

Effortless BI (Eff BI)

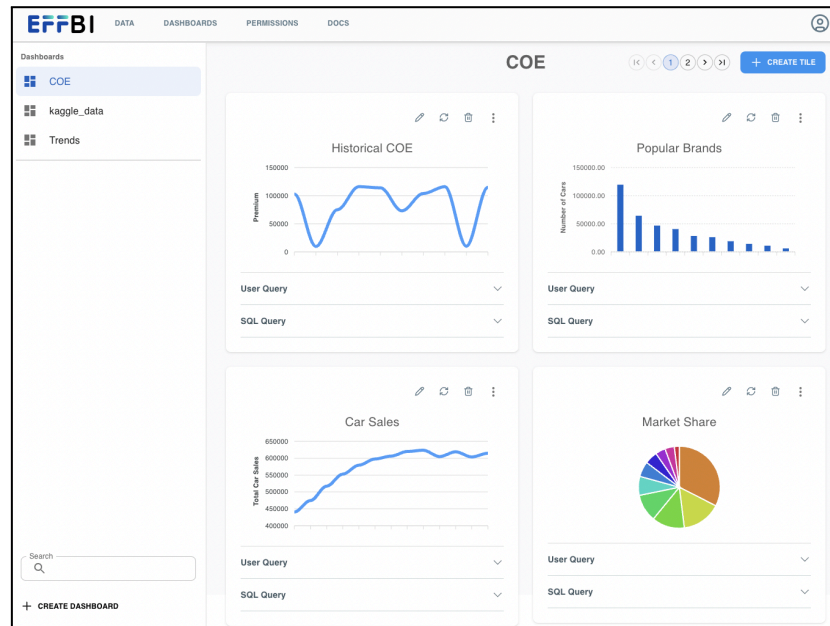
CS3216 Final Report

Tan Wee Kian Justin	A0252153N
Muhammad Reyaaz	A0218022W
Ho Cheng En Bryan	A0234509E
Vaishnav Muralidharan	A0235268Y

[Link to Project Source Code](#)

Introduction to Eff BI

Eff BI (Effortless Business Intelligence) aims to revolutionise how organisations interact with data by converting natural language queries into SQL and dynamic visualisations, thus eliminating the need for technical expertise. At the core of Eff BI is our sophisticated chart generation technology, which interprets English queries and transforms them into an appropriate visualisation chart.

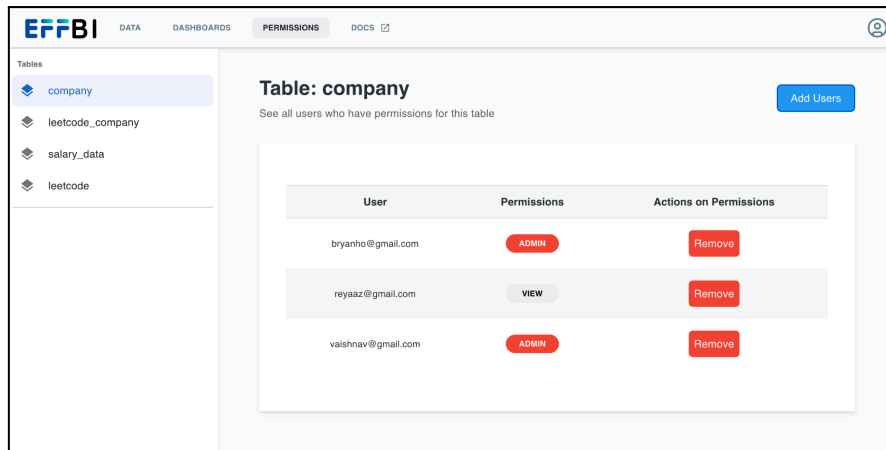


Example of a customised dashboard with visualisation charts

These charts can be grouped into custom dashboards tailored to organisations' specific analytical needs. Eff BI's also facilitates the easy export of visualisations as JPEG and PNG, enhancing the communication and sharing of insights both within and outside organisation.

Eff BI connects to an organisation's database through a specified database URI. Users can then preview sample data, and all new users added to the organisation will be automatically connected to this database. This workflow aims to streamline the onboarding process and allow users to efficiently access their data directly within Eff BI, eliminating the need to switch between multiple data tool applications.


To support organisations to manage user permissions effectively, Eff BI also incorporates user access management. This allows administrators to manage access controls and data security within users of an organisation.





Example of user permissions

As a fully open-source application, the Eff BI team believes in making data analysis and insights accessible to everyone. Our application is accompanied by comprehensive documentation guides available to both [users](#) and future [developers](#). Additionally, we maintain [blogs](#) that detail various stages of development, to document the application's evolution and give guidance on leveraging Eff BI capabilities' to the fullest.

Competitive Landscape of NL2SQL Applications

Competitors	Pros	Cons
Power BI 	<ol style="list-style-type: none"> Power BI's Q&A features enable users to explore data using natural language. Users of all skill levels can retrieve insights without needing expertise in data query languages. Visualisations generated by the Q&A feature can be easily added to dashboards or reports, enhancing data organisation and presentation. The Q&A feature interprets user query and automatically 	<ol style="list-style-type: none"> As an integrated feature in Power BI, Q&A is exclusively available within the Power BI application. Hence, this feature is only limited to users/organisations in the Microsoft ecosystem. The Q&A feature is integrated with Copilot. Users do not have the flexibility of integrating this feature with other LLMs. Queries about the data model are only stored for up to 28 days. This short retention

	<p>selects the most appropriate chart type from a range of chart type options. This ensures that visualisations are optimally formatted for data representation.</p>	<p>period could affect the data analysis process as users work under time sensitivity.</p> <p>4. Currently, Q&A is unable to support multiple conditions in queries which limit the usefulness of this feature.</p>
<p>Shakudo</p> 	<ol style="list-style-type: none"> 1. Shakudo is able to gather data from multiple sources using tools like Airbyte and n8n. 2. Shakudo enhances user data by enriching it with metadata, providing explanations and descriptions of data schema. This allows users to better understand the data context. 3. Shakudo enables users to perform queries using natural language. Relevant tables and columns are looked up and a SQL query is auto-generated. 	<ol style="list-style-type: none"> 1. Utilises open-source LLMs such as FlanT5 Models and Falcon7b Model. Users do not have the flexibility of integrating this feature with other LLMs. 2. Shakudo's pricing model, which charges per application container per hour, leads to high operational costs. According to Google Marketplace, Shakudo charges \$5.70 USD per hour per tagged container. An organisation running 100 containers throughout the year will be charged \$50,019.60 USD.
<p>ChatGPT</p> 	<ol style="list-style-type: none"> 1. Most users are familiar with ChatGPT and its conversational interaction style. ChatGPT has a low barrier of entry for users and reduces time/effort spent on user adoption. 2. User-friendly call and response interaction style. This allows for iterative improvements based on user inputs which could help to optimise the data querying process. 	<ol style="list-style-type: none"> 1. ChatGPT lacks context knowledge on database/table schemas which affects the relevancy and accuracy of the generated SQL queries. 2. ChatGPT has a threshold on the number of tokens it can take, making it prone to hallucination on large amounts of data. 3. There is no way to group and store all generated visualisations except for

		going through the chat history where these visualisations were generated, which can be cumbersome.
--	--	--

How Eff BI *Effectively* Stands Out in the Market

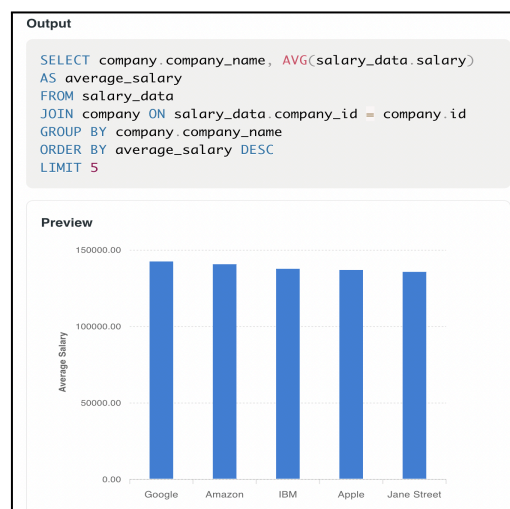
Eff BI uniquely addresses the gaps in the NL2SQL space in various aspects. Through a detailed analysis of the pros and cons associated with existing solutions, we designed Eff BI to offer an integrated, advanced, accessible business intelligence tool targeted for enterprises.

Eff BI as a centralised data platform

Eff BI provides a centralised platform that extends beyond simple NL2SQL capabilities. In addition to converting natural language into SQL, Eff BI also offers a comprehensive suite of features that facilitate data preview, user access management, query to visualisation and insights generation. It also supports dashboarding functions, enabling users to query and manage data insights in an interactive user interface. This makes Eff BI a multifaceted data analytics solution.

Advanced Query to Visualization

Eff BI innovates beyond the conventional NL2SQL model by not only translating natural language queries into SQL but also automatically converting these queries into dynamic, intuitive visualisations.



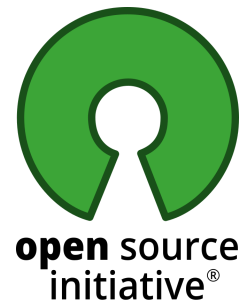
Example of SQL and data visualisation

This process of seamless conversion from user queries to visualisation allows users to bypass complex SQL interpretations and directly interact with insightful, actionable visual representations of their data. This user workflow significantly enhances organisational decision-making efficiency and effectiveness, without the need for constructing SQL outputs.

Open Source Flexibility

LLM Usage

Our application is licensed by [The MIT License](#). As an open-source solution, businesses have complete control over the type of LLM used. Companies can enhance their insights by choosing an LLM of their choice, or deploy models within their private infrastructure (such as AWS Bedrock) for maximum data security. This flexibility enhances performance and provides stronger privacy protections.



Accessibility

Being open-source, Eff BI is completely free to use, without “middleman costs”. Companies only have to pay for the LLM API requests made, making it an economically viable solution for businesses of all sizes. It is especially useful for small and medium sized businesses that need to make data-driven decisions but cannot afford to hire specialised BI developers.

Optimised for Organisations

Designed with enterprise needs in mind, Eff BI is specially designed to meet the demands of enterprise environments. Our application incorporates robust user access management to data tables and query generation, for security within organisations. This feature ensures that sensitive data is accessed only by authorised personnel within the organisation.

Review of milestones and timeline

Overall Milestones and Task Management

For sprint and task management, our team leveraged [Notion's](#) shared workspace for planning and allocation of tasks. This ensured that we remained on track and kept us accountable to one another. As a result, our group met most of the established milestones, except for those that were intentionally rescheduled.

Sprint Tasks

Table

New

Time Window - 26 Oct - 27 Oct

Task Name	Priority	Assigned To	Status	Due Date
UI Improvements		Bryan	DONE	October 16, 2024
LLM Pipeline for Chart Generation		VaishnavReyaaz	DONE	October 16, 2024
View Data Page		Justin	DONE	October 16, 2024
Deploy to GCP		Vaishnav	DONE	October 17, 2024
Database Refresh Page		BryanJustin	DONE	October 22, 2024
Validate User Access	HIGH	Vaishnav	DONE	October 25, 2024
Settings- User Profile Page	LOW	JustinBryan	DONE	October 25, 2024
Getting Started	LOW	Justin	DONE	October 25, 2024
Fix SQL Validation in Tile Generation	HIGH	VaishnavReyaaz	DONE	October 25, 2024
Fix Database URI in Sign up Flow	HIGH	Justin	DONE	October 25, 2024
Configure Logging for GCP	HIGH	Bryan	DONE	October 25, 2024
Dashboard and Tile Popup Changes (UI)	MEDIUM	ReyaazVaishnav	DONE	October 25, 2024
Progress Report 2	MEDIUM	VaishnavReyaazBryanJustin	DONE	
Explaining to the User What is wrong with the Query	MEDIUM	Bryan	DONE	October 25, 2024

Example of a Sprint Planning on Notion

Adjusted Key Milestones

Database Management System (DBMS) Support Limitation

Initially, we planned to support multiple DBMS types to cater to a broad range of database environments. However, we chose to focus exclusively on PostgreSQL as we were most familiar with it. This decision was driven by our priority to refine and perfect our core features, adding meaningful depth rather than expanding in breadth without substantial functional gains.

EFFBI

DATA

DASHBOARDS

PERMISSIONS

DOCS

Database Settings

Connect your database to view your data in Eff BI. Enter your database URI below.

Database Type

PostgreSQL

MySQL

SQLite

ORACLE

Database URI

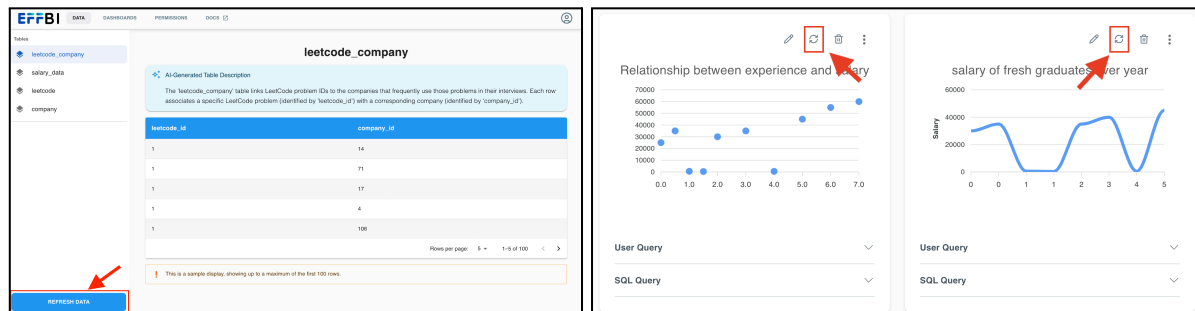
postgres://user@localhost:5432

SAVE

Database Settings page

Introduction of the Refresh Feature

An important addition in this phase was the refresh feature, which was not initially planned for. Organisation data is not stale and evolves over time. This feature allows visualisations to be fed with the latest updated data from the database. This ensures that users receive the most accurate and up-to-date data from their database.



Refresh feature (highlighted in red) on the Data Page and Visualisation charts respectively

Mockup and User Interface (UI) Adjustments

Initially, our intention was to develop high-fidelity prototypes for the entire application. However, due to time constraints, this was only applied for our landing page. Instead, we utilised Figma mockups with skeletal designs that captured the essential functionality without delving into finer details early on. As the project neared completion, user feedback indicated the need for significant improvements, including a complete overhaul of the colour scheme and numerous other UI/UX adjustments. In hindsight, making low-fidelity mockups coupled with earlier stages of user testing might have offered a more optimal balance. This approach could have streamlined our development process by identifying usability issues and design preferences earlier, potentially saving time and resources.

Fortunately, recognising the potential for unforeseen challenges, we intentionally incorporated **buffer time** into our initial project schedule. This proved to be useful as it allowed us to adapt and respond effectively to the need for the comprehensive UI revamp. We maximised the feedback received from user testing, mentors and alumni, and were able to implement critical updates that significantly enhanced the overall user experience. This ensured that our application finally met the high standards that our users and ourselves expected from Eff BI.

Individual Contributions

Justin: Full-stack developer and lead database engineer who developed features for users to interface with their database and organisation-wide access management features. He also managed the creation and deployments of our databases.

Reyaaz: Full-Stack Engineer and lead frontend developer, who integrated supertokens authentication and authorization into EffBI alongside several frontend features such as the landing page. He built the dashboards UI, chart templates and was also involved in the development of our LLM Pipeline.

Vaishnav: Product Manager and Full-Stack Engineer who built the multi-agent LLM Pipeline alongside the dashboard management features and was in charge of the team's DevOps practices (Docker, CI/CD to GCP and Vercel).

Bryan: Full-Stack Developer and Backend Engineer who developed features to snapshot database, table permissions and refresh organisation table metadata. Also wrote user guide documentation.

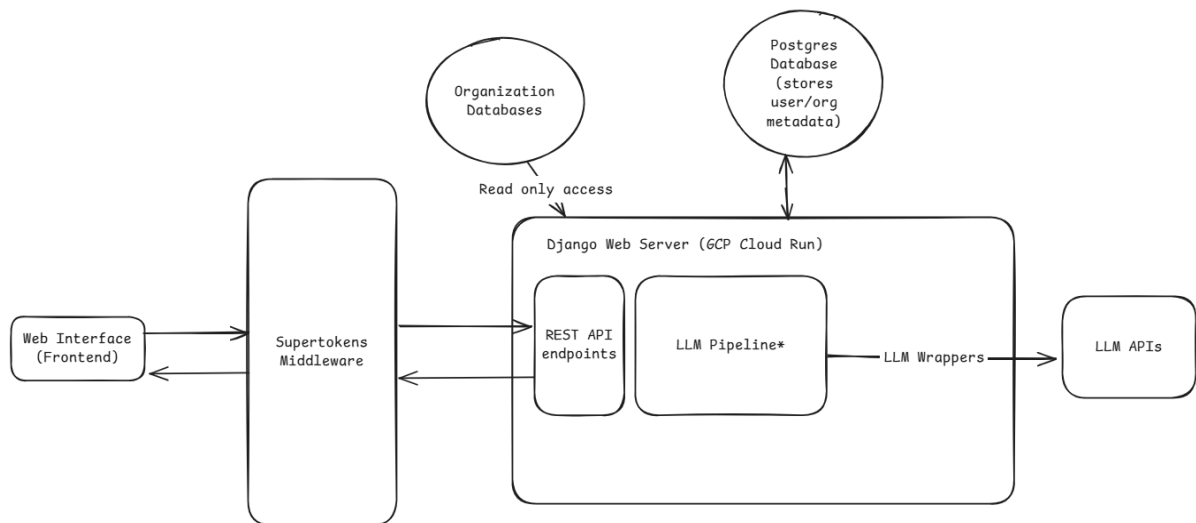
Acknowledgement of resources by external parties

We referenced this [GitHub repository](#) to guide the architecture of our LLM for text-to-visualisation generation. Additionally, the SuperTokens team on Discord provided helpful assistance in resolving deployment challenges with the SuperToken integration.

We would like to extend special thanks to Dr Suzanna Sia, who not only advised us on model selection and the strategic direction of our application but also inspired us to embrace an open-source approach. She facilitated user testing by connecting us with business users from Hyundai - Shermaine (Business Development Manager) and Japheth (Business Analyst). Suzanna also provided \$100 for all our technical needs, of which we invested over \$60 into LLM cost.

Finally, we are deeply grateful to Prof Soo for his insightful feedback which has been invaluable in enhancing our project's appeal and effectively marketing it during STePS. We also greatly appreciate all our user testers, particularly Christopher, Justin, and Didymus. Christopher gave us a crucial wake-up call that motivated a significant redesign of our application. Justin provided detailed feedback on UI improvements, while Didymus shared valuable insights on creating more intuitive user flows for first-time users.

Application Design



Above is the birdseye view of our application architecture. We will delve more into below.

Supertokens Session Authentication

We use a third-party session authentication service, SuperTokens, incorporating their SDK in both our React web application and our Django web service. We chose it over Firebase Authentication because SuperTokens is free at scale, unlike Firebase, which imposes rate limits after a certain number of users. Additionally, SuperTokens is open-source, aligning with Eff BI open-source nature.

Our authentication system works seamlessly across both frontend and backend layers. The frontend SDK automatically adds access and refresh tokens to all outgoing requests, while our backend middleware handles the complete authentication flow - from validating tokens to managing sign-in and sign-up processes. While we provide a cloud-hosted version of SuperTokens Core to manage authentication data, you can also deploy it yourself using our public Docker image, giving you complete control over your authentication infrastructure.

Backend Server and Database

Our system uses a Django Web Server to expose endpoints to users and a PostgreSQL Database to store and manage various types of data, such as:

- User and organisation information (e.g., access permissions and table metadata).
- Chart properties, which are fed into the frontend (explained in detail in the LLM pipeline section below).

We chose Django because of its powerful ORM, its compatibility with the LangChain SDK for Python, and its usage of the MVC (Model-View-Controller) pattern, which promotes clear separation of concerns, making it easier for quick development. For data storage, we opted for PostgreSQL due to its data durability and ability to efficiently handle relational queries involving user, organisation data and permissions tables.

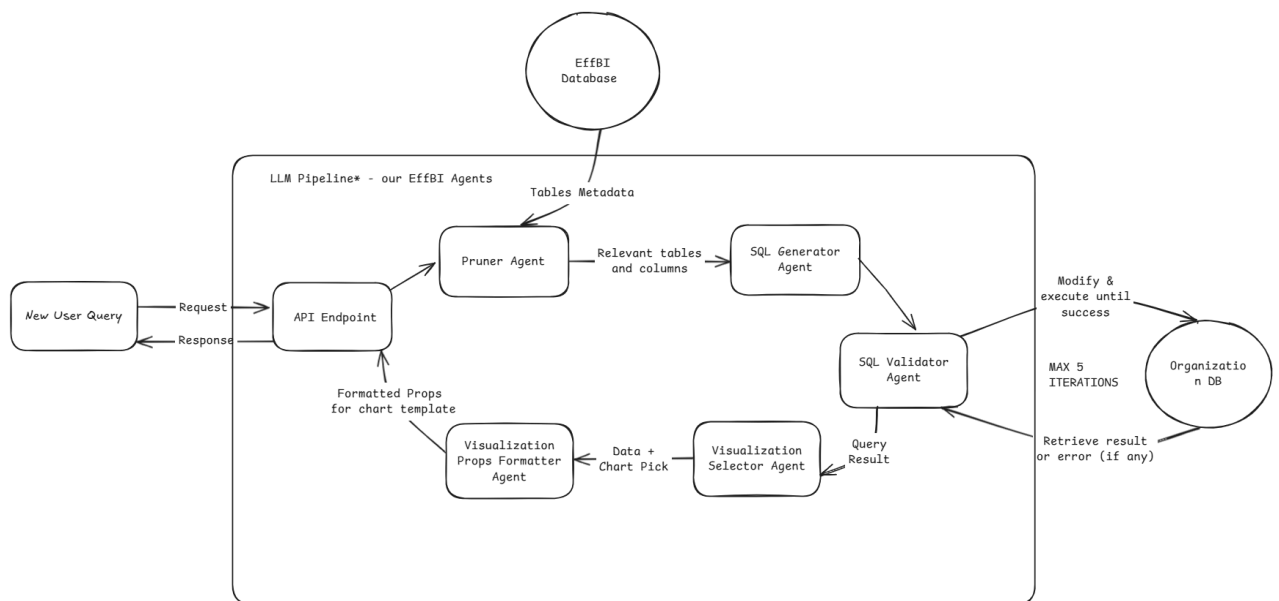
We used Docker to build the images for Django and Postgres so it could be run both locally as well as be deployed to a cloud service (GCP in our case). This was setup to ensure that everyone had the same local environment when developing their backend code. We did not containerize the frontend as its hot reload was extremely slow as opposed to running localhost directly and hence decided to make the tradeoff between containerisation and speed of development.

The CI/CD Pipeline was set up using GCP Cloud Build for the Django service and Vercel's deployment pipeline for the frontend. While the former built Docker images upon every merged PR and deployed it to GCP Cloud Run, Vercel's deployment pipeline similarly built the frontend application and deployed it to its hosted service too. We will look to move away from the 'monorepo' structure we have used in our upcoming open-source issues. While monorepo helps in speed of development and testing and ensures a single source of truth, a new backend build is triggered upon a solely frontend change and vice-versa, leading to unnecessary builds.

LLM Pipeline

The core feature of our application is a strategic use of state of the art LLMs in being able to generate SQL queries and visualisations for the user. A naive approach would have been dumping the organisation data (possibly augmenting an LLM with a vector DB) into a text-to-image model like DALL-E. However, this would pose a serious problem in maintaining consistency of the charts when refreshing them (we would need the same chart type with different data and models are not very deterministic).

Thus, we decided to opt for a multi-stage retrieval pipeline, inspired by a LangGraph based design by [Dhruv Atreja](#). We modify this by pre-computing the metadata for each table in the organisation table - table column types and descriptors, as well as table descriptors. This not only presents the user the ability to better understand their data, but also for the LLM Pipeline to make decisions on which tables and columns to use for generating the SQL query. This [blog post](#) delves deeper into how Eff BI snapshots the organisation database and stores table metadata.



As seen in the above diagram, the LLM Pipeline is triggered by the user making a request to the API endpoint to visualise a query. We have 5 “Eff BI Agents” - a graph of specialised LLM agents that reduce hallucinations to provide accurate visualisations for user queries. This [blog post](#) on our documentation site delves deeper into the different agents we use, but in short, the processing consists of 5 stages:

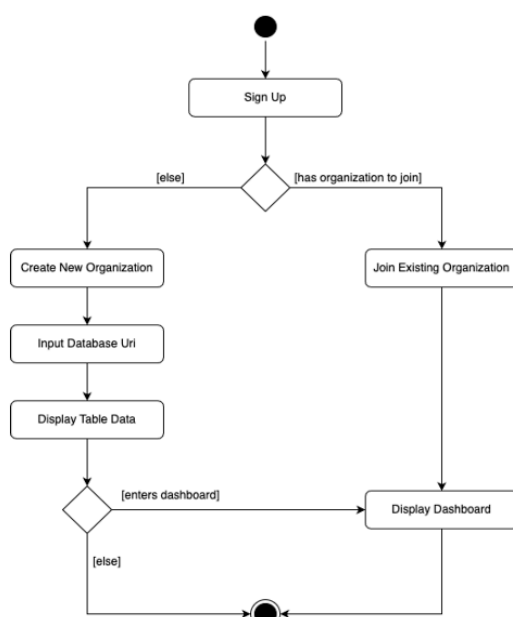
1. **Pruner Agent** - prunes relevant tables and columns for SQL generation, and terminates the pipeline if the user query is either malicious or if the user does not have access to the data required to answer the query.
2. **SQL Generator Agent** - Generates SQL query from the narrowed set of tables and columns.
3. **SQL Validator Agent** - Iteratively executes and regenerates SQL query based on the error message to ensure correct SQL Query, maximum iterations currently are capped at five.
4. **Visualisation Selector Agent** - Takes in user query, data generated from SQL and user’s preferences for chart visualisation type. Heavily biased towards user preferences for chart type.
5. **Visualisation Formatter Agent** - according to the selected visualisation, we have pre-built chart templates for which we have corresponding props. This LLM Agent formats the data to be passed as props into the frontend.

This architecture yields accurate results and achieves minimal hallucination despite the limitations of the state of the art AI. Currently, OpenAI’s GPT-4o model is being used as it is one of the best performing models and is the model that accepts the most number of

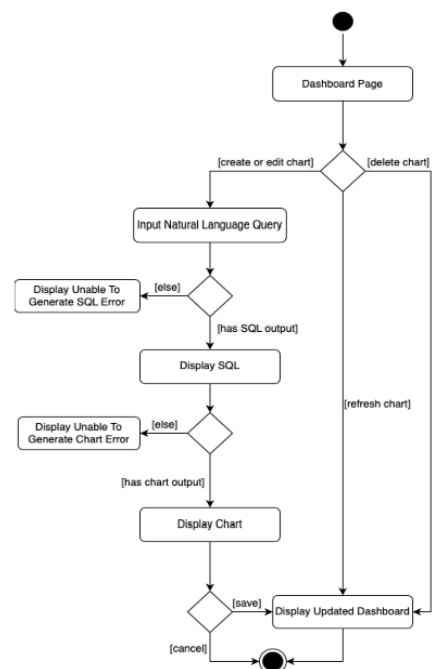
parameters according to [Zapier](#), so it scales well with large organisational data. Additionally, based on the [BIRD benchmark](#) used by Dr. Suzanna to evaluate and select an LLM model, OpenAI's GPT-4 consistently ranks as the top-performing LLM across various criteria.

However, if the user still wants to change the LLM model, our LangChain based LLM wrappers and the open-sourced nature of our project means that users can replace the LLM with one of their choice, leaving the processing of the queries untouched.

User Flows



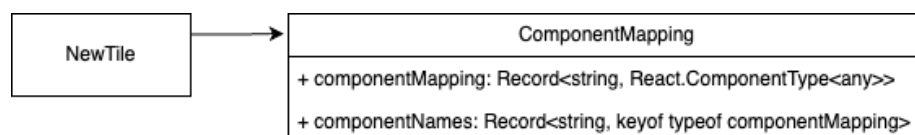
Sign up user flow



Creating a chart user flow

Design Patterns

Registry pattern



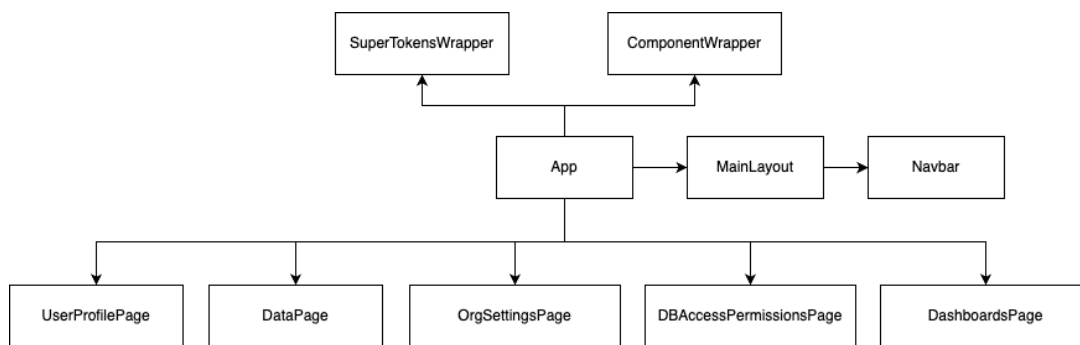
We decided to employ the registry pattern for the frontend **NewTile** component by creating the **ComponentMapping**. This aligns with the **open-closed principle**.

There are a few alternatives we considered:

1. **Naive Approach:** Using a long switch-case or if-else statements to check the input string (e.g., if the input is a bar chart, use the bar chart template, and so on).
However, this clearly violates the **open-closed principle**, as adding new chart types in the future would require modifying the function itself.
2. **Superclass-Subclass Relationship:** Creating a superclass-subclass hierarchy, where each chart type (e.g., bar chart, pie chart) extends from a main abstract parent class called **Chart**. While this approach is intuitive, we chose not to use it because it introduces tight coupling and could potentially result in a deep inheritance hierarchy, which would unnecessarily complicate the relationships between charts.

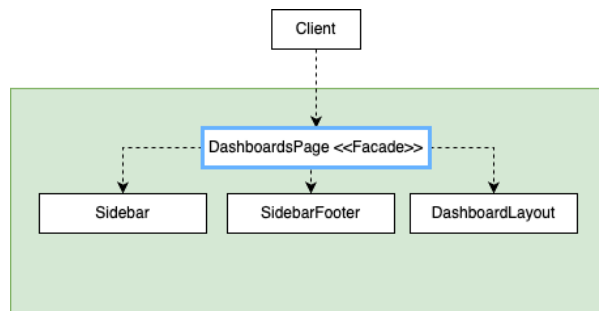
We chose **registry pattern instead**, as it promotes **loose coupling**, supports the **dynamic behaviour** we need, and avoids over-complicating the **IS-A** relationships between charts.

Wrapper design pattern



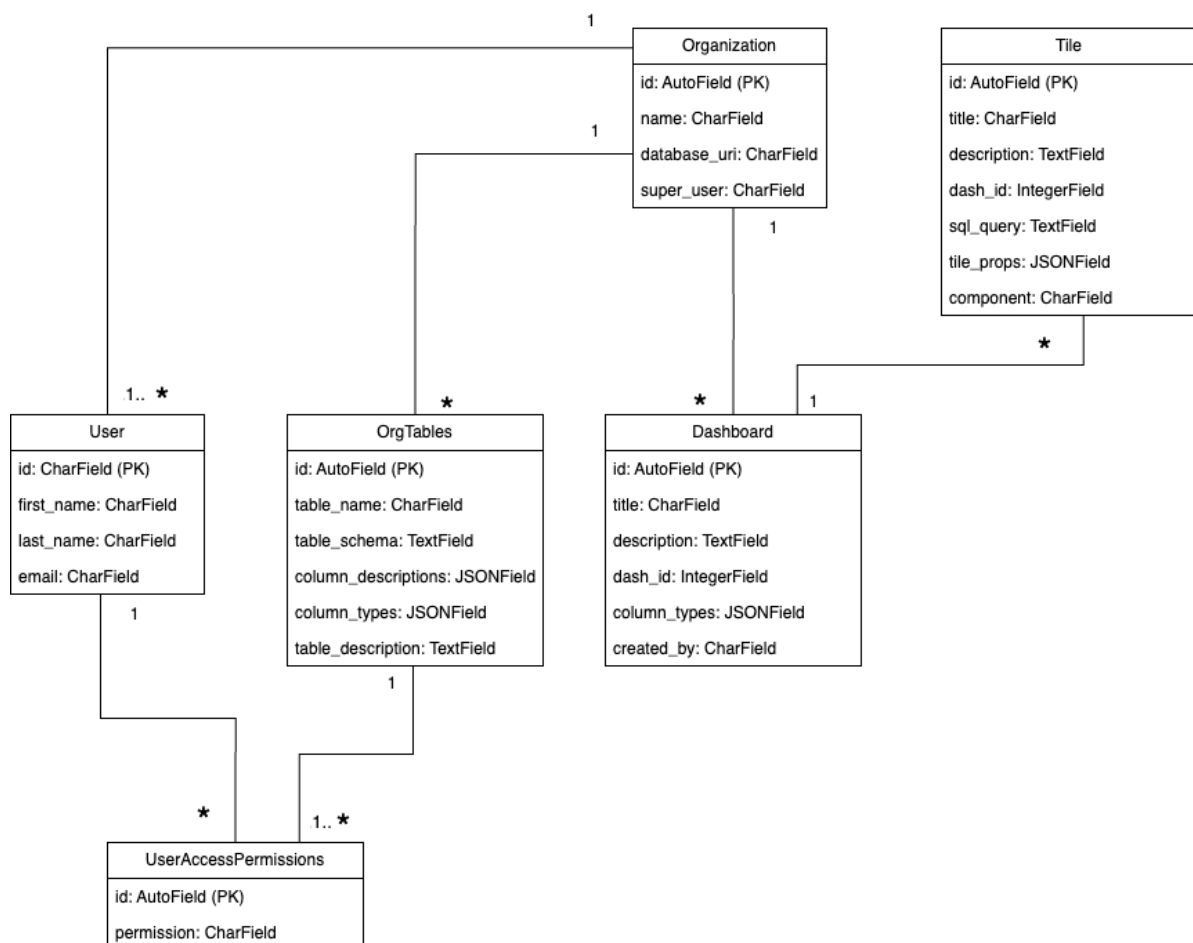
We decided to use the **wrapper design pattern** in React.js to maintain a **global state for authentication**. This approach centralises authentication logic, ensuring that authentication checks are consistent and accessible throughout all components in the application. This not only enhances maintainability but also aligns with React's design principles by promoting modularity and reusability.

Facade pattern



We used the Facade Design Pattern to simplify the construction of dashboard UI, side panel and other components. Instead of embedding all the logic for assembling these elements directly in the core code, we created a single function that acts as a high-level interface to handle the entire setup. This approach hides the complexity of the individual components, making the code cleaner and easier to maintain.

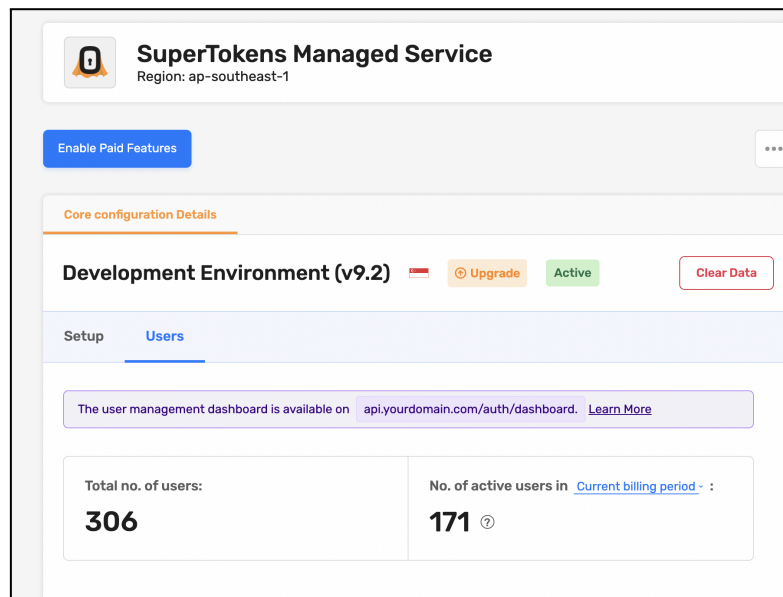
Database model UML



Finally, above are our database models. In our **Users** model, we do not store passwords. This enhances security as we rely only on Supertokens as the password manager, which securely handles password hashing, salting, peppering, ensuring safe password storage.

Marketing

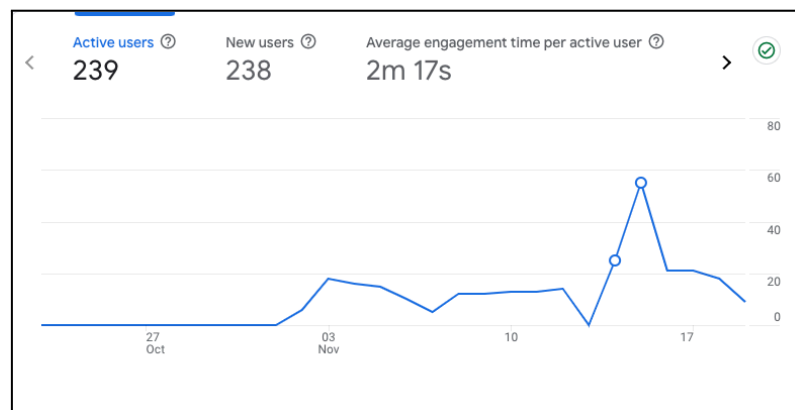
Eff BI User Analytics



Analytics for registered users in SuperTokens

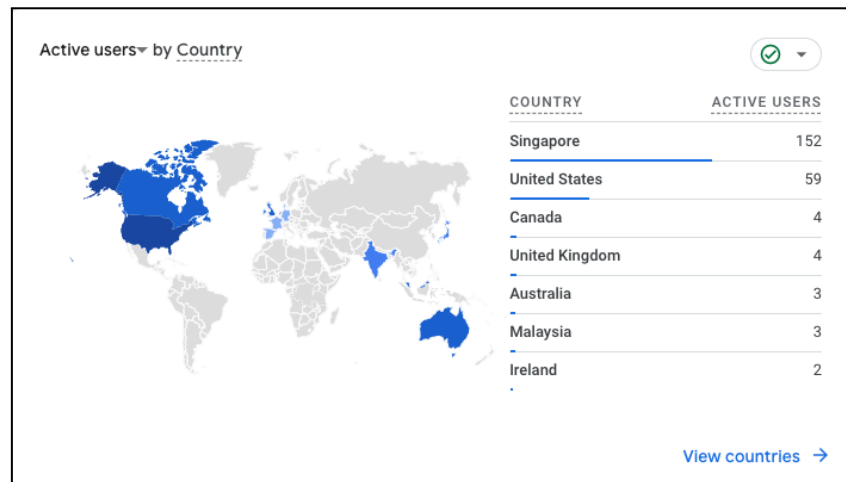
Above is the total number of signups on Eff BI. Although this does not capture the number of unique users, it can be used in conjunction with the number of active users to provide a rough picture of the unique number of people who have signed up.

We are also using Google Analytics to track the usage of our application.



Analytics for active Eff BI users

As of 20th November 2024, we have a total for **239 users**, 238 of which are active, making the active user percentage about **99%**.



Demographic of users by country

The majority of our users are from **Singapore**, followed by the **USA**. This is likely due to our marketing efforts primarily being targeted locally, particularly within our immediate network and the NUS community. Additionally, Suzanna and Prof Soo's posts about Eff BI on LinkedIn may have attracted international interest, like from their connections in the USA.

Plot rows		Search...		Rows per page: 10		Go to: 1	
<input type="checkbox"/>	Page path and screen class	Views	Active users	Views per active user	Average engagement time per active user		
<input checked="" type="checkbox"/>	Total	1,065 100% of total	146 100% of total	7.29 Avg 0%	4m 02s Avg 0%		
<input type="checkbox"/>	1 /	266	143	1.86	14s		
<input type="checkbox"/>	2 /dashboards	259	31	8.35	5m 21s		
<input type="checkbox"/>	3 /view-data	199	20	9.95	10m 22s		
<input type="checkbox"/>	4 /auth	118	29	4.07	2m 25s		
<input type="checkbox"/>	5 /access-permissions	53	9	5.89	3m 57s		
<input type="checkbox"/>	6 /auth/	44	16	2.75	21s		
<input type="checkbox"/>	7 /dashboards/1	31	4	7.75	5m 59s		
<input type="checkbox"/>	8 /profile	29	9	3.22	1m 48s		
<input type="checkbox"/>	9 /faq	17	6	2.83	1m 47s		
<input type="checkbox"/>	10 /settings/table-permissions	10	4	2.50	12s		

Number of views by page

When looking at **views by page**, our landing page (denoted by the "/") emerged as the most visited, followed by the dashboard and the view data page. They have a difference of 7

views, which highlights the landing page's effectiveness in drawing user attention and serving as the primary entry point to our application as the number of users from landing to dashboard page are almost equal.

Future plans and strategies

Given that the first iteration of our application primarily focused on PostgreSQL, we intend to expand support for other RDBMS such as MySQL, Oracle DB and MSSQL Server.

During a conversation with a senior engineer at Micron, we identified a crucial challenge: companies are hesitant to adopt new analytics platforms because they've already invested significant time building dashboards in tools like Tableau and PowerBI. To address this, we're developing integrations with these popular BI tools through their APIs, allowing companies to keep their existing dashboards while gaining the benefits of our platform.

The additional features require a lot of manpower to implement, which is why we aim to build an open-source community and leverage it to achieve a fast pace of development and utilisation by opening tickets for these features. During our STePS presentation, it was rewarding to find out that several SoC students and visitors were deeply interested in contributing and some even opened new issues on our [github](#).

Under the advice of Dr Suzanna, we also endeavour to increase the visibility of our project through conference participation. We will be submitting a paper for the World Wide Web Conference 2025. This would be an effective channel for feedback and a valuable opportunity to expand our open source community.

Insights gained from the project

One key lesson we learned was that integrating a large language model (LLM) is not just about getting it to work but ensuring its outputs are reliable and aligned with user needs. While we managed to set up a basic LLM pipeline within three weeks, the real challenge was refining the pipeline to reduce hallucinations that led to errors being thrown. For example, our frontend charts required specific data formats, such as a pie chart needing segment percentages rather than x-y value pairs. This meant the LLM had to produce correctly formatted data tailored to the chart type, which required significant iteration and testing.

Initially, we were satisfied as long as the system produced output that did not lead to an error. However, during user testing, particularly with business users, we noticed that they often posed vague or general questions. The LLM struggled to provide relevant answers and

frequently returned errors. This pushed us to enhance the pipeline by introducing additional agents like a query relevance checker, a SQL query validator, and a looping mechanism to regenerate SQL queries if the initial ones were irrelevant. Incorporating these steps, along with chain-of-thought reasoning and few-shot prompting, significantly improved the system's performance and usability.

Another key takeaway was to experiment fast and fail fast. From a frontend perspective, the challenge of developing a fully functional and demonstrable open-source application within a short time frame was both demanding and rewarding. We gained insights into abstracting concepts and working with streaming responses from POST requests. Initially, we tried D3.js for visualisations, but found it too low-level for our needs and quickly pivoted away. Switching to ApexCharts proved to be a game-changer, offering an excellent abstraction layer and extensive customization options.

In terms of project management, one major takeaway from this project was the importance of effective goal prioritisation while staying adaptable to change. Early on, we frequently debated which features to implement and which to defer. Categorising features into must-haves, nice-to-haves, and non-essentials provided clarity and helped streamline our focus. This structure minimised unnecessary back-and-forth and kept the team aligned. However, rigidly adhering to these priorities sometimes proved counterproductive. Unexpected challenges, like LLM outputs failing in certain scenarios, required us to make tough decisions, such as dropping features, simplifying their implementation, or pivoting entirely. At first, compromising on our vision of "perfect" features felt frustrating. Over time, we came to appreciate that delivering a functional product by the sprint deadline was more valuable than chasing perfection. This experience underscored the importance of balancing project goals with flexibility. Development is as much about practical trade-offs as it is about creative problem-solving. Ultimately, building a successful product isn't just about implementing the best features; it's about making thoughtful decisions to ensure progress, usability, and value for the end user.

Contact Report

To document the final meeting with Suzanna and one last user testing: [Contact Report 3](#)