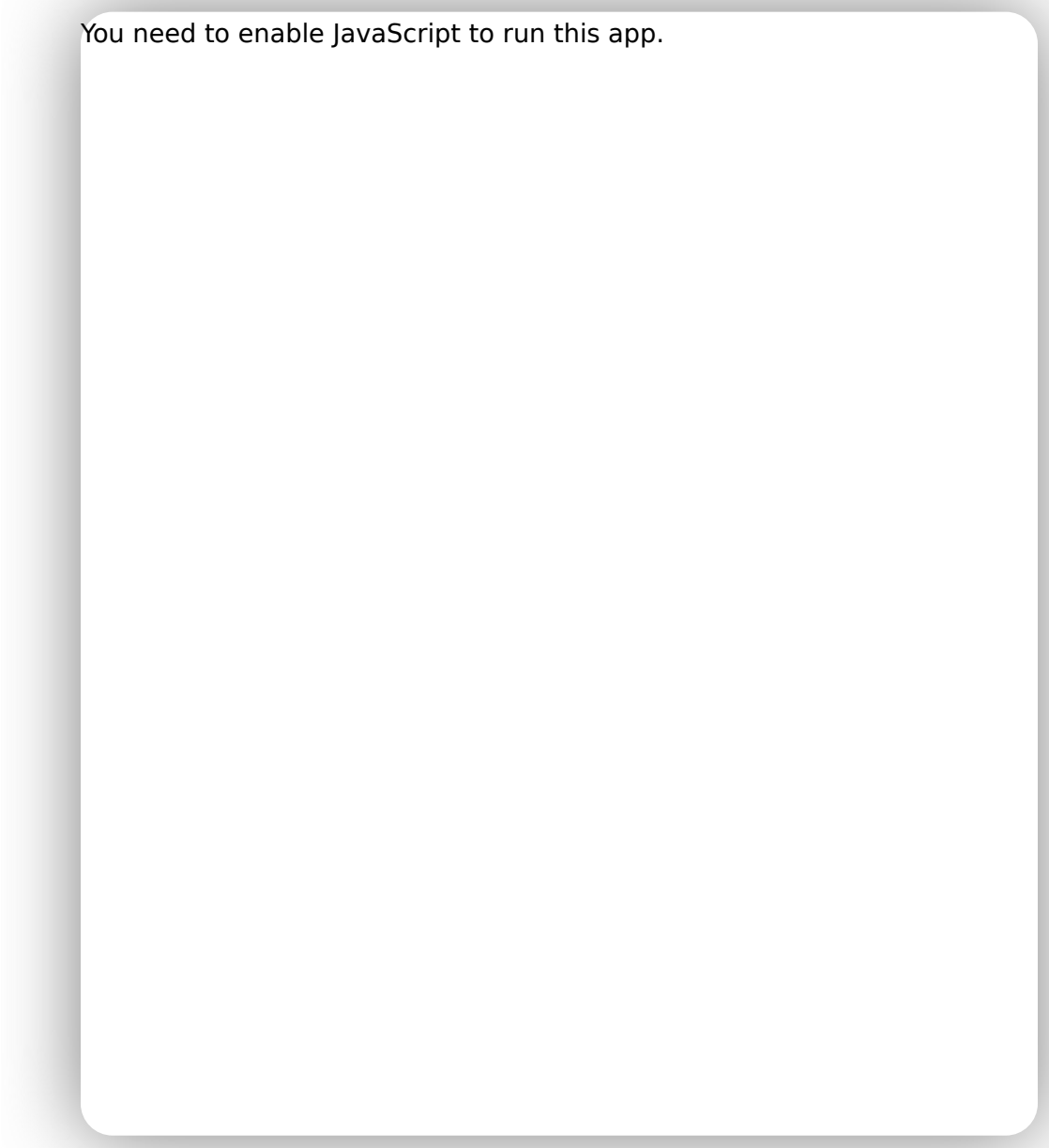
Overview

Casibase is a Open-Source Domain Knowledge Database & IM & Forum Software powered by ChatGPT.



Casibase features

1. Front-end and back-end separate architecture, developed by Golang,

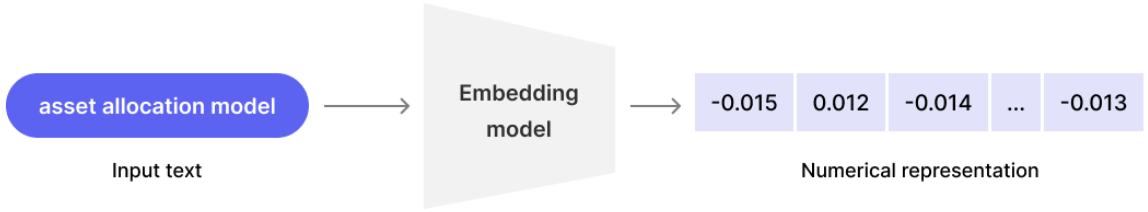
Casibase supports high concurrency, provides web-based managing UI and supports multiple languages(Chinese, English).

2. Casibase supports third-party applications login, such as GitHub, Google, QQ, WeChat, etc., and supports the extension of third-party login with plugins.
3. Based on embedding and prompt engineering for knowledge management, Casibase supports customized embedding methods and language models.
4. Casibase supports integration with existing systems by db sync, so users can transition to Casibase smoothly.
5. Casibase supports mainstream databases: MySQL, PostgreSQL, SQL Server, etc., and supports the extension of new databases with plugins.

How it works

Step 0 (Pre-knowledge)

Casibase's knowledge retrieval process is based on Embedding and Prompt engineering, so it is highly recommended that you take a brief look at how Embedding works. An introduction to Embedding.



Step 1 (Importing Knowledge)

To get started with Casibase, users need to follow these steps to import knowledge and create a domain-specific knowledge database:

1. Configure Storage: In the Casibase dashboard, users should first configure the storage settings. This involves specifying the storage system to be used for storing knowledge-related files, such as documents, images, or any other relevant data. Users can choose from a variety of storage options based on their preferences and requirements.

2. Upload Files to Storage: Once the storage is set up, users can proceed to upload files containing domain-specific knowledge into the configured storage system. These files can be in various formats, such as text documents, images, or structured data files like CSV or JSON.

3. Select Embedding Method for Knowledge Generation: After the files are uploaded, users have the option to choose the embedding method for generating knowledge and corresponding vectors. Embeddings are numerical representations of textual or visual content, which facilitate efficient similarity search and data analysis.

> TIP
>
> How knowledge is embedded?
>
> - For textual data: Users can choose from various embedding methods, such as Word2Vec, GloVe, or BERT, to convert the textual knowledge into meaningful vectors.
>
> - For visual data: If the uploaded files contain images or visual content,

users can select image embedding techniques like CNN-based feature extraction to create representative vectors.

- More methods coming soon...

By following these steps, users can populate their domain knowledge database with relevant information and corresponding embeddings, which will be used for effective searching, clustering, and retrieval of knowledge within Casibase. The embedding process allows the system to understand the context and relationships between different pieces of knowledge, enabling more efficient and insightful knowledge management and exploration.

Step 2 (Retrieving Knowledge)

After importing your `domain knowledge`, Casibase transforms it into `vectors` and stores these vectors in a `vector database`. This vector representation enables powerful functions like `similarity search` and `efficient retrieval of related information`. You can quickly find relevant data based on context or content, enabling advanced querying and uncovering valuable insights within your domain knowledge.

Step 3 (Building the Prompt)

Casibase performs a similarity search on the stored knowledge vectors to find the closest match to the user's query. Using the search results, it creates a `prompt template` to frame a specific question for the `language model`. This ensures accurate and contextually relevant responses, delivering comprehensive answers based on the domain knowledge in Casibase.

Step 4 (Achieving the Goal)

At this stage, using Casibase, you have successfully acquired the knowledge you sought. Through the innovative combination of domain knowledge transformed into vectors and powerful language models like ChatGPT, Casibase provides you with accurate and relevant responses to your inquiries. This enables you to efficiently access and utilize the domain-specific information stored in Casibase, meeting your knowledge requirements with ease.

Step 5 (Optional Fine-tuning)

If you find that the results are not entirely satisfactory, you can try to get better results by doing the following:

- Tweaking Language Model Parameters
- Asking multiple questions
- Optimizing the original files

By utilizing these fine-tuning options, you can improve the efficiency of your knowledge management in Casibase, ensure that the system is better aligned with your goals, and provide more accurate and insightful information.

> ⓘ HINTS
>
> Other ways to optimize results (may require source code changes):
>
> - Updating `Embedding` Results: Refine the knowledge representation by adjusting the embedding results of your domain knowledge.

- Modifying `Prompt` Templates: By customizing the prompts, you can elicit more precise responses from the language model.
- Exploring Different `Language Models`: Experiment with different models to find the one that best suits your requirements for generating responses.

Online demo

Casibase

Here is an online demo deployed by Casbin.

- Casibase official demo

Global admin login:

- Username: `admin`
- Password: `123`

Architecture

Casibase contains 4 parts:

Name	Description	Language	Source code
Frontend	User interface for the casibase application	JavaScript + React	https://github.com/casbin/casibase/tree/master/web

Name	Description	Language	Source code
Backend	Server-side logic and API for casibase	Golang + Beego + MySQL	https://github.com/casbin/casibase
AI Model	Artificial intelligence model	Python + OpenAI	https://github.com/casbin/casibase/tree/master/ai
Knowledge Base	Storage for casibase application domain knowledge	pgvector	https://github.com/casbin/casibase/tree/master/ai

Core Concepts

As Casibase's user, you should get familiar with at least 4 core concepts: `Provider`, `Storage`, `Chat` and `Vector`.

Provider

Providers are the backbone of Casibase, offering essential services and integration with external systems. The Provider class definition is shown as follows:

```
type Provider struct {
    Owner       string `xorm:"varchar(100) notnull pk"
json:"owner"`
    Name        string `xorm:"varchar(100) notnull pk" json:"name"`
    CreatedTime string `xorm:"varchar(100)" json:"createdTime"`

    DisplayName  string `xorm:"varchar(100)" json:"displayName"`
    Category     string `xorm:"varchar(100)" json:"category"`
    Type         string `xorm:"varchar(100)" json:"type"`
    ClientId     string `xorm:"varchar(100)" json:"clientId"`
    ClientSecret string `xorm:"varchar(2000)" json:"clientSecret"`
    ProviderUrl  string `xorm:"varchar(200)" json:"providerUrl"`
}
```

TIP

There are two primary types of providers in Casibase:

- **Storage Provider:**The Storage Provider facilitates the storage and retrieval of data within Casibase. It supports various storage options, including:
  - AWS
  - Azure
  - Local File System
- **AI Provider:**The AI Provider is responsible for handling AI-related tasks and services in Casibase. It supports multiple AI models and technologies, including:
  - OpenAI
  - ChatGLM
  - InternLM

Vectors

Vectors in Casibase represent numerical representations of different types of data. These vectors enable efficient processing and analysis of information. Some of the vector types available are:

- Text Vector
- Image Vector
- ... (other vector types)

The Provider class definition is shown as follows:

```
type Vector struct {
```

Chats

Chats are at the core of interactive communication between users and the AI models in Casibase. They consist of three essential components:

- Question: The user's input or query, seeking information or assistance.
- Query Prompt: A formatted version of the user's question, prepared for processing by the AI models.
- Answer: The AI-generated response to the user's question, providing relevant information or solutions.

The Chat class definition is shown as follows:

```
type Chat struct {
    Owner        string   `xorm:"varchar(100) notnull pk"
json:"owner"`
    Name         string   `xorm:"varchar(100) notnull pk"
json:"name"`
    CreatedTime  string   `xorm:"varchar(100)" json:"createdTime"`
    UpdatedTime  string   `xorm:"varchar(100)" json:"updatedTime"`

    DisplayName  string   `xorm:"varchar(100)" json:"displayName"`
    Category     string   `xorm:"varchar(100)" json:"category"`
    Type         string   `xorm:"varchar(100)" json:"type"`
    User1        string   `xorm:"varchar(100)" json:"user1"`
    User2        string   `xorm:"varchar(100)" json:"user2"`
    Users        []string `xorm:"varchar(100)" json:"users"`
    MessageCount int      `json:"messageCount"`
}
```

Embedding

Embedding is the process of transforming various types of data, such as text and images, into dense vector representations. This step is crucial for facilitating efficient data processing and analysis within Casibase.

TIP

- By embedding, the questions in chat and the knowledge files in storage will be turned into vectors and used in the next step of knowledge search.
- Casibase's default embedding method is provided by OpenAI at a rate of up to three calls per minute. We propose to minimize the knowledge file coupling as much as possible to facilitate embedding and further processing.

Server Installation

Requirements

OS

All major operating systems including Windows, Linux and macOS are supported.

Environment

- Go 1.17+
- Node.js LTS (18)
- Yarn 1.x

> INFO
>
> The use of Casibase is divided into two steps:
>
> - step1: Deploy and run Casdoor
> - step2: Deploy and run Casibase
>
> We strongly suggest you use Yarn 1.x to run & build Casdoor&Casibase frontend, using NPM might cause UI styling issues, see more details at: casdoor#294

> CAUTION
>
> For **Chinese** users, in order to download the Go dependency packages

> successfully, you need to use a Go proxy by Configuring the GOPROXY environment variable. We strongly recommend: https://goproxy.cn/

Database

Casibase uses XORM to talk to the database. Based on Xorm Drivers Support, Casibase currently provides support for following databases:

- `MySQL`
- `MariaDB`
- `PostgreSQL`
- `CockroachDB`
- `SQL Server`
- `Oracle`
- `SQLite 3`
- `TiDB`

Download

The source code of Casibase is hosted at GitHub: https://github.com/casbin/casibase. Both the Go backend code and React frontend code are inside the single repository.

Name	Description	Language	Source code
Frontend	Web frontend UI for Casibase	JavaScript + React	https://github.com/casbin/casibase/tree/master/web

Name	Description	Language	Source code
Backend	RESTful API backend for Casibase	Golang + Beego + XORM	https://github.com/casbin/casibase

Casibase supports `Go Modules`. To download the code, you can just simply clone the code via git:

```
cd path/to/folder
git clone https://github.com/casbin/casibase
```

Configuration

Configure Database

Casibase supports mysql, mssql, sqlite3, postgres. Casibase uses mysql by default. If you use another database, please modify the import package in object/adapter.

MySQL

Casibase will store its users, nodes and topics information in a MySQL database named: `casibase`. If the database does not exist, it needs to be created manually. The DB connection string can be specified at: https://github.com/casbin/casibase/blob/master/conf/app.conf

```
driverName = mysql
dataSourceName = root:123456@tcp(localhost:3306)/
```

PostgreSQL

Since we must choose a database when opening Postgres with xorm, you should prepare a database manually before running Casibase.

Let's assume that you have already prepared a database called `casibase`, then you should specify `app.conf` like this:

```
driverName = postgres
dataSourceName = "user=postgres password=postgres host=localhost
port=5432 sslmode=disable dbname=casibase"
dbName =
```

> INFO
>
> For PostgreSQL, make sure `dataSourceName` has non-empty `dbName` and leave the standalone `dbName` field empty like the above example.

CockroachDB

You can also use cockroachdb with postgres driver. It has same configuration as postgreSQL.

```
driverName = postgres
dataSourceName = "user=postgres password=postgres host=localhost
port=5432 sslmode=disable dbname=casibase
serial_normalization=virtual_sequence"
dbName =
```

> INFO

> For CockroachDB, don't forget to add `serial_normalization=virtual_sequence` to the `dataSourceName` like the above example. otherwise you will get error regarding existed database, whenever the service started or restarted. Notice, this must be added before the database created.

Sqlite3

You should specify `app.conf` like this:

```
driverName = sqlite
dataSourceName = "file:casibase.db?cache=shared"
dbName = casibase
```

Run

There are currently two methods to start, you can choose one according to your own situation.

> ⚠ CAUTION
>
> **Casibase** requires **Casdoor** to provide access control and some back-end services, so you must make sure **Casdoor** is running properly before running **Casibase**.
>
> How to install and run Casdoor:
>
> - Casdoor Installation

Development mode

Backend

Casibase's Go backend runs at port 14000 by default. You can start the Go backend with the following command:

```
go run main.go
```

After the server is successfully running, we can start the frontend part.

Frontend

Casibase's frontend is a very classic Create-React-App (CRA) project. It runs at port `13001` by default. Use the following commands to run the frontend:

```
cd web
yarn install
yarn start
```

Visit: `http://localhost:13001` in your browser. Log into Casibase dashboard with the default global admin account: `built-in/admin`

```
admin
123
```

Production mode

Backend

Build Casibase Go backend code into executable and start it.

For Linux:

```
go build
./casibase
```

For Windows:

```
go build
casibase.exe
```

Frontend

Build Casibase frontend code into static resources (.html, .js, .css files):

```
cd web
yarn install
yarn build
```

Visit: `http://localhost:14000` in your browser. Log into Casibase dashboard with the default global admin account: `built-in/admin`

```
admin
123
```

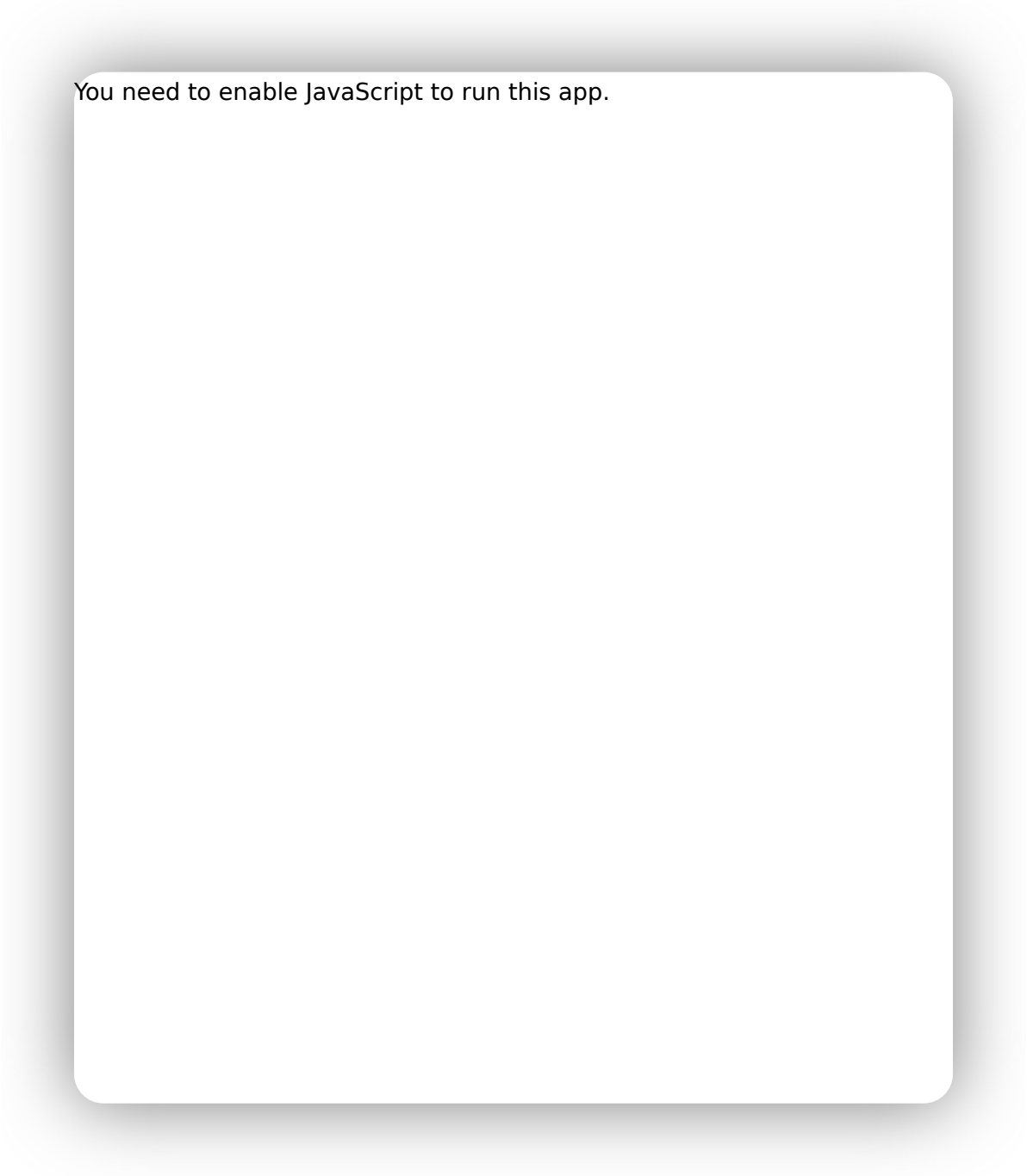
TIP

To use another port, please edit `conf/app.conf` and modify `httpport`, then restart the Go backend.

Casdoor-SSO

Introduction

Casdoor is a powerful and lightweight open-source Identity Access Management (IAM) / Single-Sign-On (SSO) server. It's developed and maintained by **Casbin**.



Casdoor serves both the web UI and the login requests from the application users.

Features

- **Single-Sign-On (SSO)**: Sign in to multiple applications with one set of login credentials.
- **Social Login**: Sign in with GitHub, Google, etc.
- **Integrated Provider Management**: Manage all your providers in one place.
- **Authentication**: Verify the identity of your users.

TIP

Casibase manages third-party service providers through Casdoor:

- **Storage**: Manage your storage providers, such as AWS, Azure, etc.
- **AI**: Manage your chat providers, such as OpenAI, ChatGLM, etc.