Project Context - Design Document

Overview

The purpose of this documentation is to document all the relevant design decisions being made regarding **Project Context**. The purpose of this application is to provide small teams the capabilities of giving an LLM the necessary context needed to be able to answer Junior Engineers questions effectively and efficiently. The goal is to reduce the amount of time it takes for a Junior to get the necessary answer without digging through years of documentation and thousands of lines of code in order to get their answer, while simultaneously saving Senior Engineers the time from answering repeated questions.

Backend

Data Ingestion

Data Ingestion Flow

Could not identify user

- **Data Sources**
  - Messaging Systems: **Slack, Discord, Webex**
  - Code Files: **BitBucket, GitHub**
  - Documentation: **Bitbucket, GitHub, Confluence**
- **Data Ingestion Flow**
  - See diagram above for high level plan of handling data ingestion
  - We will use a directory and have users initially manually add files to this directory. Once data ingestion complete, these files can be moved to a separate **processed** directory, indicating we ingested it and stored in Vector DB
- **Filtering Of Bad Data**
  - Not all data (specifically Slack, Discord, Webex messages) is necessarily ***quality*** data
  - There should be some sort of filtering algorithm that we go through and create that will help determine whether or not something is **true** or **false**
    - EX) “The answer is 7” → two messages later by same user → “I was mistaken, its 9”
- **Automated Data Ingestion**
  - In the long run, we would like the overall data ingestion process to be automated
  - This can be done via some sort of Webhooks from the data source (if they have any) or some sort of nightly cron job that will go through and look for any new relevant files, messages, documentation that correspond to the particular workspace/project
  - The big thing to note here is that the automated data ingestion will need some sort of “diff” algorithm (i.e this code file was updated, so I will update the record in the vectorized database accordingly)

Data Storage

- **Vectorized Database**
  - *Purpose:* This will be store all embeddings corresponding to user messages, code files, and documentation that will be utilized within our RAG flow
  - *Options:*
    - **1)** Chroma
      - Simple setup, however, not for production as scale
    - **2)** Qdrant
      - Bit more complex setup, but production ready and has some docker support
  - *Collections*

        - In order to keep data separate, we will have **one Collection for each Project**
    - *Models*
        - Unlike SQL, there is no traditional schema that needs to be defined in Vector DB, but we should still standardize the convention of the data we are storing
        - **DocumentMetadata**
            - **required fields**
            - project_id
            - source_type
            - created_at
            - **optional git fields**
            - file_path
            - git_commit
            - language
            - **optional docs field**
            - page_title
            - page_url
            - space_key
            - **optional message fields**
            - channel
            - thread_id
            - message_timestamp
- **Relational Database (PSQL)**
    - *Purpose:* This will store all relevant database entities that **are not** specific to the actual content being used in RAG capabilities
    - *Models*
        - **Company**
            - ***Overview:*** Model to store general information about company (i.e company preferences, name, etc)
            - has many Spaces
        - **Space**
            - ***Overview:*** Model to store general information about a Space/Sector within the Company (i.e description)
            - belongs to Company

    - has many Teams
- **Team**
    - ***Overview:*** Model to store meta data about a particular team (team name, what space they are in, what users are on the team, etc)
    - belongs to Space
    - has many TeamProjects
    - has many TeamMembership
- **User**
    - ***Overview:*** Model to store generic information about User who is using this application, and correspond to selected preferences)
    - has many TeamMembership
    - has one UserPreferences
- **Project**
    - ***Overview****:* Model to store general information about a specific project that some teams may be working on
    - has many TeamProjects
    - has many DataSources
    - references VectorDB Collection
- **TeamMembership**
    - ***Overview:*** Model to store relationship between what users are on what teams
    - belongs to Team
    - belongs to User
- **TeamProjects**
    - ***Overview:*** Model to store relationships between what teams are on what projects
    - belongs to Team
    - belongs to Project
- **DataSource**
    - ***Overview:*** Model to store information about collected information and its corresponding source (i.e type, name(Confluence, GitHub, BitBucket), last synced time, sync status, etc)
    - belongs to Project
    - has many IngestionJobs
- **IngestionJob**

        - ***Overview:*** Model to store information about a particular job that was run/being run to sync data from a particular data source
        - belongs to DataSource
    - **UserPreferences**
        - ***Overview:*** Model to store user preferences that they may have selected (priority project, theme, etc)
        - belongs to User

Security & Authentication

- One of the biggest fundamental goals of this application is to ensure companies can securely use our application and not be worried that their data is being viewed by restricted people
- That is why we are going to allow users to run our application via a **docker-compose.yml** file that they can setup custom configurations (open AI keys, anthropic keys, limits)
- The only real authentication required will be via JWT Tokens for individual users, allowing for Company Admins to setup User profiles, correspond them to a particular space

Model Accuracy & Citations

- In order to ensure the model is accurate, we should have some sort of functionality in place that will cite the source(s) in which are used when responding back to end user during our RAG Pipeline
- This will instill some confidence in the end user that the information provided by the LLM has proof of where it got it's answer, while also providing necessary links to user to go and dig deeper themselves if need be
- We should also implement some logic where if the question is too generic, or the model is unsure about the question, that we are able to not hallucinate an answer, but actually prompt user for more information or specify the assumption being made

LLM Model Selection & Abstraction

- There is a couple of options we can consider for implementing some sort of model abstraction
- **1) Abstract Classes**

    - We can create an abstract class **Provider** and then create individual Provider classes for each potential model (Llama, OpenAI, Claude, etc)
    - An abstract method would also be created for **calling** the specific provider (each provider expects to be called in different ways)
- **2) LLama Index**
    - We can use the **Settings.llm** to configure the LLM being utilized globally, and all queries that we generate using LLamaIndex will use our selected LLMs

API Design

- **Authentication**
    - **POST** /api/auth/register
    - **POST** /api/auth/login
    - **POST** /api/auth/logout
    - **POST** /api/auth/refresh-token
- **User Management**
    - **GET** /api/users → (*admin only) get all users*
    - **GET** /api/users/{id}
    - **PATCH** /api/users/{id}
    - **DELETE** /api/users/{id} → *(admin only) delete user*
- **Space Management**
    - **GET** /api/spaces → *get all spaces*
    - **GET** /api/spaces/{id} → *get space details*
    - **POST** /api/spaces → (*admin only) add new space*
    - **PATCH** /api/spaces/{id} → (*admin only) update space*
    - **DELETE** /api/spaces/{id} → (*admin only) delete existing space*
    - **GET** /api/spaces/{id}/teams → *get all teams in space*
- **Project Management**
    - **GET** /api/projects → *get all projects*
    - **GET** /api/projects/{id} → *get specific project details*
    - **POST** /api/projects → *(admin only) create a project*
    - **PATCH** /api/projects/{id} → *(admin only) update a project*
    - **DELETE** /api/projects/{id} → (*admin only) delete a project*

- **GET** /api/projects/{id}/stats → *get project stats (document count, last indexed, etc)*
- **POST** /api/projects/{id}/teams → *(admin only) add team to project*
- **DELETE** /api/projects/{id}/teams/{id} → *(admin only) remove team from project*
- **GET** /api/projects/{id}/teams → *get list of teams corresponding to project*

- **Team Management**
  - **GET** /api/teams → *get all teams*
  - **POST** /api/teams → (*admin only) create team*
  - **GET** /api/teams/{id} → *get team details*
  - **PATCH** /api/teams/{id} → (*admin only) update team*
  - **DELETE** /api/teams/{id} → *(admin only) delete team*
  - **GET** /api/teams/{id}/members → *get team's users (members)*
  - **POST** /api/teams/{id}/members → (*admin only) add user to team*
  - **DELETE** /api/teams/{id}/members/{user_id} → (*admin only) remove user from team*
  - **PATCH** /api/teams/{id}/members/{user_id} → *(admin only)* update users role
  - **GET** /api/teams/{id}/projects → *get list of team's projects they are assigned to*
- **Company Management**
  - **GET** /api/company → *get company info*
  - **PATCH** /api/company → *(admin only) update company settings*
  - **GET** /api/company/stats → *get overall stats for company (number of users, projects, spaces, etc)*
- **Data Source Management**
  - **GET** /api/projects/{project_id}/data-sources → *list data sources*
  - **POST** /api/projects/{project_id}/data-sources → *(admin only) add data source*
  - **GET** /api/data-sources/{id} → *get data source details*
  - **PATCH** /api/data-sources/{id} → *update data source config*
  - **DELETE** /api/data-sources/{id} → *(admin only) delete data source*
  - **POST** /api/data-sources/{id}/sync → *trigger manual sync*
  - **POST** /api/data-sources/{id}/test-connection → *test connection to data source*
  - **GET** /api/data-sources/{id}/sync-history → *get data source sync history*
- **File Upload**
  - **POST** /api/projects/{project_id}/upload *# upload files*
  - **POST** /api/projects/{project_id}/upload-batch *# upload multiple files*

Testing Plan

- Testing will be a bit difficult without having actual access to any large scale code bases
- I believe a good way to test this is by using our Budget Application project and having the integration with GitHub be tested this way (making sure we can see specific commits and files with explanations)
- It also would be very helpful if we could find code base on GitHub and use their code and documentation to help train a user
- This type of flow could easily be adopted in the **OpenSource** community to allow for new developers to easily jump on to a new code base and get up to speed relatively quickly

Frontend