Group 75: Voco, make your memory more solid

Member Contribution

Student Name	Student ID	Contribution
Qi Xiaojian	3036382765	React native development FastAPI Development Database implement Web Crawler Data collection Report
Su Shiyun	3036410354	React native development Database Design FastAPI Development Video
Tian Chen	3036409941	Software Design Database Design Web Crawler Data collection Video
Zuo Shanyin	3036381709	Backgroun reasearch Software Design Database Design Video
Qiu Xuechen	3036381814	Background research Web Crawler Data collection Report FastAPI Development

Chen Jinming	3036381591	Background research Report FastAPI Development Overall prototype design

Background

The goal of our team is to develop an APP for memorizing IELTS words. Word memorization applications have always been a hot field in the English learning market, especially with the increasing demand for IELTS. The development direction of such apps has gradually changed from simple vocabulary memorization to comprehensive interaction, and personalized and scenario-based learning, emphasizing the improvement of user experience and practical application capabilities. The main reason for our development of the IELTS word recitation APP is that IELTS requires a large number of professional vocabulary and its flexible application in specific contexts, and the traditional word memorization model makes it challenging to meet the expectations of modern learners for efficient, interesting, and interactive learning.

Analysis For Competitive Products

To this end, we analyzed the existing competitive products in the market, including the functions of these competitive products and their user data, and analyzed their user usage on the App Store.

BuBeiDanCi: There are many ways to assist users in memory, such as the memory mode in the context, including spelling, example sentence filling in the blank, etc. However, the memory curve module of the words is not reasonable, and users often need to review the content of their memory in a large range. In many cases, users are already familiar with the content of the words, but they still need to memorize the learned words in a large range. Besides, they may have to struggle in many practices.

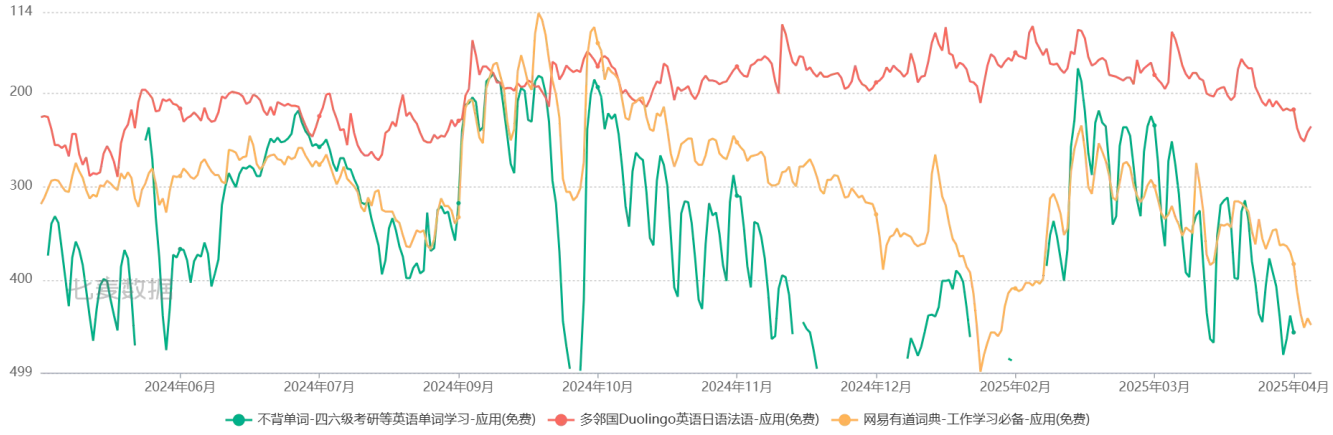
Duolingo: Provides comprehensive training in listening, speaking, reading, and writing, which is the weakest point of other apps. Moreover, Duolingo is more inclined to understand the language itself than just the exam. Its social features are strong, and users can communicate and interact with other learners around the world. But Duolingo does not

have a memory for IELTS-specific words, it is essentially a tool for learning the entire language system. Its practice mode is also too simple, and can not meet the requirements of the difficult IELTS test.

Youdao: Youdao is a dictionary software. However, it still has the function of a word notebook and can also help users memorize words. However, the problem with Youdao lies in that they do not apply a reasonable memory curve. The way to memorize words in Youdao is based on the users' self-driven approach rather than through notifications. In addition, adding a word book on Youdao is overly cumbersome, forcing users to spend a lot of time on word collection

Product Data

According to the statistics and survey of the list data of Qimai data, including global downloads and other data, we can find that Duolingo has the largest audience, but it also reflects that Duolingo's emphasis on IELTS words is insufficient. This is also the focus of our subsequent application.



Our motivation

In light of these limitations, our motivation is to create a product that strategically integrates the most effective elements of these competitors while addressing their pain points. Our app will:

- Prioritize IELTS-specific vocabulary, categorized by topics and test sections.
- Apply a scientifically validated memory curve (e.g., spaced repetition algorithms like SM2) to optimize review timing and retention.
- Incorporate interactive and scenario-based learning modules, including listening and speaking exercises tailored to real IELTS situations.

- Support custom wordbook uploads, allowing learners to personalize their vocabulary database effortlessly.
- Provide instant feedback and adaptive learning paths, enhancing learner engagement and learning outcomes.

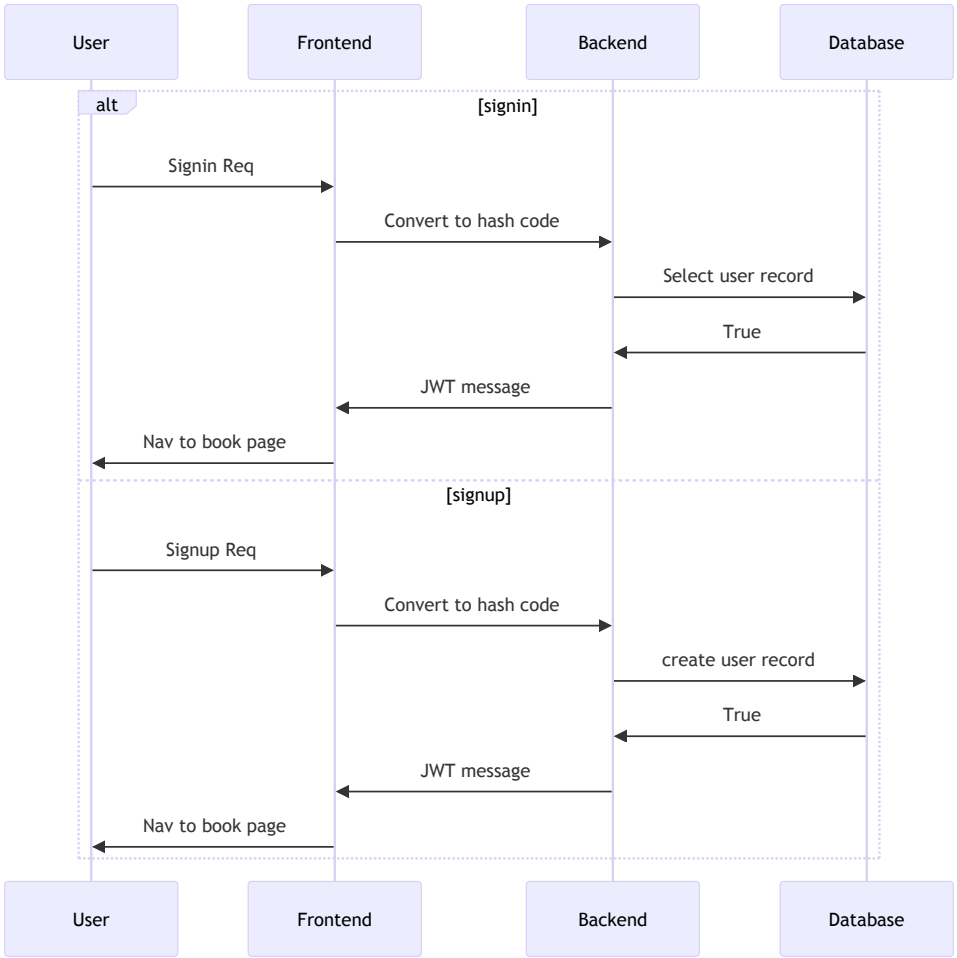
By focusing on these user-centered improvements, we aim to bridge the gap between general language learning apps and the targeted needs of IELTS test takers, offering a more efficient, personalized, and engaging learning experience.

Design

Firstly, we design a basic logic prototype of our software:

Prototype Design

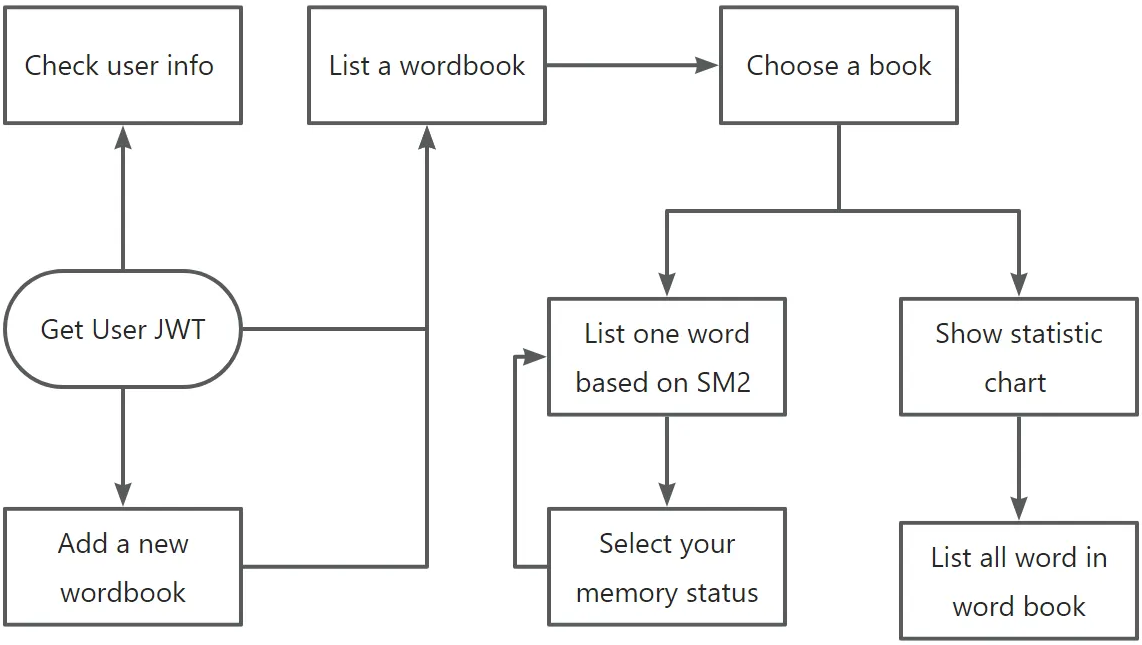
Signin & Signup



Our user management based on JWT, JWT(JSON Web Token) is a technology to record user login information. It is a very popular mechanism in stateless authentication. When the user signin or signup, the server will generate a JSON with username and login timestamp, then transfer this data to the frontend. After this, user should take a HTTP token Authorization connecting to the backend.

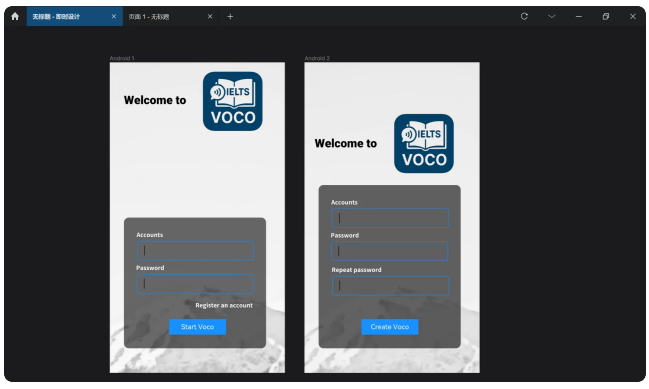
Service logic

This is our Service logic design

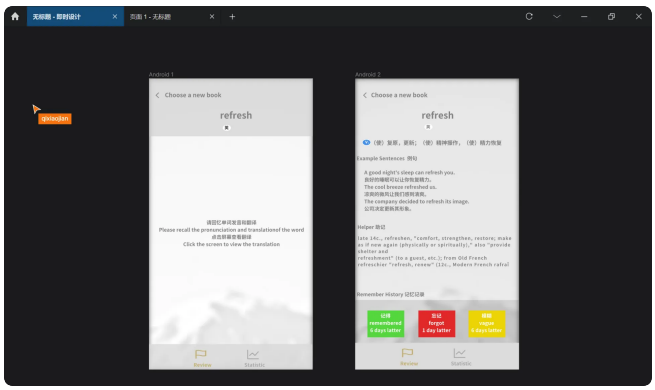


Frontend Design

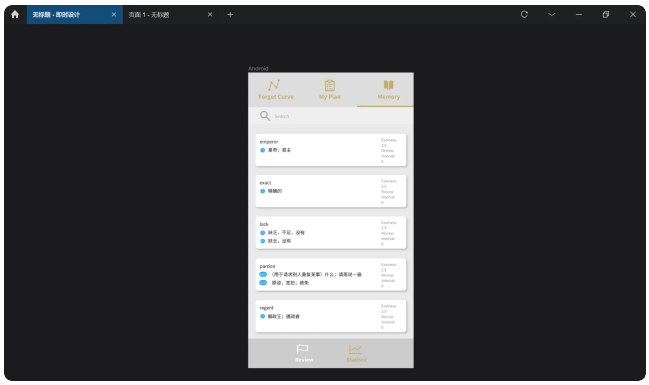
We use JS Design to design a basic prototype of our project, which is a design software like Figma. Our design style is light and perceptual, with the above function.



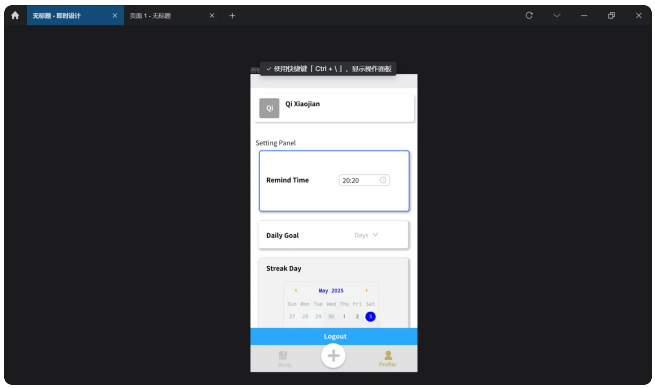
User Login



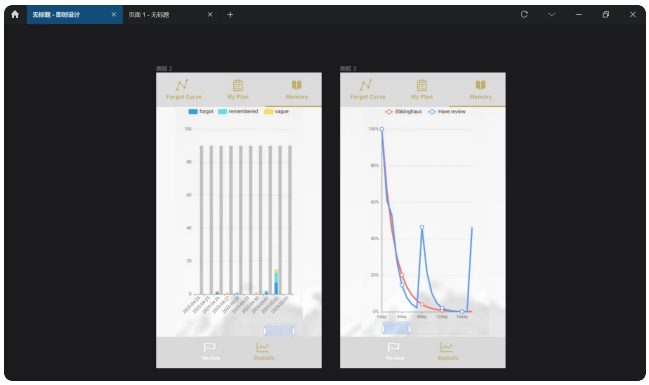
Word Page



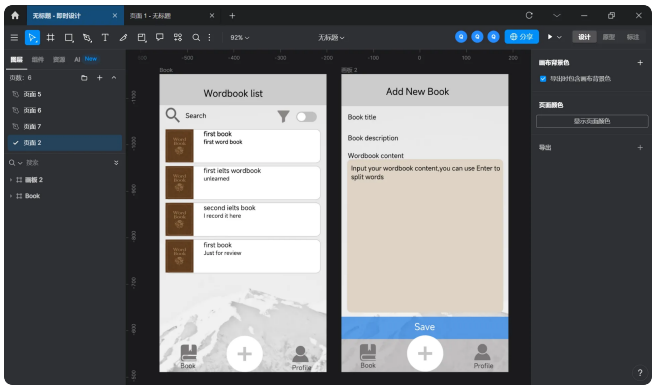
User Profile



Word list



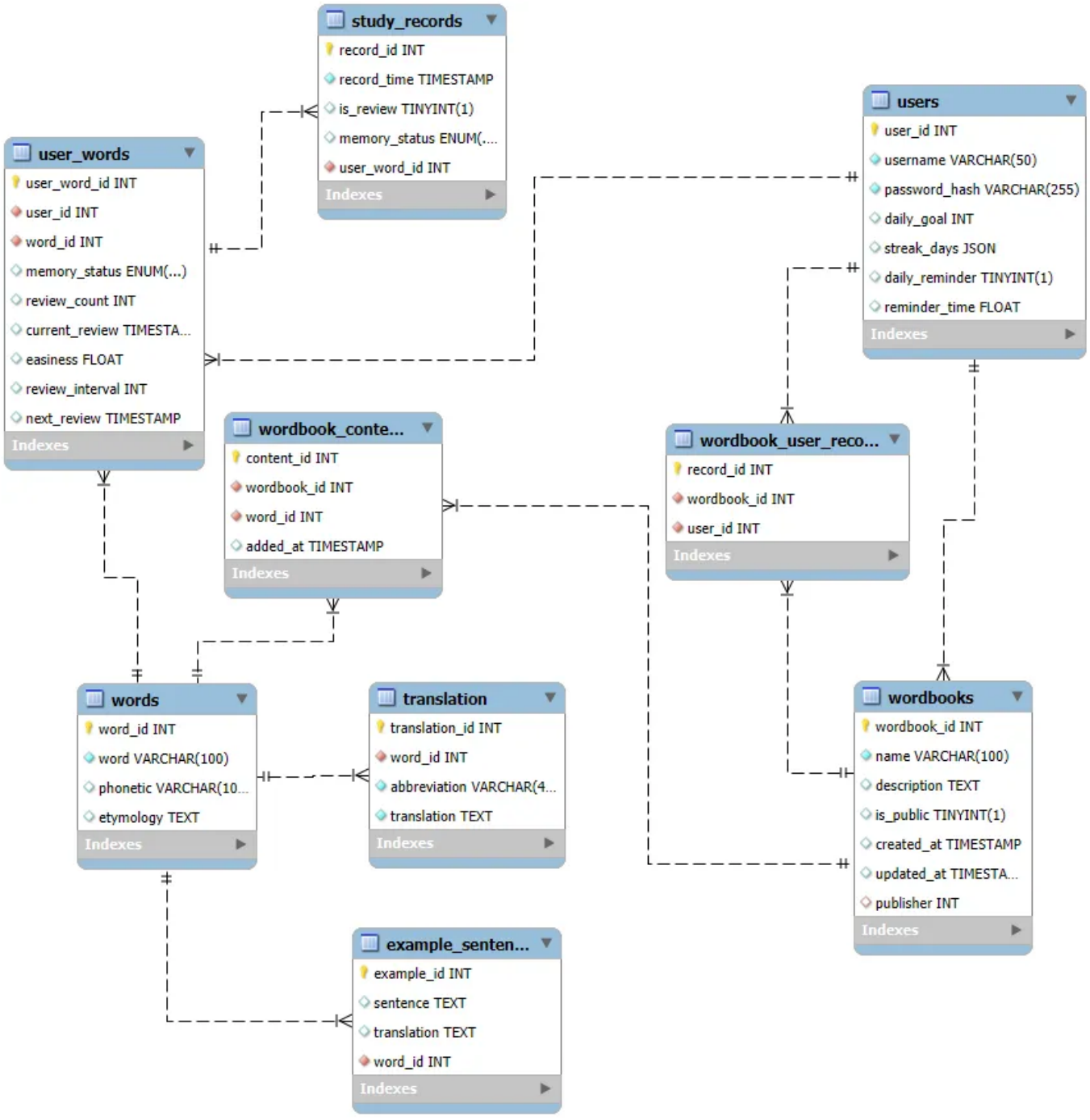
Wordbook



Word Statistics

Backend Design

For the Backend part, we try to use FastAPI to build a system based on the router. This implies that our backend is not going to adopt a heavyweight construction approach. Instead, it will be more lightweight and ensure that the addition of functions is smooth. The modularity of Python is well-suited to our requirements.



Web crawler

In the design of this project, in order to obtain richer and more professional information for each word, we introduced a customized Web Crawler module (Web Crawler). This module is mainly used to capture data from two authoritative websites, the Cambridge Dictionary and the Online Etymology Dictionary. Vocabulary is the core resource of the entire application. However, public word banks available on the market often lack multi-dimensional information such as

pronunciation, etymology, and contextual example sentences, and thus fail to meet the demands of high-quality recitation systems.

Therefore, we decided to build our own lexical semantic database through an automated approach. We download the COCA 20000 word list, which lists the most popular words in the COCA corpus. And then we can extract word information from the Cambridge Dictionary and get etymology from the Online Etymology Dictionary.

Implementation

Frontend Implementation

Our software is built based on React Native. React Native is a cross-platform mobile development framework launched by Facebook, which supports writing code using TypeScript or JavaScript and building user interfaces through JSX. Its greatest advantage is "one set of code, running on multiple platforms", which can be deployed simultaneously on both iOS and Android platforms, greatly enhancing development efficiency and reducing maintenance costs. We chose React Native as the basic framework of this project mainly because it has a rapid development feedback mechanism, an active open-source community, rich third-party component resources, and good integration with TypeScript, which is conducive to improving code quality and maintainability. For apps like ours that focus on vocabulary memory and interactive experience, React Native can well meet the product's requirements in aspects such as page construction, state management, and user operation responses. It is an efficient and reliable technical choice. The syntactic features of JSX can reduce our development costs. Meanwhile, RN also provides a more aesthetically pleasing component library for use.

RN can also reduce our application size and get higher speed. In Hybrid development, the application is based on a webview or other browsers. So, for each application, a complete set of browser systems actually needs to run behind it. However, in many cases, most of the functions of the browser are not what is required for development

Our Implementation involved several modules:

1. RequestWrapper

JavaScript

```
export const requestWrapper = async (
  requestURL,
  body,
  options
): Promise<number | Response> => {
  const { method, signal } = options ?? {};
  const auth = await AsyncStorage.getItem('access_token');
  const requestOptions: any = {
    method: method ?? 'POST',
    headers: {
      'Content-Type': 'application/json',
      Authorization: 'Bearer ' + (auth || ''),
    },
  };
  try {
    const response = await fetch(
      BASE_URL + requestURL,
      {
        ...requestOptions,
        body: body ? JSON.stringify(body) : null,
        signal,
      },
    );
    if (response.ok) {
      return response;
    } else {
      if (expireCounter < MAX_RETRY_TIME) {
        // refresh token
        expireCounter++;
        return requestWrapper(requestURL, body, options);
      } else {
        expireCounter = 0;
        if (response.status === 401) {
          // token expired
          await AsyncStorage.removeItem('access_token');
          userStore.logout();
          return 401;
        }
        return response.status;
      }
    }
  } catch (error: any) {
    console.log('fetch error', error);
    return -1;
  }
};
```

This Request Wrapper has the logic of access_token, it will get the token from storage to ensure the state of users. Check the token in each request, and If the backend detects that the token has expired and returns 401, we will remove the token stored locally.

2. SM2 Algorithm

On the front end, we develop another utility function that can calculate the time for the next review and the result of each review. The specific implementation of this function is as follows:

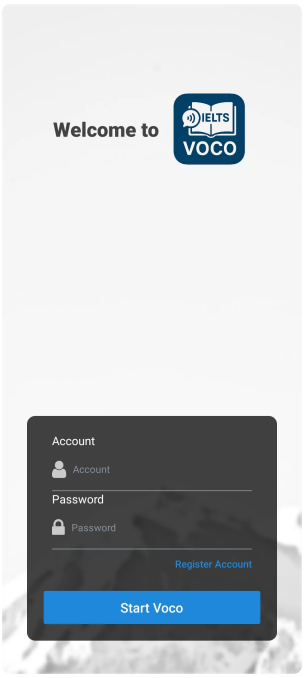
```typescript
export function computeSM2(params: ReviewParams): ReviewResult {
  let {qualityText, easiness, interval, review_count, review_datetime} = params;
  const quality = {
    remembered: 5,
    forgot: 0,
    vague: 3,
  }[qualityText];
  if (quality < 3) {
    interval = 1;
    review_count = 0;
  } else {
    if (review_count === 0) {
      interval = 1;
    } else if (review_count === 1) {
      interval = 6;
    } else {
      interval = Math.ceil(interval * easiness);
    }
    review_count += 1;
  }
  easiness += 0.1 - (5 - quality) * (0.08 + (5 - quality) * 0.02);
  easiness = Math.max(easiness, 1.3);
  const now = review_datetime ? new Date(review_datetime) : new Date();
  now.setMilliseconds(0);
  const nextReviewDate = new Date(
    now.getTime() + interval * 24 * 60 * 60 * 1000,
  );
  if (nextReviewDate < new Date()) {
// If the time for the next review is less than the current time, calculate the next time from the current time    return computeSM2({
      qualityText,
      easiness,
      interval,
      review_count,
      review_datetime: new Date(),
    });
  }
  return {
    easiness: parseFloat(easiness.toFixed(2)),interval,review_count,review_datetime: nextReviewDate
      .toISOString().replace('T', ' ').slice(0, 19),
  };
}
```

The front end classifies words based on three user choices: remembered, vague, and forgot. Different choices correspond to different QualityText. easiness is calculated each time, and the

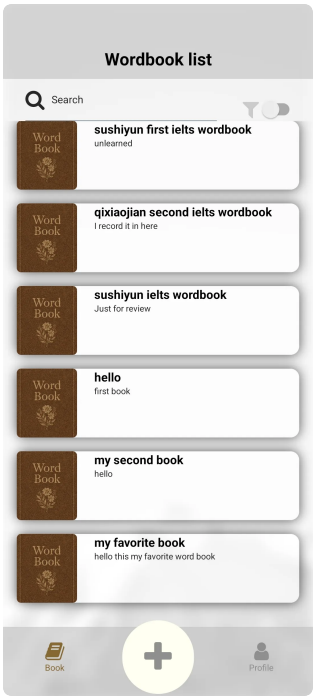
previous easines are also referred to in the next calculation.

3. RN basic implementation

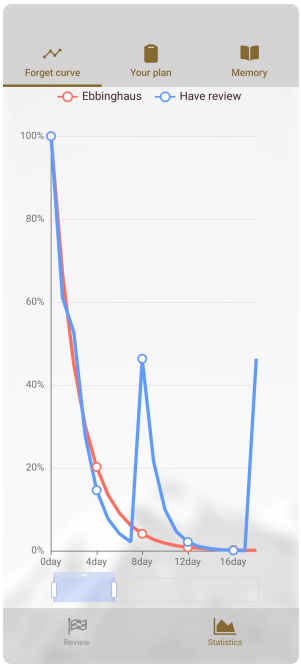
The overall development of the front end is based on routing and components. On the root component of the project, we defined the required routing component, which can well achieve page redirection and connection. On this basis, we developed the functions of each component.



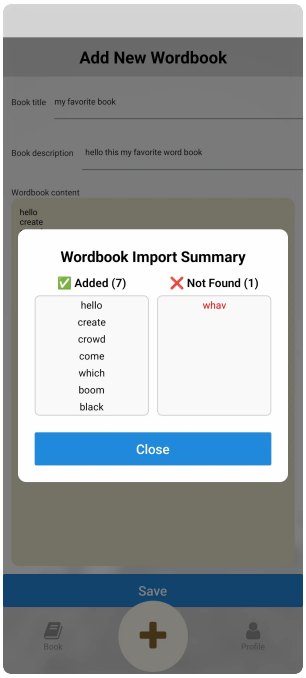
User Login



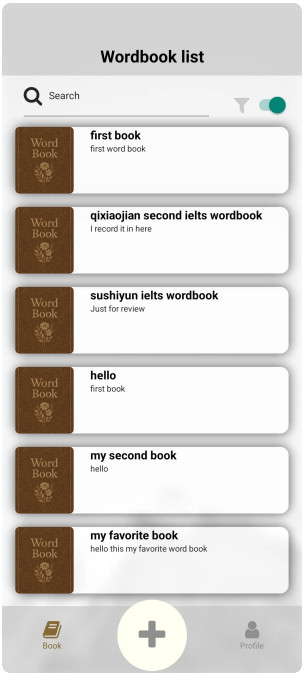
Word book

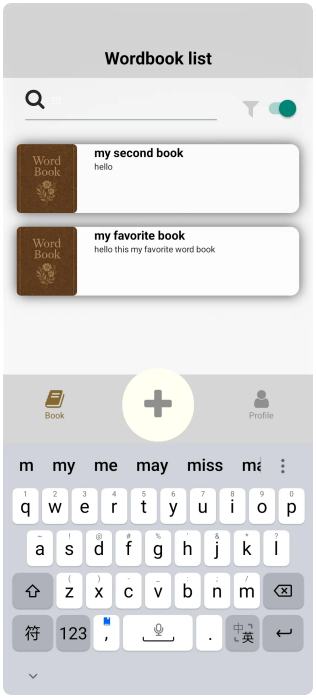


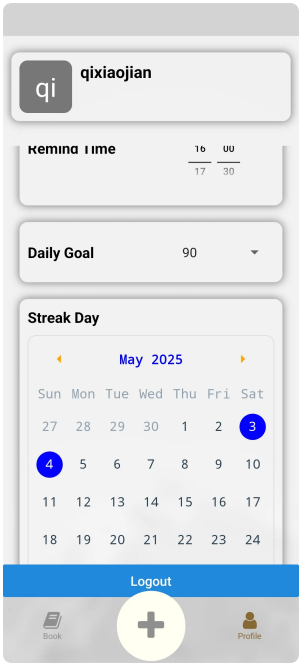
Curve of your memory

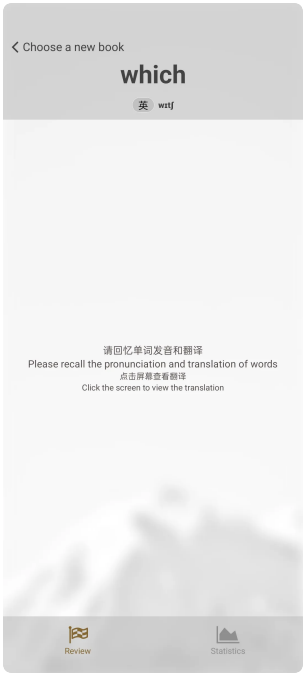


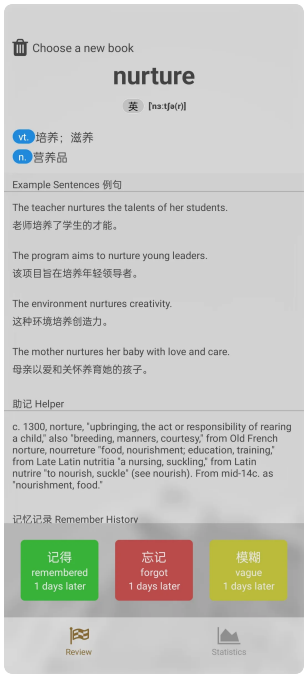
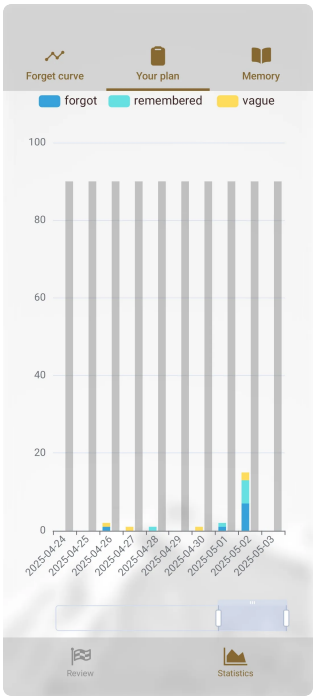
check word in the book









Filter all book create by yourself	Word book search	User basic profile	Word
			
Word content	Statistics of your study	Book content	Book content search

Backend Implementation

1. Routers

In FastAPI, Python uses router to manage the entire system. Our system is service–based, which means that users actually access a certain service. When memorizing words, they will be directed to the specified service through the URL. The overall router is divided into Word router and Auth router. Be responsible for the background logic of words and the login and registration of users respectively

Auth Router	
POST /login	Authenticates user and returns JWT token
POST /register	Creates a new user account and returns JWT token
POST /set_user_setting	Updates user's daily goal and reminder time
POST /set_streak_day	Sets the user's current streak day

Word Router	
POST /wordbook_list	Retrieves all wordbooks (optionally filtered by name)
POST /wordbook_list_by_user	Gets wordbooks belonging to the current user
POST /get_words_from_wordbook	Fetches both unlearned and reviewed words from a wordbook
POST /get_words_detail_from_wordbook	Returns detailed word list (with optional search filter)
POST /get_word_info	Retrieves metadata for a specific word
POST /set_word_status	Updates memory status (e.g., learning/review) of a word
POST /add_wordbook	Creates a new wordbook and adds words to it
POST /get_memory_status	Returns 30-day memory statistics for the user

2. Database and Data

We designed a crawler to obtain the detailed information of words in Cambridge Dictionary and Online Etymology Dictionary. The overall process is that we will first use BeatifulSoup to locate the div or span where the word information of Cambridge Dictionary and Online Etymology Dictionary is located, then parse the content, and finally put the parsed content into a JSON file. Then another function is used to extract the content of the JSON and write it to the database.

```
FUNCTION get_trans(word):
    TRY:
        // 1. Setup request
        url = "https://dictionary.cambridge.org/.../" + word
        headers = {"User-Agent": random_user_agent()}

        // 2. Extract translations
        translations = []
        FOR EACH div IN soup.find_all("translation_div_classes"):

        // 3. Get pronunciation
        phonetic = ""
        IF found_phonetic_tag:
            phonetic = phonetic_text

        // 4. Get example sentences
        examples = []
        FOR EACH example_div IN first_5_examples:
            example = {
                "sentence": english_sentence,
                "translation": chinese_translation
            }
        // 5. Return structured data
        RETURN res
    CATCH error:
        PRINT "Error processing", word, ":", error
        RETURN None
END FUNCTION
```

Besides, we use technology of connection pool in database connection. it can reduce the time to connect with the database

```python
def init_connection_pool():
    global connection_pool
    connection_pool = pooling.MySQLConnectionPool(
        pool_name="voco_pool", pool_size=5, **dbconfig
    )
    logger.debug(f"连接池初始化完成，大小: {connection_pool.pool_size}")
def close_connection_pool():
    global connection_pool
    if connection_pool:
        connection_pool = None
        logger.debug("连接池已关闭。")
@contextmanager
def get_connection():
    conn = None
    try:
        conn = connection_pool.get_connection()
        yield conn
    except mysql.connector.Error as err:
        logger.debug(f"获取连接失败: {err}")
        raise
    finally:
        if conn:
            conn.close()
```

In the implementation of the database, we first used the reverse engineering of MySQL to draw the EER diagram of the database and then established the tables. The advantage of this approach is that it can achieve the table structure of the database from the very beginning, especially the primary key and foreign key relationships between different tables.