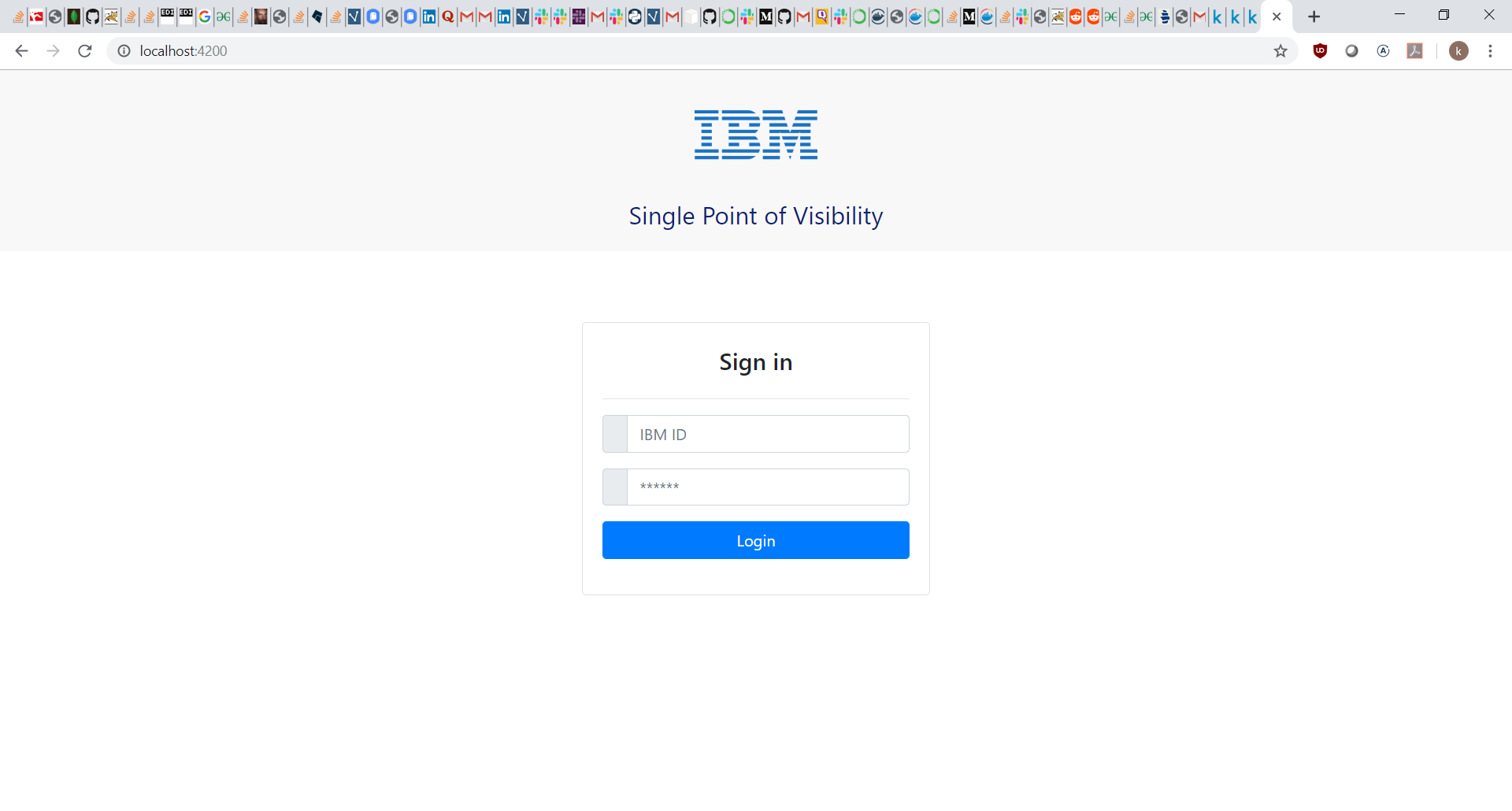
Single Point of Visibility

UI - Documentation

Kshiteej Gilda

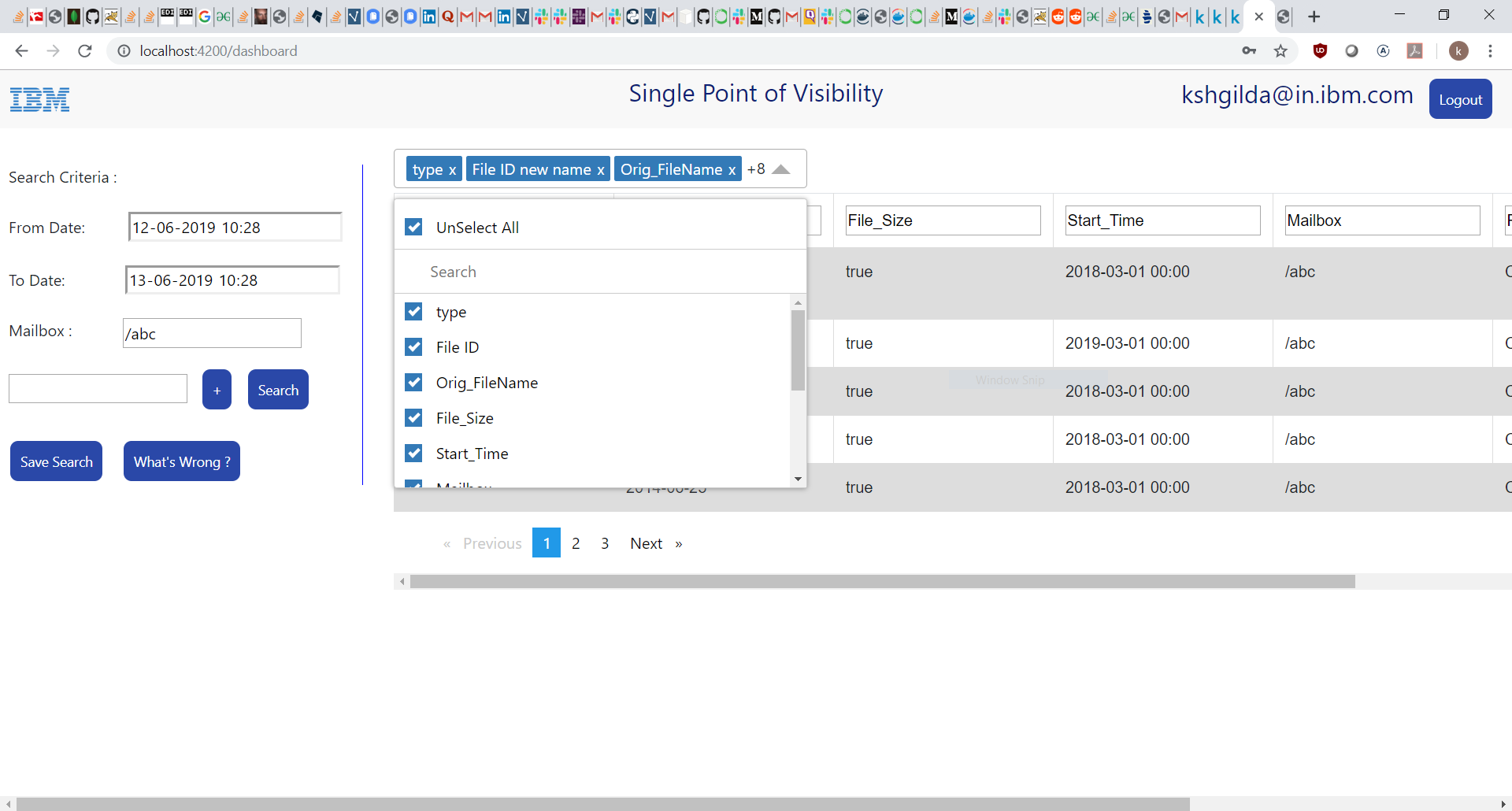
**Features :**

1. Login Page :



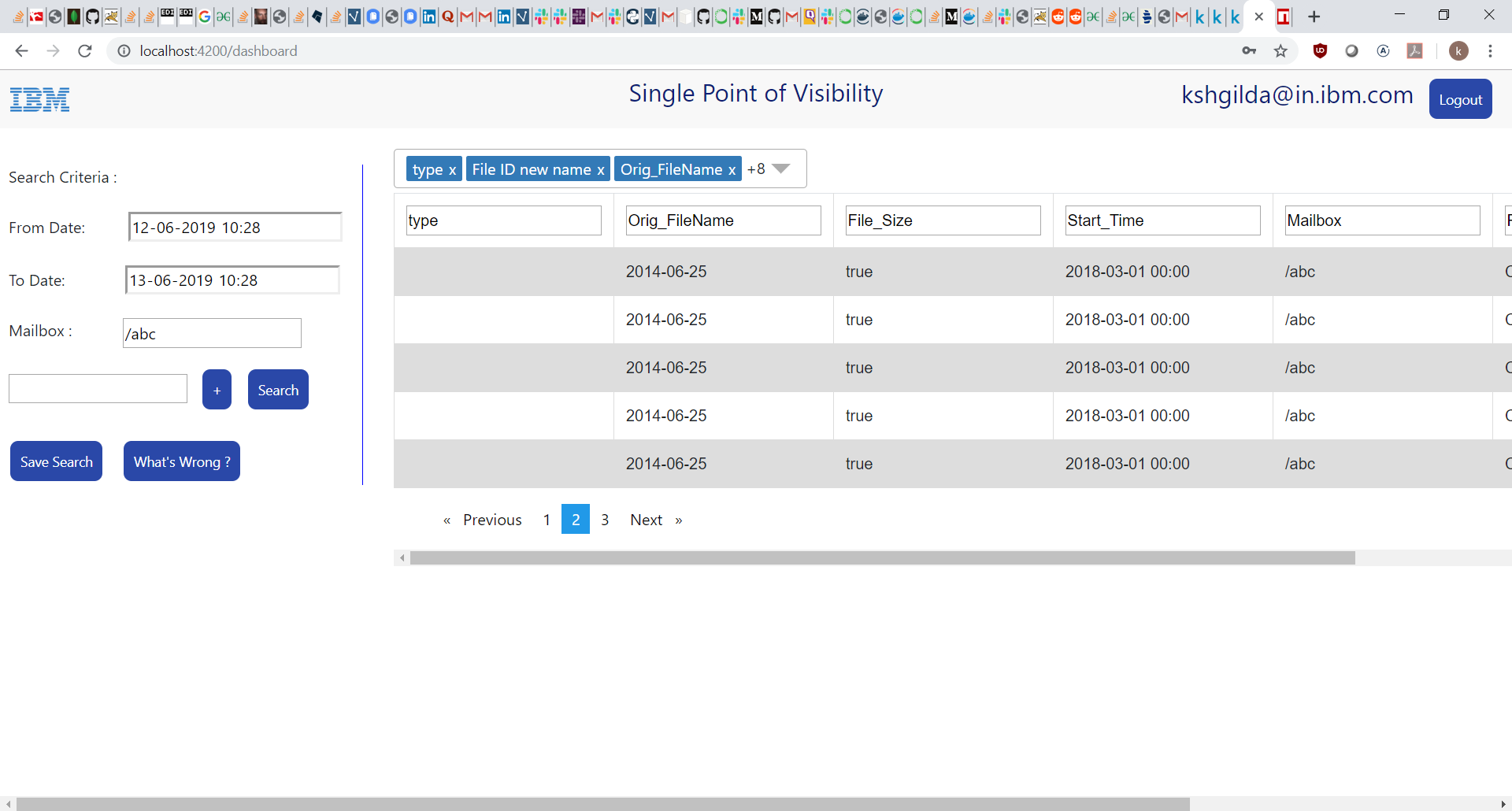
The Login Page is a separate component in Angular. Upon clicking the Login button the application sends a post request to the NodeJS login API in the background with the IBM ID and password as its parameters. The login API uses the ‘ibm\_bluepages’ package in NodeJS to perform authentication.

1. Select Columns to Display :



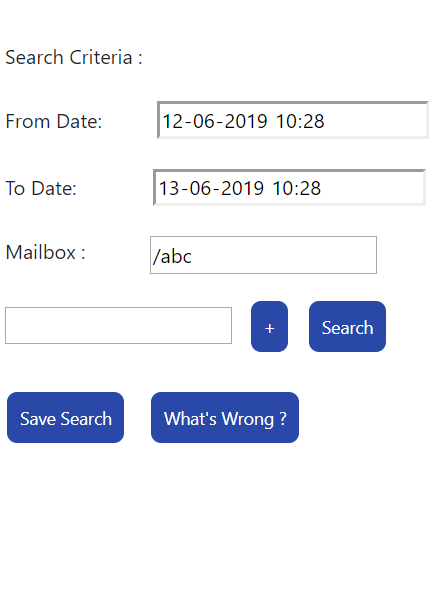
The user can select the number of columns to be displayed in the table by searching for the column name and selecting it. The user also has to the option of selecting all columns or unselecting all columns. The selected columns will be displayed on the top. This feature has been implemented using the ‘ng-multiselect-dropdown’ package for Angular.

1. Pagination :



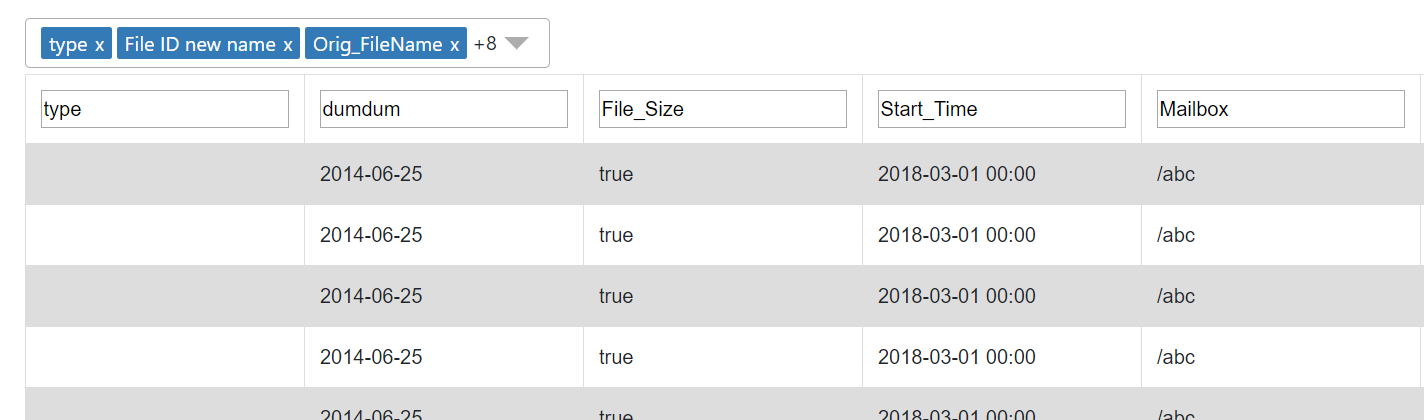
The applications supports pagination. The user can navigate between different pages of the application. There are currently 5 rows displayed per page. The number of rows displayed can be changed by changing the items per page property in paginate pipe in search-criteria.component.html

1. Add Tag :



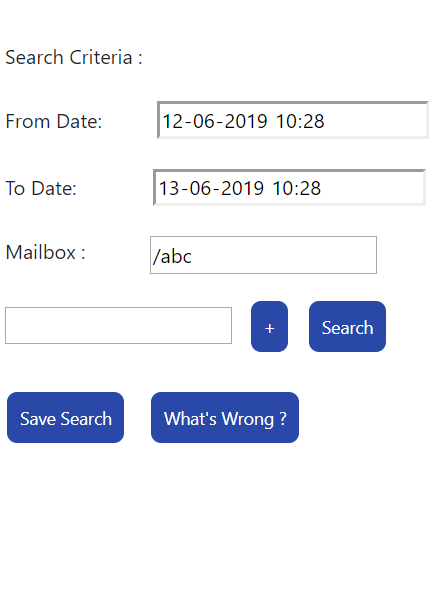
The Search criteria tags can be dynamically added by searching and selecting the tag in the dropdown. The tag values then be entered in their respective field. The displaytags array is being dynamically populated upon clicking on the + button after selecting the tag. The displaytags array is being dynamically rendered in the search criteria.

1. Editable Column headers :



The user can edit the name of the column headers. The same updated name would be reflected everywhere, ie) in the tags and the tag name. The column mapping is being stored in the user profile in CouchDB. The Column mapping is basically a map between the original column name and the edited column name. The displayed column names are the edited column names.

1. Save Search



The user can save the search in the ‘spv’ database of CouchDB. The save search saves the search parameters like the tag values and the column mapping which stores the map between the original column names and edited column names. A sample user profile JSON Document structure is as follows :

{

"\_id": "kshgilda@in.ibm.com",

"\_rev": "44-8f7f53d51d22b4177b2ab383e2edc490",

"type": "user\_profile",

"File\_ID": "",

"Orig\_FileName": "",

"File\_Size": "",

"Start\_Time": "",

"Mailbox": "/abc",

"Payload": "",

"Event\_ID": "",

"Parent\_Document\_ID": "",

"Child\_Document\_ID": "",

"Number\_Of\_Columns": [

{

"item\_id": 1,

"item\_text": "type"

},

{

"item\_id": 2,

"item\_text": "File ID new name"

},

{

"item\_id": 3,

"item\_text": "Orig\_FileName"

},

{

"item\_id": 4,

"item\_text": "File\_Size"

},

{

"item\_id": 5,

"item\_text": "Start\_Time"

},

{

"item\_id": 6,

"item\_text": "Mailbox"

},

{

"item\_id": 7,

"item\_text": "Payload"

},

{

"item\_id": 8,

"item\_text": "Event\_ID"

},

{

"item\_id": 9,

"item\_text": "Parent\_Document\_ID"

},

{

"item\_id": 10,

"item\_text": "events"

},

{

"item\_id": 11,

"item\_text": "\_\_typename"

}

],

"Column\_Mapping": [

{

"orig\_tag": "type",

"new\_tag": "type"

},

{

"orig\_tag": "File\_ID",

"new\_tag": "File ID"

},

{

"orig\_tag": "Orig\_FileName",

"new\_tag": "Orig\_FileName"

},

{

"orig\_tag": "File\_Size",

"new\_tag": "File\_Size"

},

{

"orig\_tag": "Start\_Time",

"new\_tag": "Start\_Time"

},

{

"orig\_tag": "Mailbox",

"new\_tag": "Mailbox"

},

{

"orig\_tag": "Payload",

"new\_tag": "Payload"

},

{

"orig\_tag": "Event\_ID",

"new\_tag": "Event\_ID"

},

{

"orig\_tag": "Parent\_Document\_ID",

"new\_tag": "Parent\_Document\_ID"

},

{

"orig\_tag": "events",

"new\_tag": "events"

},

{

"orig\_tag": "\_\_typename",

"new\_tag": "\_\_typename"

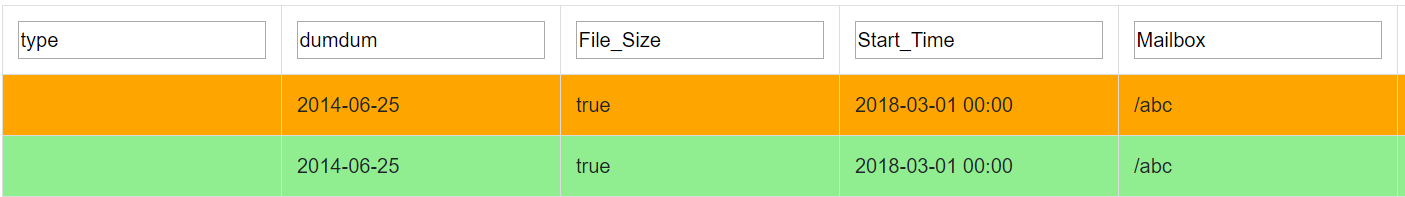
}

]

}

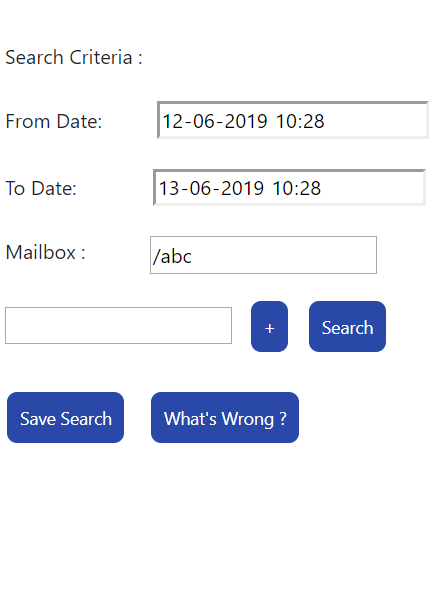
The ‘Number\_Of\_Columns’ field stores the columns which have been selected for display in the dropdown. When the user clicks on Save Search the application sends a POST request to the NodeJS API for saving search. NodeJS is using the Node-CouchDB driver to connect to the CouchDB instance on the server.

1. Display What’s Wrong in the table :



Upon Clicking the What’s Wrong button, the table would be automatically coloured as – Light Green for a row whose all events have succeeded and Amber for a row whose any one events has failed. The application uses the id of the row and the associated event status to colour a row as Light Green or Amber.

1. Searching for Data



Upon Clicking on the Search Button the application hits the GraphQL API hosted on NodeJS. The application can query for the data hosted on the server by specifying the various search parameters. A sample GraphQL Query is given below :

"operationName": null,

"variables": {

"Mailbox": "",

"File\_ID": "",

"Parent\_Document\_ID": "",

"Orig\_Filename" : "",

"File\_Size" : "",

"Start\_Time" : "",

"Payload" : ""

},

"query": "query ($File\_ID: String, $Orig\_FileName: String,$Mailbox: String) {\n file(File\_ID: $File\_ID, Orig\_FileName: $Orig\_FileName,Mailbox:$Mailbox) {\n type\n File\_ID\n Orig\_FileName\n File\_Size\n Start\_Time\n Mailbox\n Payload\n Event\_ID\n Parent\_Document\_ID\n events {\n File\_ID\n Event\_ID\n End\_Point\n Timestamp\n End\_port\n Protocol\n WFID\n Start\_time\n End\_Time\n Adapter\_Type\n Adapter\_Name\n Session\_ID\n Principal\n Credential\_Type\n State\n Is\_Success\n Document\_ID\n Remote\_FileName\n Entity\_type\n Producer\_Name\n Consumer\_Name\n Layer\_Type\n Layer\_File\_Name\n Bytes\_Transferred\n \_\_typename\n }\n \_\_typename\n }\n}\n"

In the query we can specify what fields we want to be returned as response. In the variables we can specify the values for the different fields based on which we want to search.