ChairVisE

Code Repository URL:
https://github.com/CS3219-SEM1/chairvise-project-2018-team-14

Student Name	Cai Di	Jeremy Yew Ern	Tan Jun An	Tan Li Hao
Matriculation Number	A0085159W	A0156262H	A0139052L	A0140023E

Introduction

The ChairVise app enables conference program Chairs of academic and industry conferences to make statistical insights about their conferences. It provides a simple, convenient way for Chairs to produce aesthetically pleasing and informative and visualizations which summarize relevant data about conference submissions, reviews, and authors.

The insights produced by these visualizations will empower conference organizers to make more informed decisions about conference programming and outreach; researchers and industry stakeholders will also be able to learn more about the research landscape of different domains.

a) ChairVisE 1.0 Requirements

F1 ChairVisE 1.0 - web app that visualizes conference submission statistics	**Priority**	**Sprint**
F1.1 The application should have an interface to allow a user to upload files.	H	-
F1.2 The application should be able to parse CSV files.	H	-
F1.3 Analyze author data		
F1.2.1 The application should count the authors with the most submissions	H	-
F1.2.2 The application should count the countries with the most submissions	H	-
F1.2.3 The application should count the organizations(affiliations) with most submissions	H	-

F1.4 Analyze submission data		-
F1.4.1 The application should calculate the total acceptance rate (accepted submissions / total submissions).	H	-
F1.4.2 The application should calculate the acceptance rate per track (accepted submissions per track / total submissions per track).	H	-
F1.4.3 The application should compare the acceptance rate for full papers and short papers and compare it to previous years.	H	-
F1.4.4 The application should analyze the last edited datetimes as as a time series.	H	-
F1.4.5 The application should analyze the submission datetimes as as a time series.	H	-
F1.4.6 The application should compare the submission datetimes time series and the last edited datetimes time series and provide an analysis about it.	H	-
F1.4.7 The application should count the number of times each keyword appear for all submissions, accepted submissions and rejected submissions.	H	-
F1.4.8 The application should count the number of times each keyword appear per track.	H	-
F1.4.9 The application should count the top accepted authors.	H	
F1.4.10 The application should count the top accepted authors per track.	H	-
F1.5 Analyze review data		-
F1.5.1 The application should calculate the weighted score per submission (sum all submission(score * confidence) / total confidence).	H	-
F1.5.2 The application should calculate the weighted recommendations per submission (sum all submission(is recommended * confidence) / total confidence).	H	-
F1.5.3 The application should calculate the average confidence per submission (sum all submission (confidence) / total confidence).	H	-
F1.5.4 The application should calculate the distribution of weighted scores (for each score falling into a range group and count them).	H	-
F1.5.5 The application should calculate the distribution of weighted recommendations (for each recommendation falling into a range group and count them).	H	-
F1.5.6 The application should calculate the mean weighted scores, mean weighted recommendations and the average	H	-

confidence over all submissions.		
F1.6 Render analyzed author data as charts		-
F1.6.1 The application should render the authors with the most submissions as a bar chart.	H	-
F1.6.2 The application should render the countries with the most submissions as a pie chart, allowing it to be changed it to a bar chart.	H	-
F1.6.3 The application should render the organizations(affiliations) with the most submissions as a bar chart, allowing it to be changed it to a pie chart.	H	-
F1.7 Render analyzed submission data as charts		-
F1.7.1 The application should render the submission datetimes time series and last edited datetimes time series in a line chart.	H	-
F1.7.2 The application should allow rendering the acceptance rate per track as a bar chart.	H	-
F1.7.3 The application should allow rendering the acceptance rate per track as a radar chart.	M	-
F1.7.4 The application should render the acceptance rates for full papers and short papers for as a line chart.	H	-
F1.7.5 The application should render the top accepted authors as a bar chart.	H	-
F1.7.6 The application should render the top accepted authors per track as a bar chart.	H	-
F1.7.7 The application should render the keywords for all submissions, keywords for accepted submissions, keywords for rejected submissions, keywords per track as word clouds	H	-
F1.8 Render analyzed review data as charts		-
F1.8.1 The application render the distribution of weighted scores per submission as a bar chart.	H	-
F1.8.2 The application render the distribution of weighted recommendations per submission as a bar chart.	H	-
F1.8.3 The application should show the mean weighted scores, mean weighted recommendations and the average confidence over all submissions in a table.	H	-
F1.9 The application should allow the user to generate a PDF document containing the currently rendered charts (and their captions) marked as ‘included’	M	-
NF1.10 The application should take no longer than 5 seconds to analyze and render the charts.	H	-

b) ChairVisE 1.0 Design Diagrams

c) ChairVisE 2.0 Requirements

F2 ChairVis 2.0 - web app that visualizes conference submission statistics	**Priority**	**Sprint**
F2.1 The application should allow the user to switch between tabs to view different sets of visualizations generated by different files.		
F2.1.1 The application should switch to the appropriate tab after a new file has been uploaded.	H	Week 10
F2.2 The application should have session management.		
F2.2.1 The application should allow a user to sign up and login.	H	Week 11
F2.2.2 The application should allow user to save data.	H	Week 12
F2.3 The application should save the analyzed data of the user's session, and load them on the next session.		
F 2.3.1 The application should signal to the user that the previous analyzed data is being loaded.	H	Week 11
F2.4 The application should allow the user to generate a PDF document containing all currently rendered charts (and their captions) marked as 'included' in one PDF document.		
F 2.4.1 The application should signal to the user that the document is being generated	H	Week 10
F 2.4.2 The application should notify the user on success or failure of the document generation.	H	Week 10
NF 2.4.3 The application should generate the PDF document in under 3 secs per visualization included.	M	Week 12
F2.5 The application should allow the user to map the columns of their uploaded csv files to the required headers that our application needs, regardless of how each their column are arranged.		
NF 2.5.1 The application should only allow uploading of files which have at least the required number of columns.	H	Week 12
NF 2.5.2 The application should only allow each required header to map to one distinct column.	H	Week 12
F2.6 For visualizations that require data from multiple files, they should be available once the necessary files have been submitted.		
F2.7 Render analyzed review data as charts		

F2.7.1 The application should display a word cloud of submission keywords, which may be filtered by track, and by whether the submission is accepted or rejected.	H	Week 12
F2.8 Render analyzed author and review data as charts		
F2.8.1 The application should render the authors with the most reviews as bar chart.	H	Week 11
F2.9 Render analyzed author and submission data as charts		
F2.9.1 The application should render the top organizations(affiliations) with the most accepted submissions as a bar chart or pie chart.	H	Week 10
F2.10 The application should notify the user on failure of data analysis of each submitted file.		
F2.11 The application should provide visualizations that are more meaningful and informative to users than in 1.0.		
F2.11.1 The application should allow user to select top 1-10 entities (e.g. author, country, organization) for each visualization that involves ranking.	M	Week 13
F2.11.2 The application should display axis labels for all relevant charts.	H	Week 12
NF2.12 The application should have basic web app security.		
NF2.12.1 The application should store passwords that are hashed and salted using libraries.	H	Week 11
NF2.12.2 The application should implement authentication through well-established libraries.	H	Week 11
NF2.12.3 The application should use OAuth 2.0 protocol for third party authorization.	H	Week 11
NF2.13 The application should be delivered continuously when pull requests are merged to master automatically	M	Week 10
NF2.14 The application should store data in a database to ensure consistency and durability.	H	Week 13

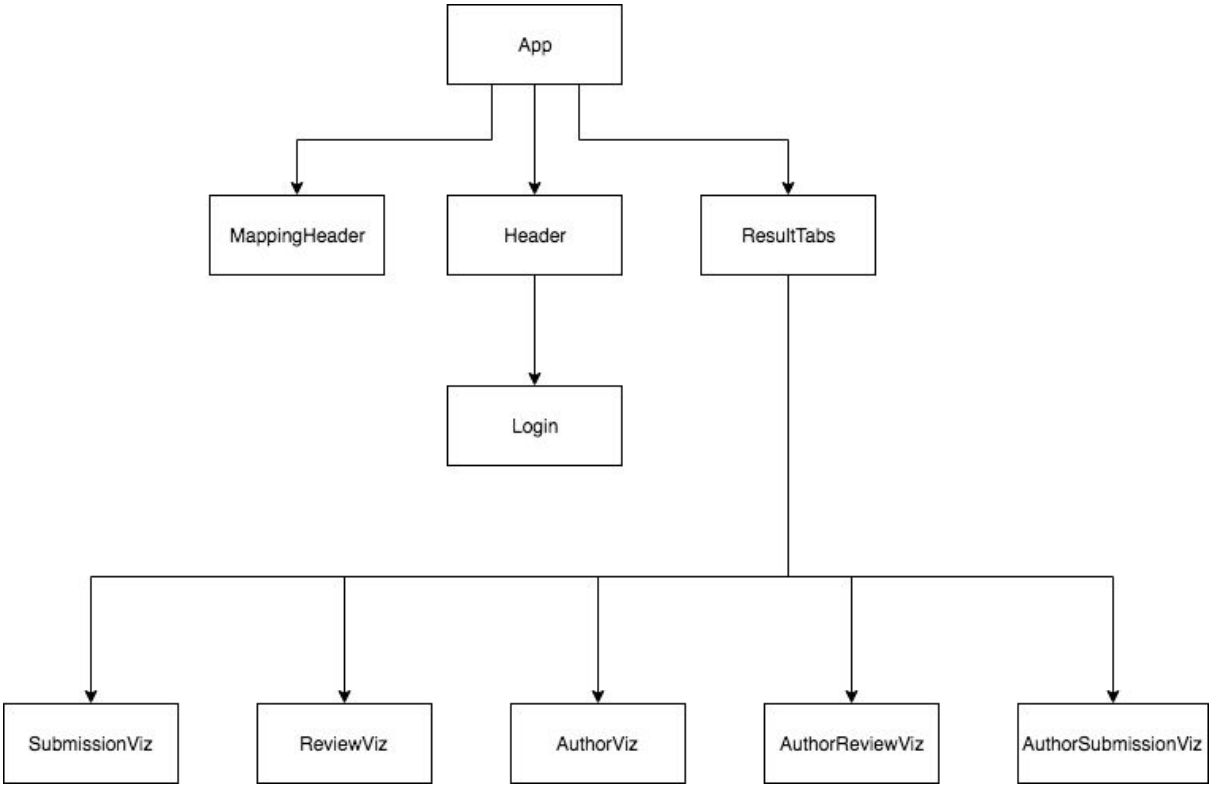
d) ChairVisE 2.0 Developer documentation

Introduction

The web app follows a client-server architecture. Each user is a client accessing the frontend of the application developed using Vue.js. The backend is developed using Django will run on server(s) serving the clients. In our case, we do not manage the servers on our own and instead use a PaaS (Heroku) that manage the servers for us.

The development process is a simple one where we create a pull request, wait for build to pass, merge, and the new web app will be automatically deployed.

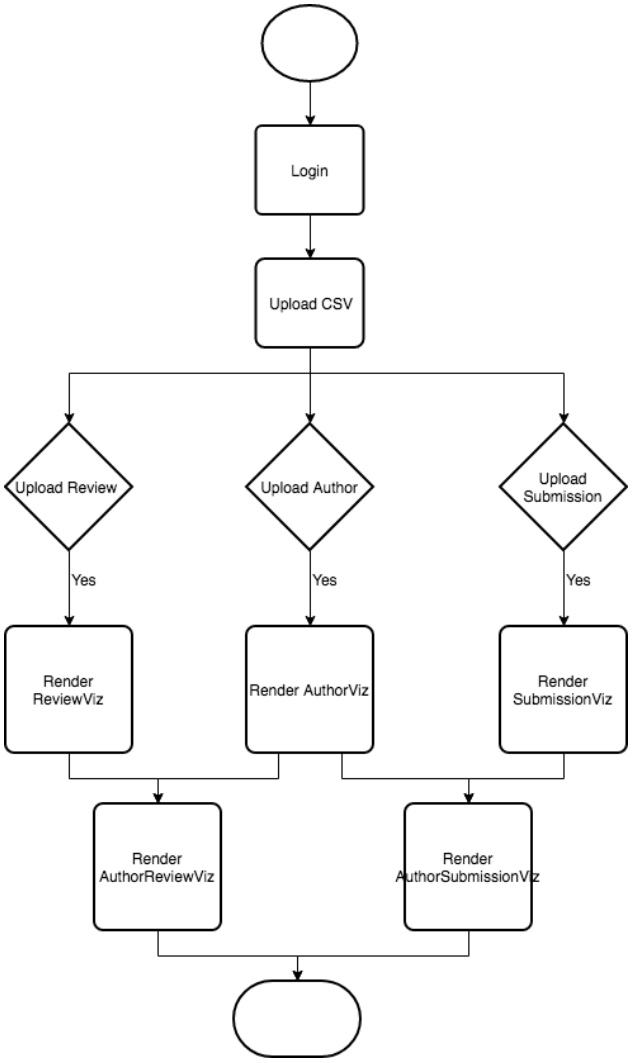
1. Vue Components Architecture



Design Decision:

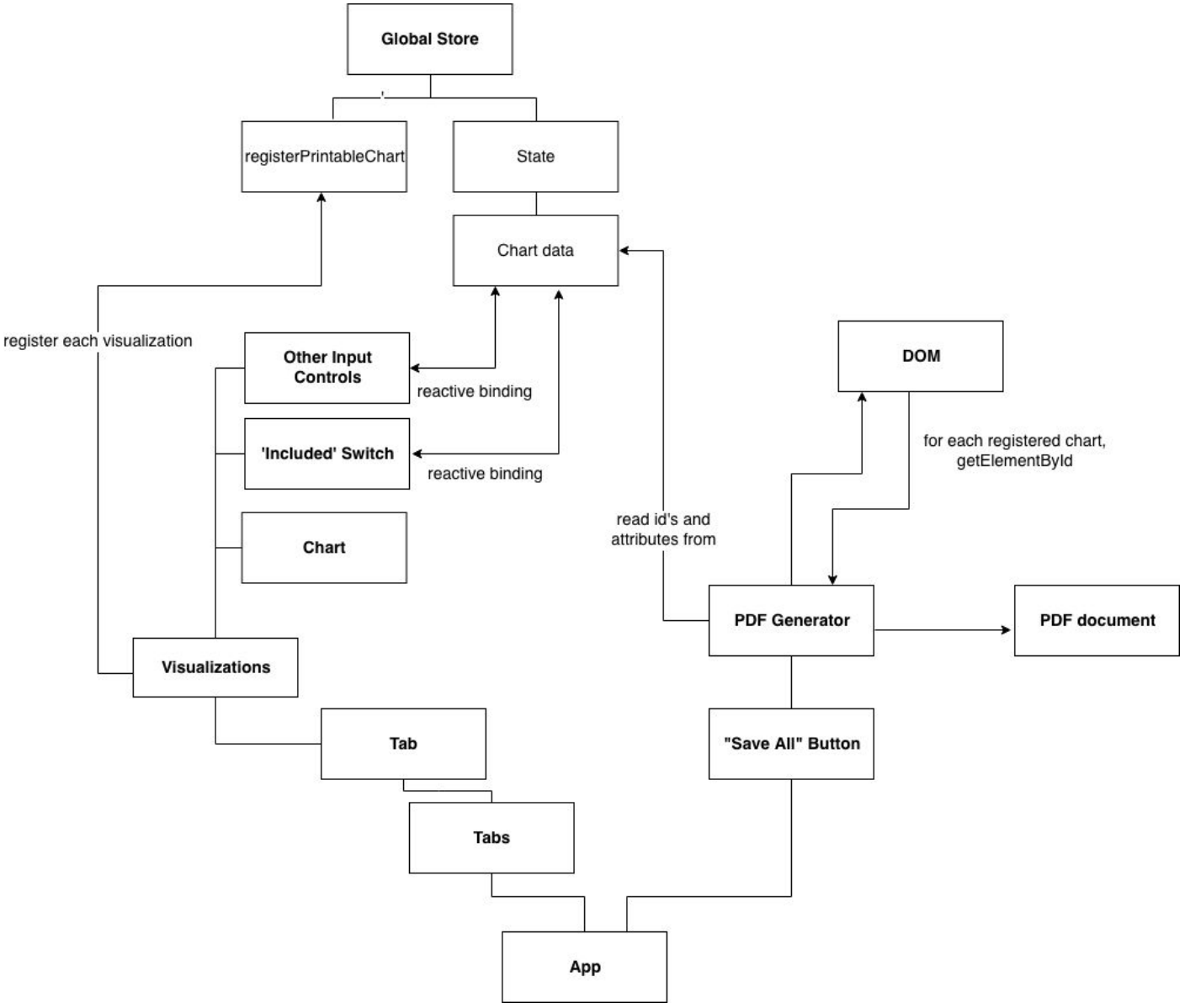
In ChairVisE 1.0 all the visualizations are drawn in one component. In ChairVisE 2.0, we decided to adopt modular programming by splitting the visualisation rendering into five independent components. We choose to follow component-based software engineering and separation of concerns. Each rendering component addresses only one concern. For example, if the user uploads an author file, only the AuthorViz component will be triggered and handles the drawing for visualisations that are related to author data.

2. User Flow - render files on uploading files



This diagram details the user flow involved in Functional requirement F2.6. For visualizations that require data from multiple files, they should be available once the necessary files have been submitted. For example, once review and author are available, Author x Review visualizations will be rendered.

3. Generating PDF



Overview

In order to save all charts into a PDF, we must store information about each chart at the global level - in a store - so that any component can access it. This will allow us to invoke the PDF generation process from any level of the app. The alternative is to use events, which will not only be inconvenient, it will also couple each chart with whichever root component contains its data.

Previous approach: component-level methods and data

Previously the saving of visualizations into PDF was done via standalone component-level methods such as `saveAuthor` or `saveSubmission`. These were lengthy and highly repetitive.
The main motivation of the new mechanism detailed below is to be able to print all charts in one document, which was not possible via the previous approach as each method had to access component-level data. The new mechanism also abstracts the process of generating the PDF, and reduces code duplication.

New approach: global store and PdfGenerator class

The class PdfGenerator in utils uses html2canvas to transform the html elements created with chartjs to canvas elements, and then saves them in a pdf document. It takes care of formatting, margins, styling, etc. We use a global store object to keep track of all the charts and whether each is included or not - this helps us avoid having to:

- declare chart-printing-related information at the App level and pass them as props to individual components through ResultTabs to each tab, and
- use events to trigger changes in chart info from the child to the parent, thus decoupling each set of charts from the main App.

Usage at viz component level:

1. Register the chart id's in the global store at the created() hook:

```
created() {
  // use constants, and use snake case for the actual id string
   Store.registerPrintableChart(CHART_ID_TOP_AUTHOR_BAR);
   Store.registerPrintableChart(CHART_ID_TOP_COUNTRY_BAR);
}
```

2. As you can see, registerPrintableChart simply assigns id as a key with included true by default.We use Object.assign because of Vue's change detection caveats for added properties (we want to register charts dynamically at component level).

```
registerPrintableChart(id, included = true) {
  this.state.charts = Object.assign({}, this.state.charts,
    { [id]: { included } }
  );
}
```

3. Bind the included switch, or any other data, to global state:

```
<el-switch
      v-model="storeState.charts[CHART_ID_TOP_AUTHOR_BAR].included"
      ...
  />
```

Toggling the switch will automatically modify that info in the global state under chart id, allowing PdfGenerator to parse that info when needed.

Usage at print button level:

```
generatePdf() {
  const pdfGenerator = new Utils.PdfGenerator();
  pdfGenerator.generate([CHART_ID_TOP_AUTHOR_BAR,
CHART_ID_TOP_COUNTRY_BAR]);
}
```

PdfGenerator may be invoked in any component that you need to insert a print button, and can print any set of charts.
Currently PdfGenerator is only used at the App level, and is given all id's explicitly. This allows explicit ordering of charts. If it is not given any id's it will automatically iterate through all charts registered in the global state (in no guaranteed order).

Notes

- When we bind the global state with v-model we are directly mutating state, which is not good practice - however, the code is way simpler this way: we do not have to define a unique computed property with a getter and setter for every single chart's included attribute. I tried to do something generic with either computed or methods but wasn't able to. We would have to do individual computed properties for other data that involve more complicated mutation (such as addition of properties) in the future.
- On a related note, when directly exposing global data in the Vue instance, it is best to declare Store.state as opposed to some property such as Store.state.charts. This is because that property may rely on Object re-assignment (see registerPrintableChart) in order to set new reactive properties. This means upon a new registration, Store.state.charts now holds a reference to another object, and will not detect changes made by the any bindings made on the initial Store.state.charts data declared.
- Currently, we avoid storing the caption data, and simply access the caption text in the DOM via document.getElementById(CHART_ID).nextElementSibling.innerText. This is a weak assumption but greatly simplifies the process.

e) Suggestions for improvements and enhancements

1. Frontend: Usability

1.1. Grid Layout for Charts (e.g. https://github.com/jbaysolutions/vue-grid-layout)
  1.1.1. With vue-grid-view, visualisations can be drag around and relocate to the desired location. It is easier for the user to compare two or more visualisation side by side.
1.2. Allow user to choose whether headers are included or not when mapping headers
1.3. Map all columns in headers
1.4. Printing PDF
  1.4.1. User Preferences: Users can set preferences for PDF document such as font, aspect ratio, image size, etc.

2. Frontend: System Design

2.1. State Management with VueX: M
2.2. Abstracting Generic Chart Functionality with Higher-Order Components
2.3. Chart Rendering
2.4. Printing PDFs
  2.4.1. Chart Library with native Canvas/Image conversion/Export feature
  2.4.2. Document generation in backend

3. Backend: System Design

3.1. Apply appropriate normalization and further decompose CSV data to data more amenable for storing in a database. Right now, the data is stored almost as-is from the CSV for simplicity. This is also related to performance as we did not care too much about

performance but we would have to when we think about normalization as too much normalization causes performance problems but too little causes data integrity issues.

3.2. Further enhance scalability and performance of application (if requirements require). The application already takes into some consideration some scalability concerns such as memory concerns with large files. This is done by deferring analysis of data to database as much as possible (the way data is stored does not always allow this) and use of generators. Performance may be improved by redesigning schema, optimizing queries and use of indexes.

3.3. Re-structure code to follow “MVC” (more accurately MTV in Django) so there is better separation of concerns.

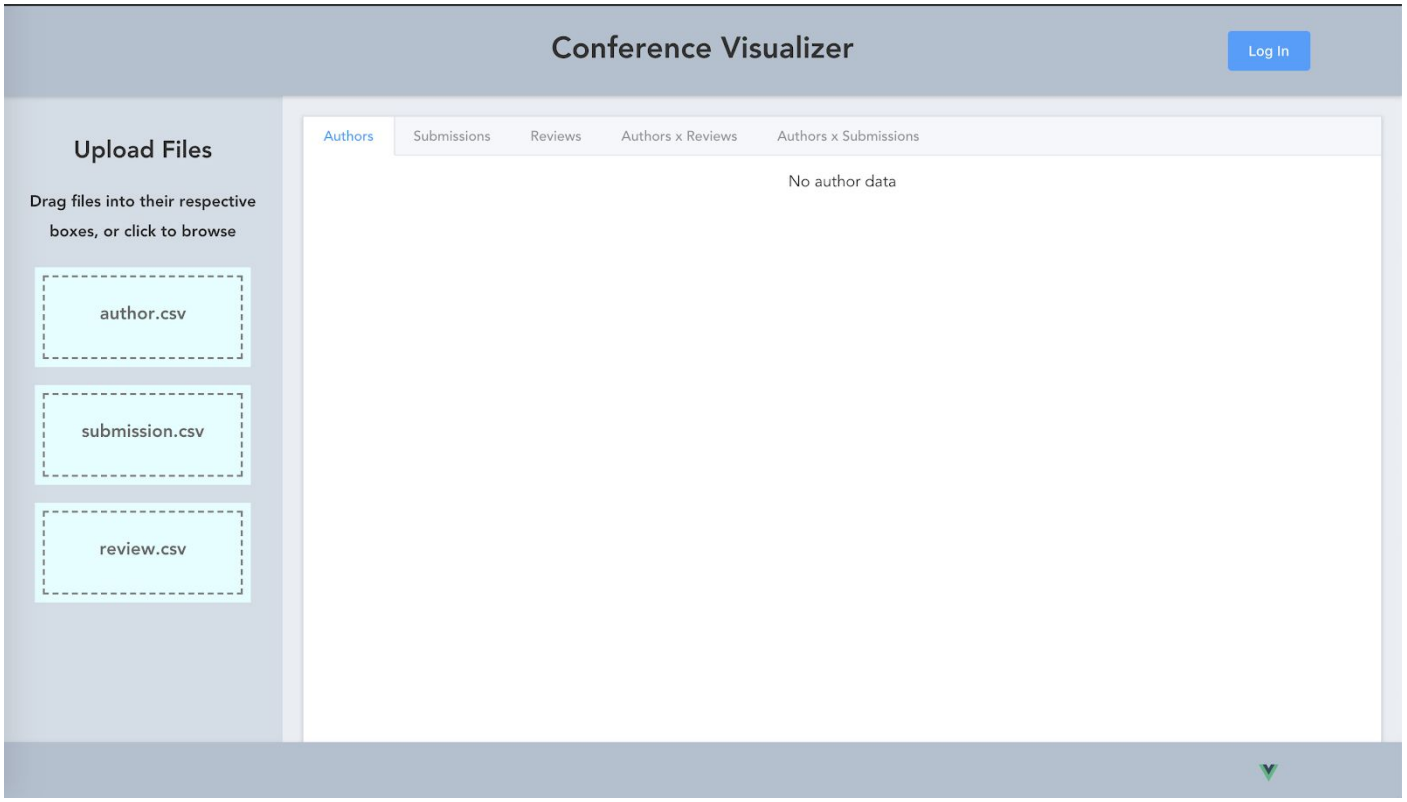
4. Others

4.1. Enforce a consistent coding style. Right now in the frontend we have not enforced a strict coding style to use and the backend has a mix of code that follows the PEP style guide and other code that deviates from it.

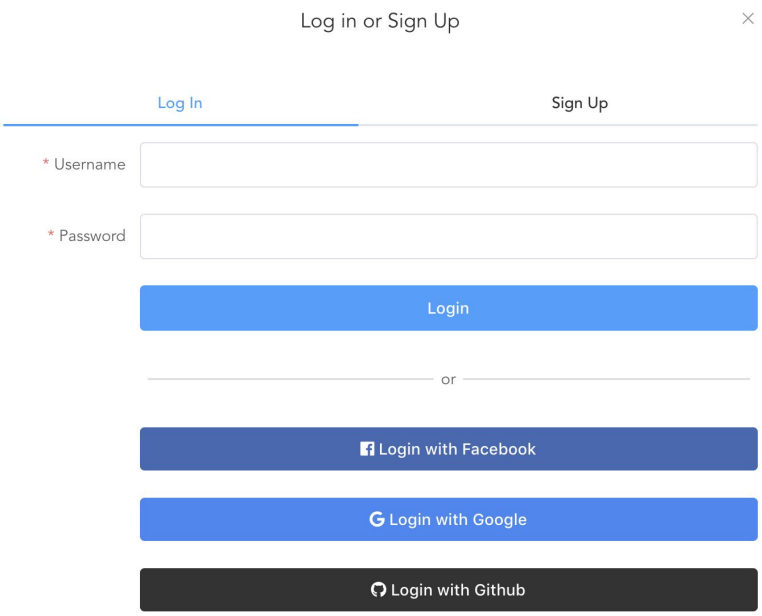
4.2. Use a consistent naming convention for Restful APIs

Enhancement and Visualization Screenshots

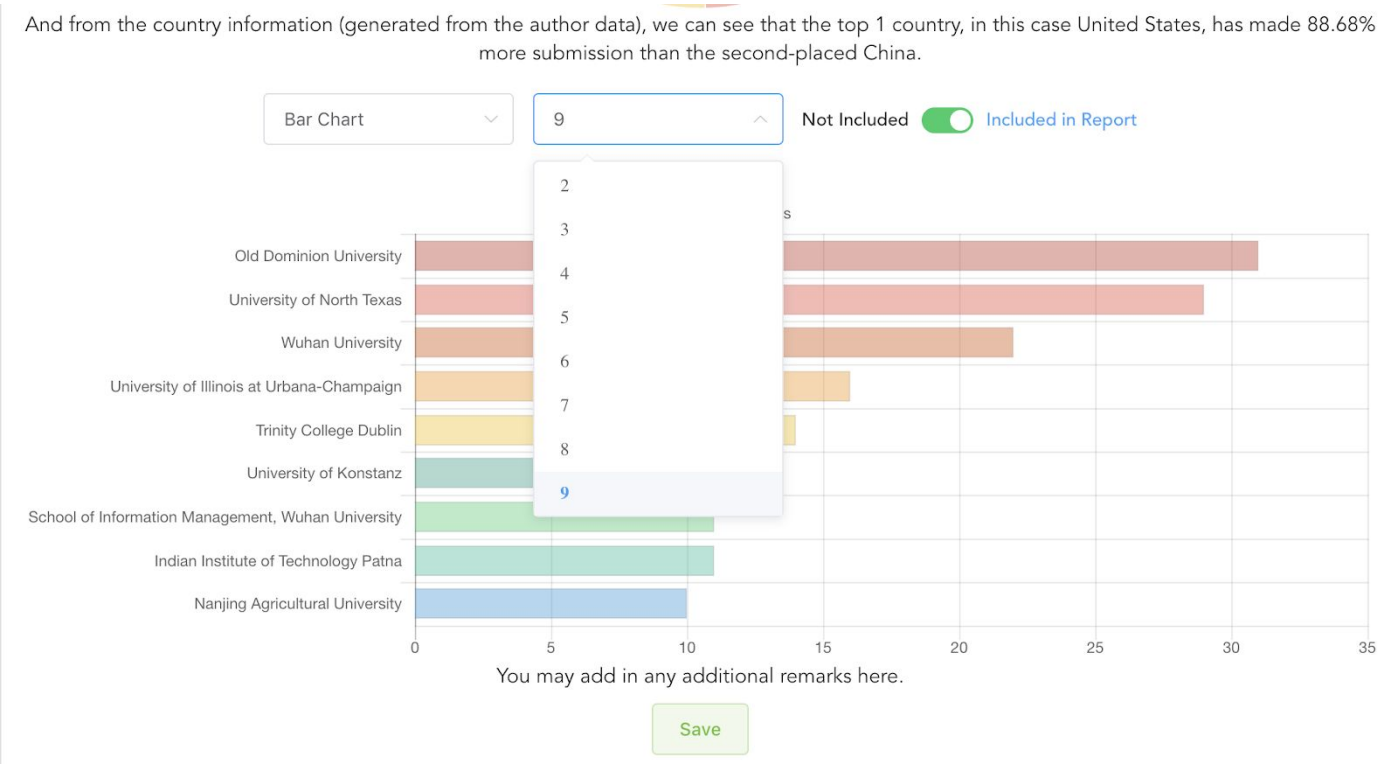
1. Enhancement: New App Launching Page with Tabs to render visualisation



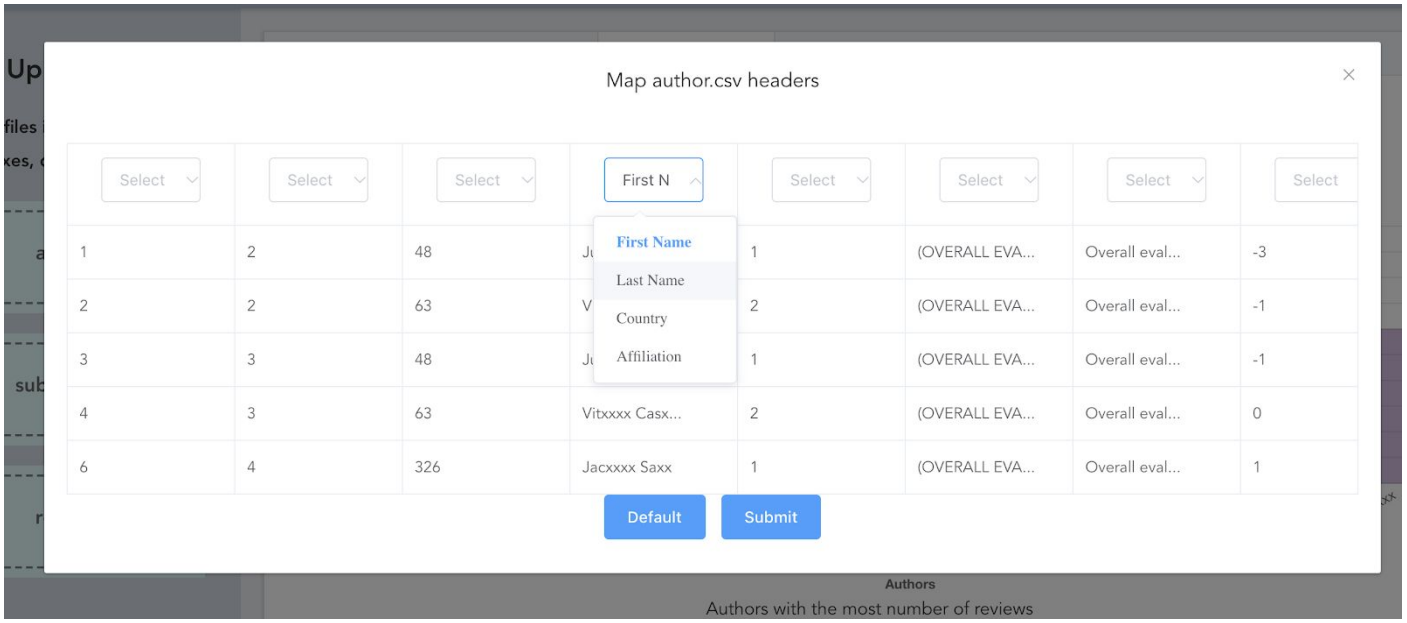
2. New Feature: Login/Signup modal



3. Enhancement: Customizing Rendering Data Length



4. Enhancement: Mapping Headers

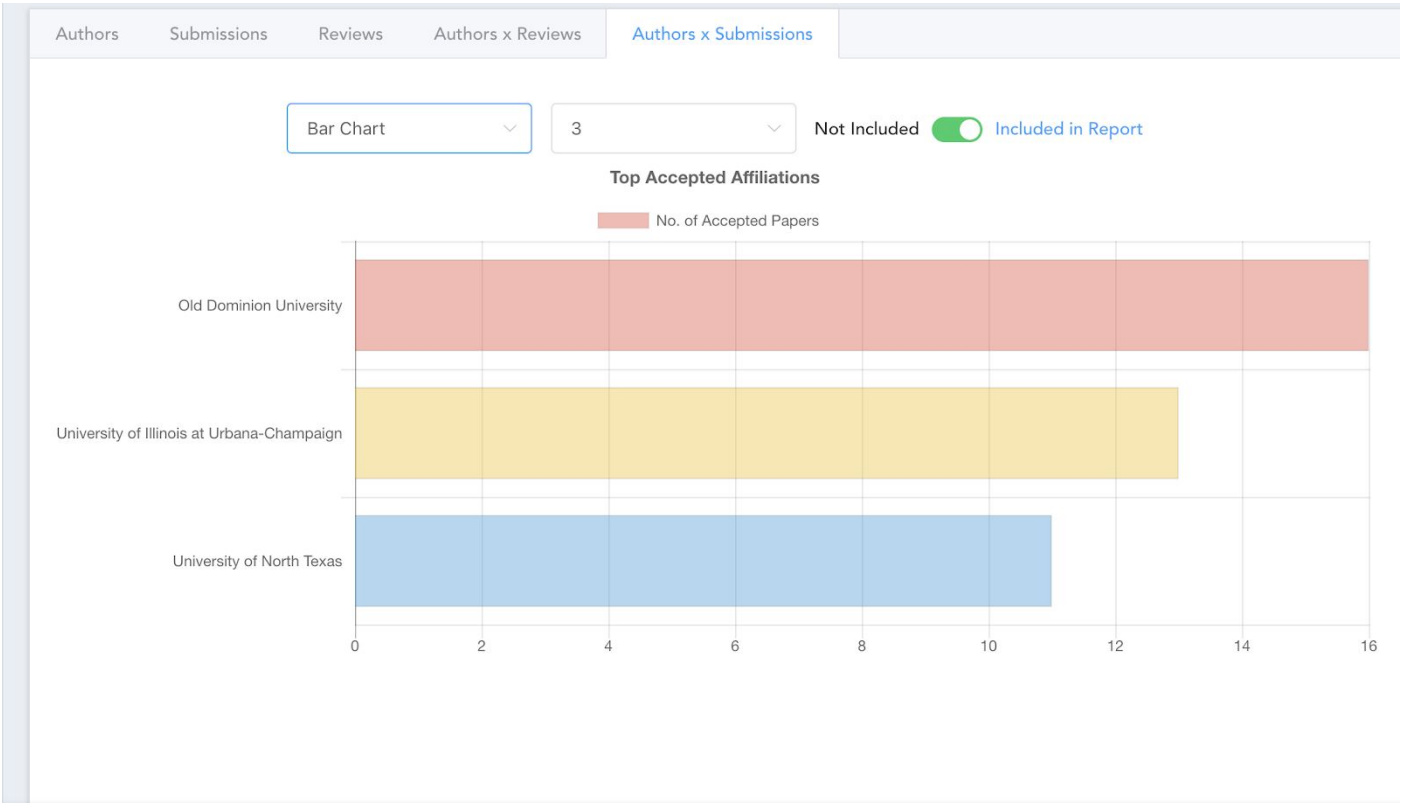


5. Word Cloud Filter (Filter by All/Accepted/Rejected Submissions)

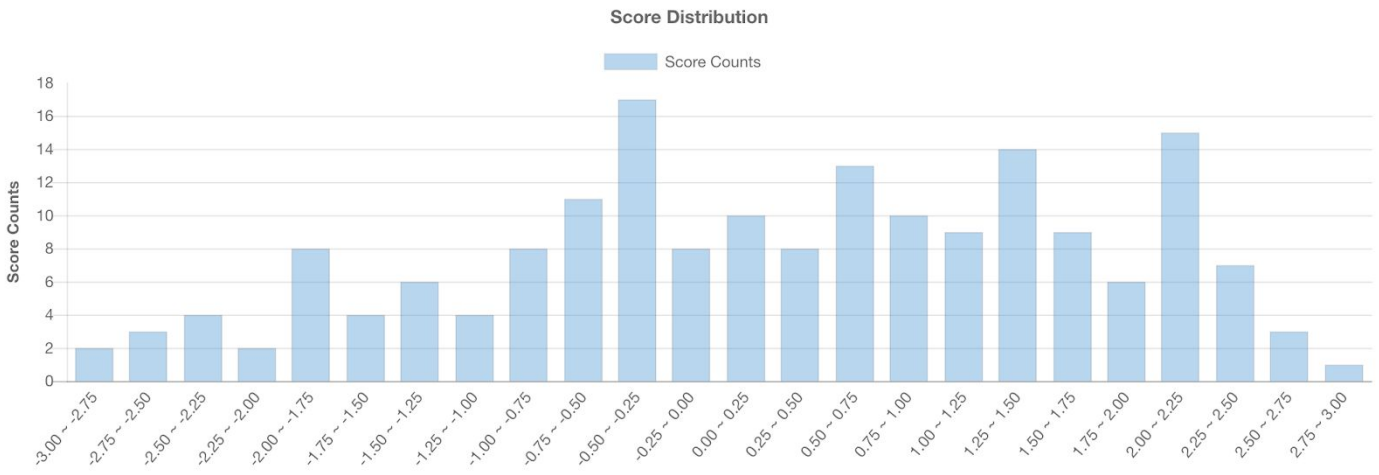
6. New Viz: Top Reviewed Author

Authors with the most number of reviews

7. New Viz: Top Accepted Affiliations



8. Enhancement: Added Axis Label for Visualisations


Score Distribution
Score Counts
Score Counts
18
16
14
12
10
8
6
4
2
0
-3.00 ~ -2.75
-2.75 ~ -2.50
-2.50 ~ -2.25
-2.25 ~ -2.00
-2.00 ~ -1.75
-1.75 ~ -1.50
-1.50 ~ -1.25
-1.25 ~ -1.00
-1.00 ~ -0.75
-0.75 ~ -0.50
-0.50 ~ -0.25
-0.25 ~ 0.00
0.00 ~ 0.25
0.25 ~ 0.50
0.50 ~ 0.75
0.75 ~ 1.00
1.00 ~ 1.25
1.25 ~ 1.50
1.50 ~ 1.75
1.75 ~ 2.00
2.00 ~ 2.25
2.25 ~ 2.50
2.50 ~ 2.75
2.75 ~ 3.00