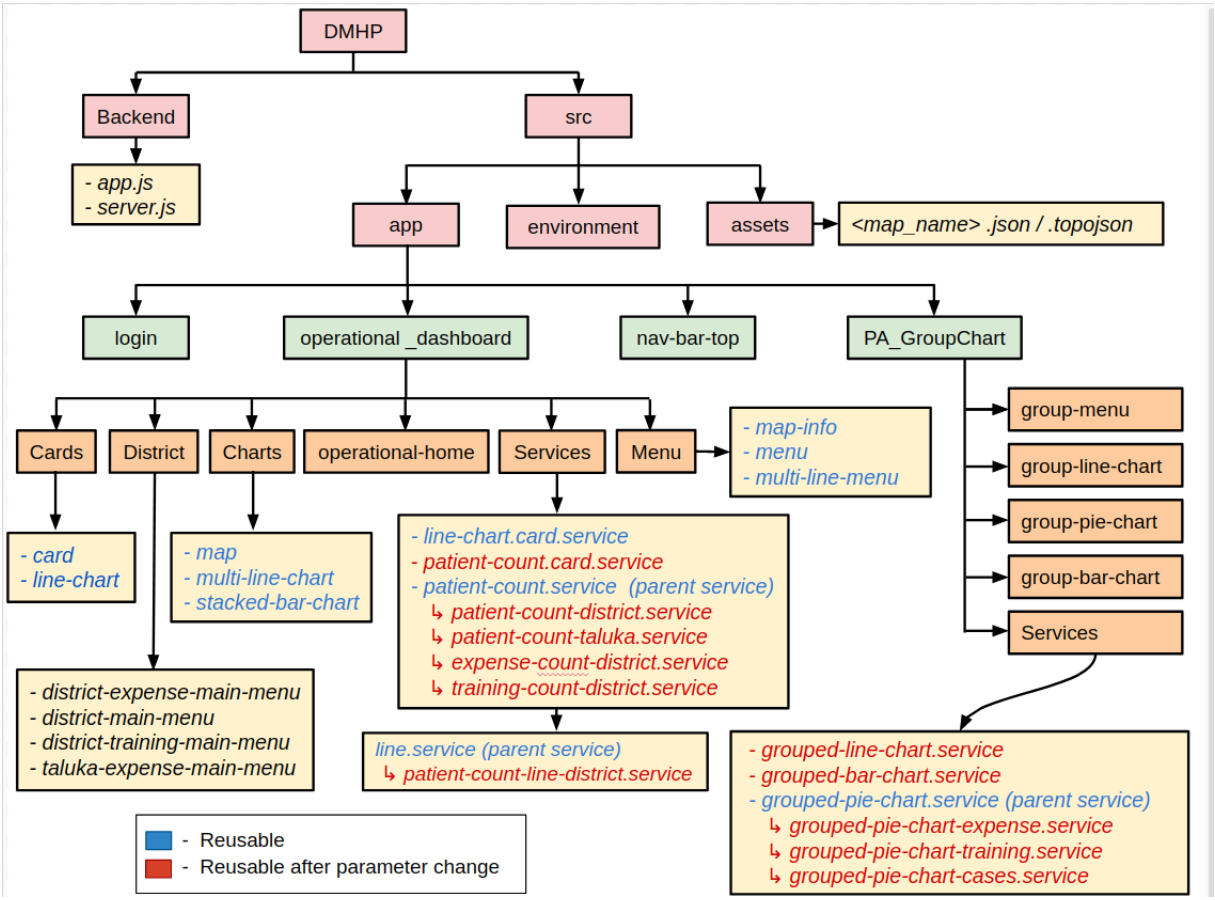


Figure 7: Code Structure

A11 Screenshots

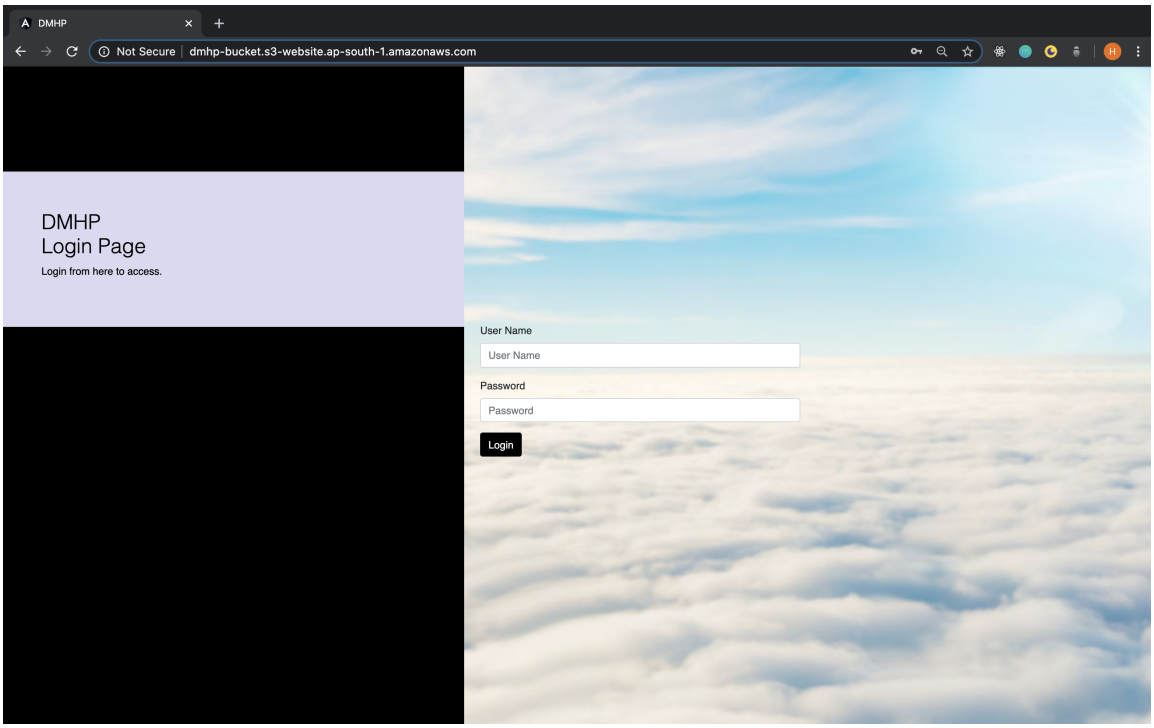


Figure 8: Login Page

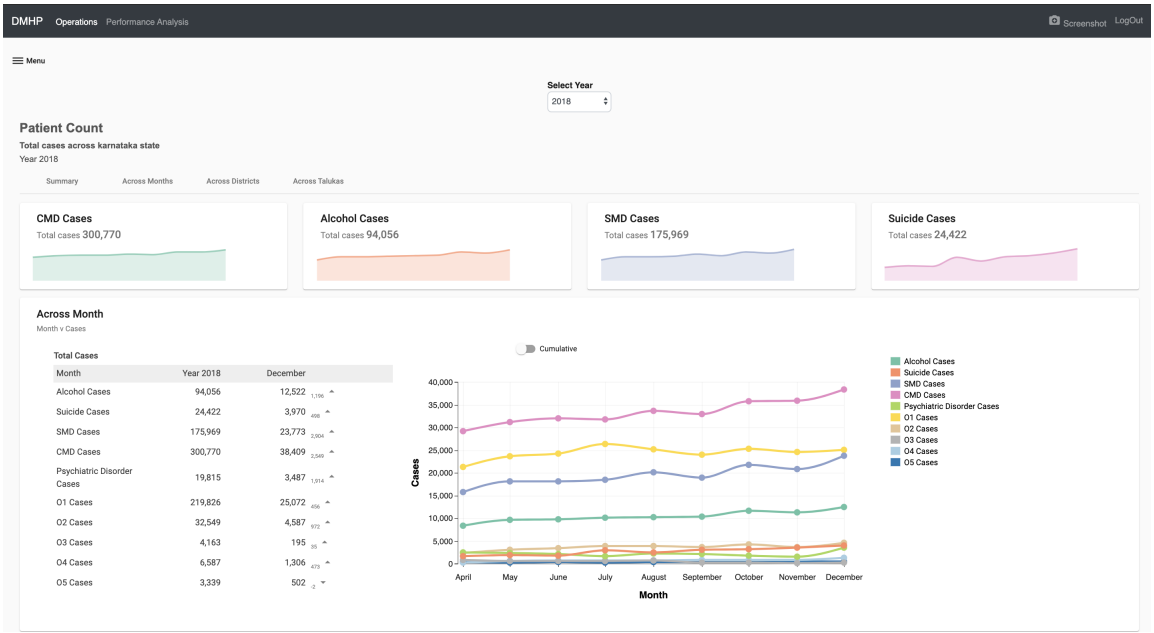


Figure 9: Operations Dashboard

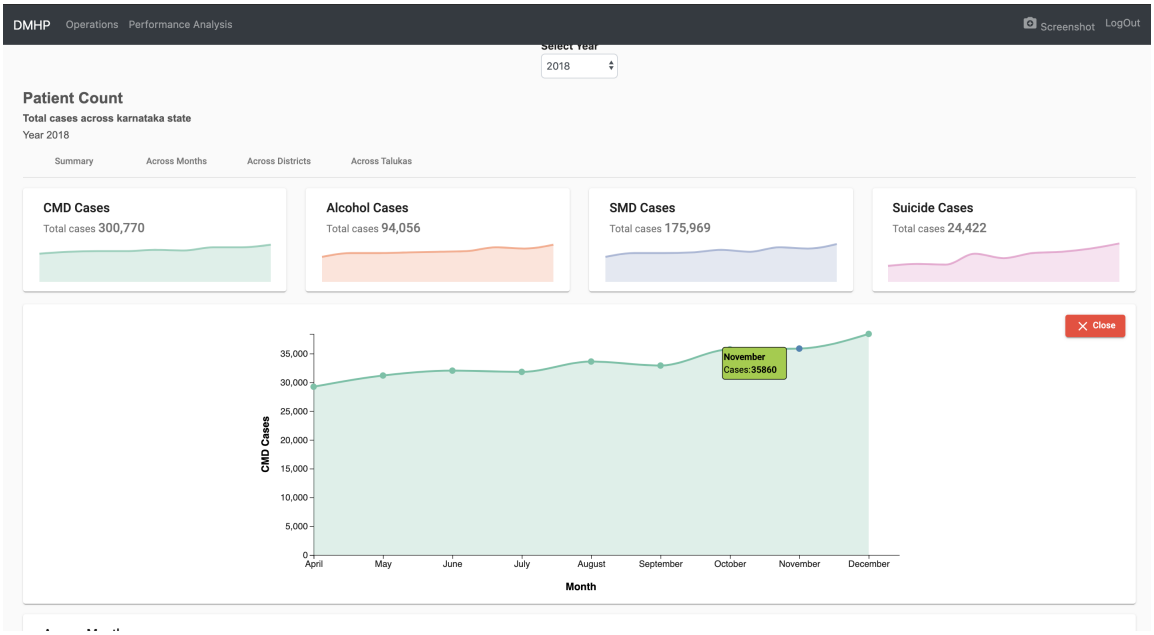


Figure 10: Summary Line chart

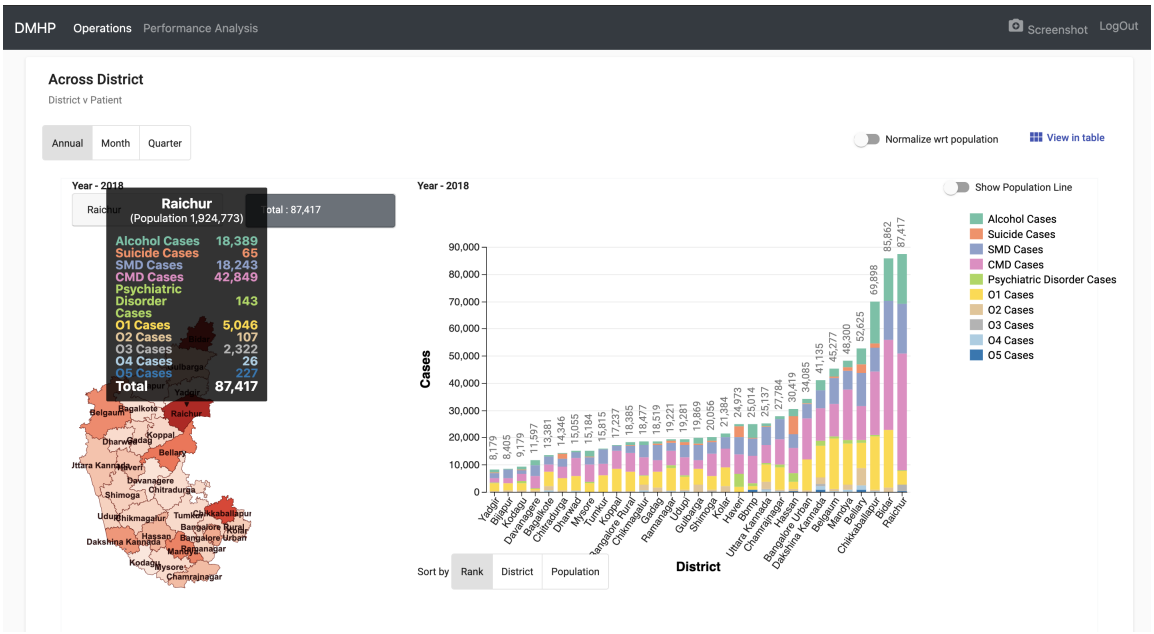


Figure 11: District Wise Patient Count

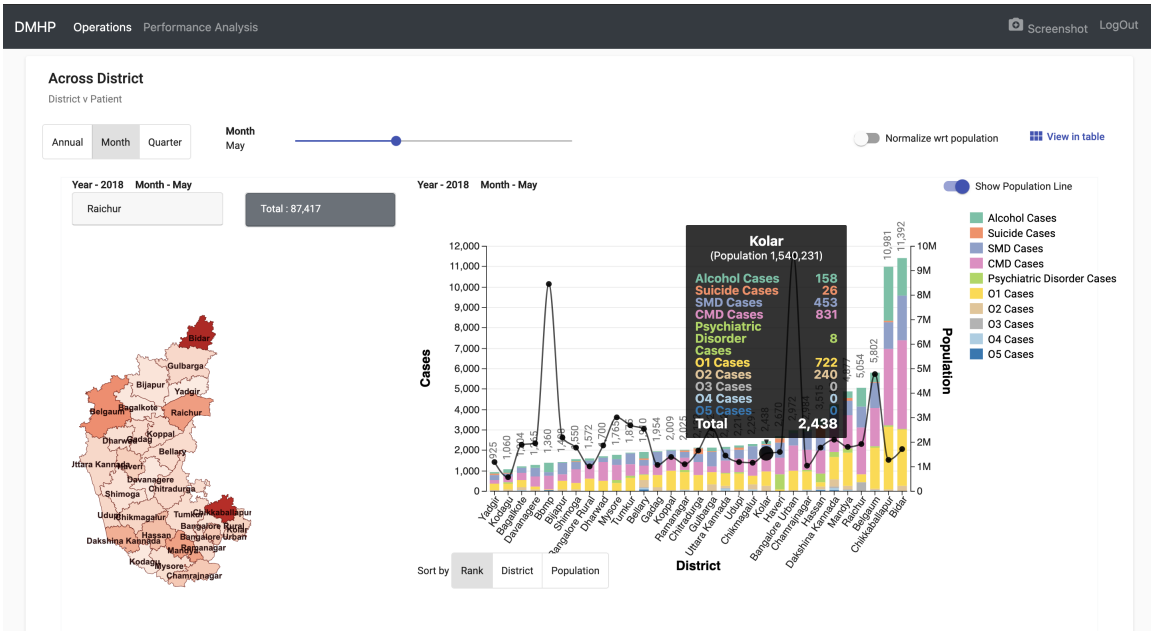


Figure 12: District Wise Patient Count with Population line

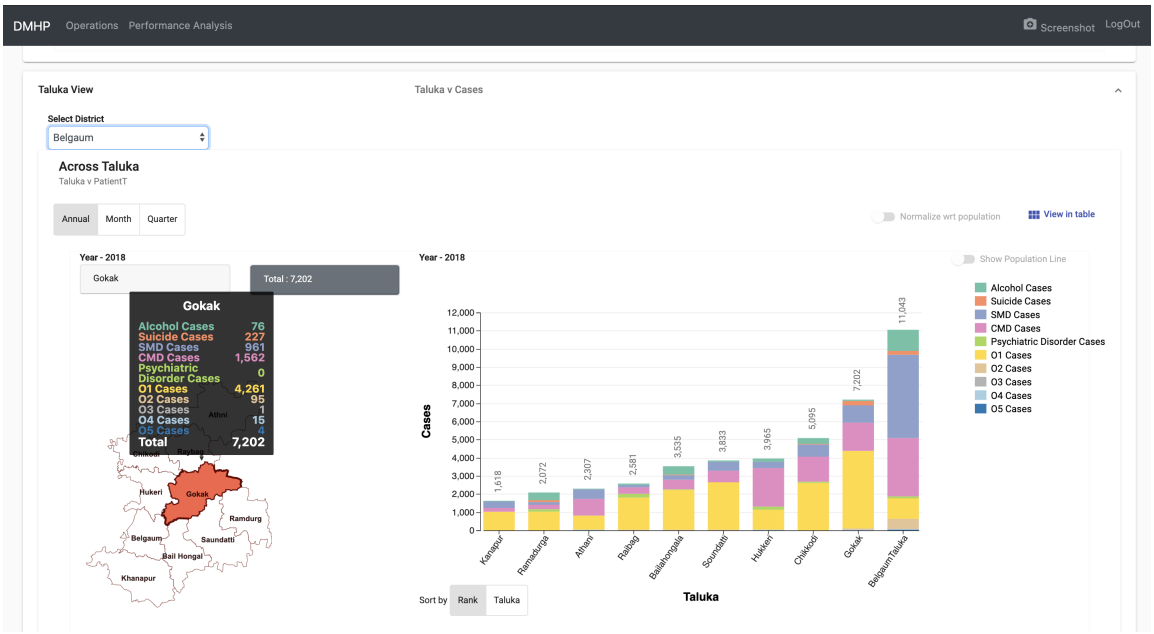
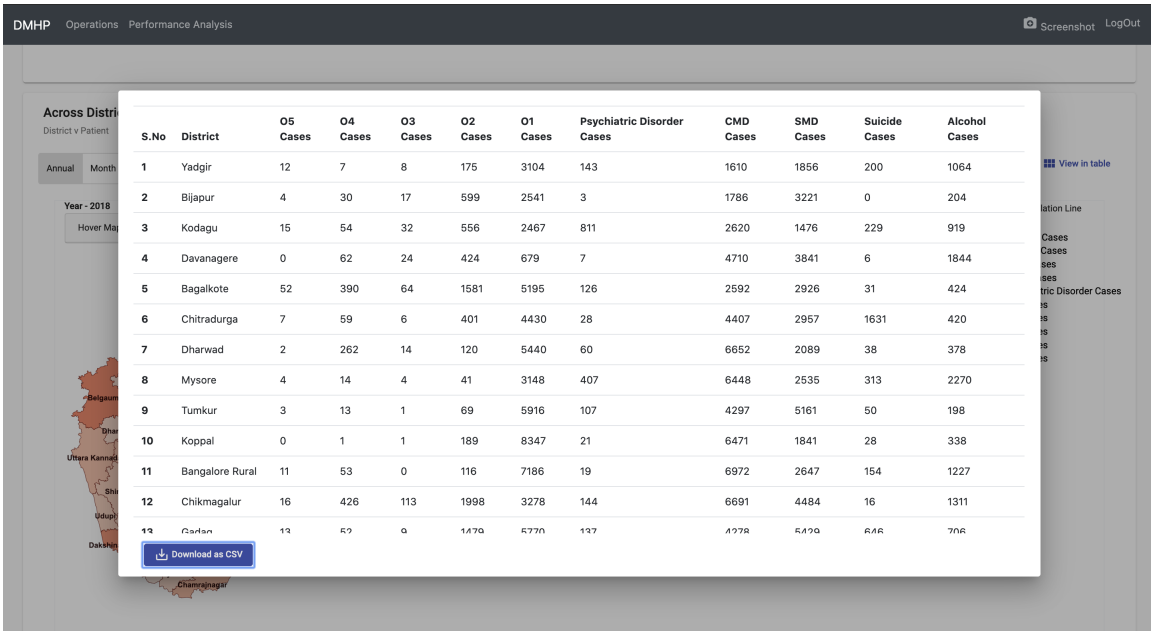


Figure 13: Taluka Wise Patient Count



S.No	District	O5 Cases	O4 Cases	O3 Cases	O2 Cases	O1 Cases	Psychiatric Disorder Cases	CMD Cases	SMD Cases	Suicide Cases	Alcohol Cases
1	Yadgir	12	7	8	175	3104	143	1610	1856	200	1064
2	Bijapur	4	30	17	599	2541	3	1786	3221	0	204
3	Kodagu	15	54	32	556	2467	811	2620	1476	229	919
4	Davanagere	0	62	24	424	679	7	4710	3841	6	1844
5	Bagalkote	52	390	64	1581	5195	126	2592	2926	31	424
6	Chitradurga	7	59	6	401	4430	28	4407	2957	1631	420
7	Dharwad	2	262	14	120	5440	60	6652	2089	38	378
8	Mysore	4	14	4	41	3148	407	6448	2535	313	2270
9	Tumkur	3	13	1	69	5916	107	4297	5161	50	198
10	Koppal	0	1	1	189	8347	21	6471	1841	28	338
11	Bangalore Rural	11	53	0	116	7186	19	6972	2647	154	1227
12	Chikmagalur	16	426	113	1998	3278	144	6691	4484	16	1311
13	Gadag	13	52	9	1479	5770	137	4278	5429	646	706

Figure 14: Table view of Patient Count

Figure 15: District Wise Expense Data

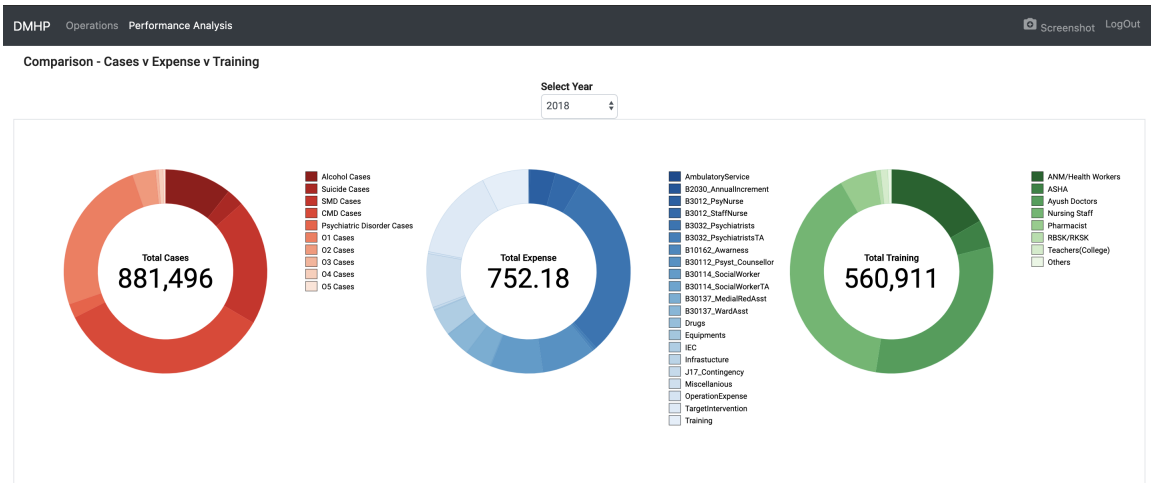


Figure 16: Performance Analysis Dashboard - Pie Chart

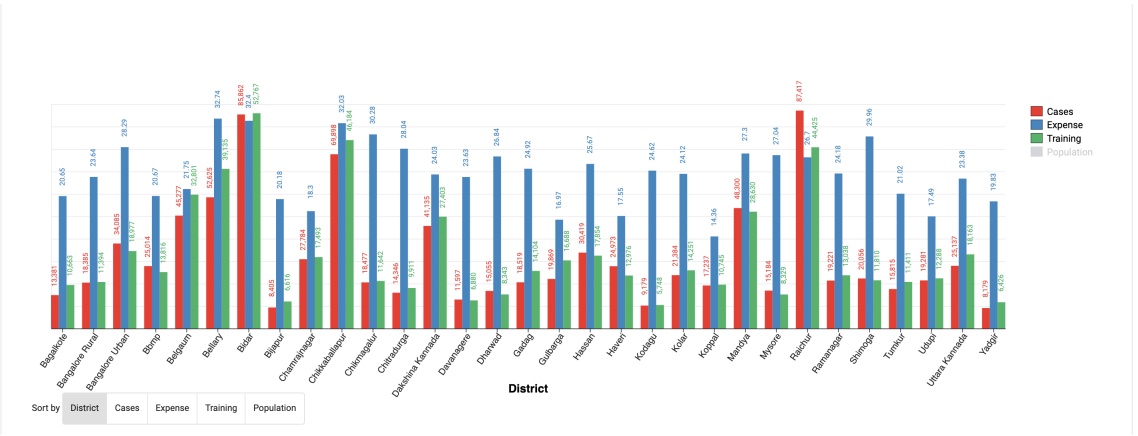


Figure 17: Performance Analysis Dashboard - Bar Chart

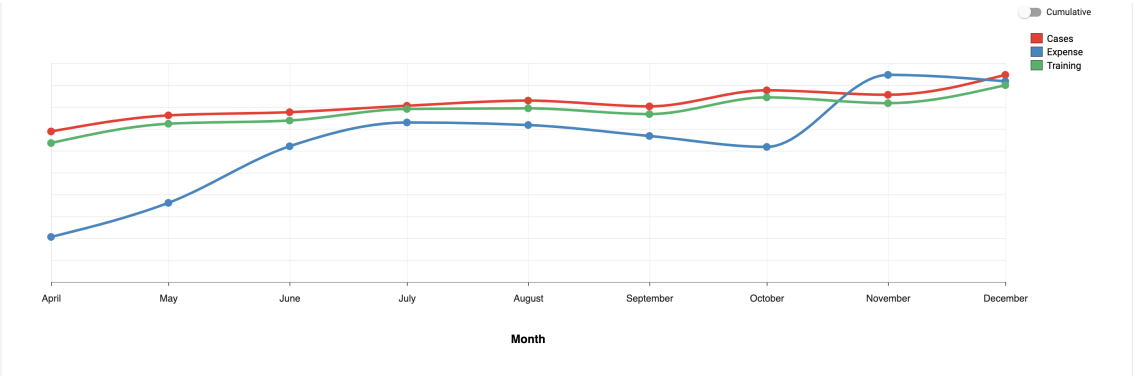


Figure 18: Performance Analysis Dashboard - Multi-Line Chart

References

- Mastering data visualization in D3.js
- D3 Examples
- D3 Popular Blocks
- Karnataka Geographic Information System
- Udemy Course : Angular - The Complete Guide
- Udemy Course : Mastering data visualization in D3.js
- Udemy Course : Angular NodeJS - The MEAN Stack Guide [2020 Edition]
- Making a world map with d3, topojson, and a csv