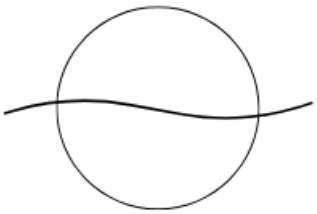
Technical Report
for
Tiku

Prepared by:

Jun Wu Wang
164911



January 2025

Contents

Chapter 1

Outline

1.1 Introduction

Tiku is a question bank (QBank) hosting service.

A QBank is a collection of questions, usually used to practice for exams or more generally practice or test one's knowledge on a topic.

Tiku allows users to create and share QBanks, with the goal of fostering a community of people following a learning model based on active recalling and spaced repetition.

1.2 Features/Product backlog

- As an anonymous user, I want to sign up, so that I can have a personal account
- As an anonymous user, I want to sign in, so that I can access to user-specific experience and functionalities
- As a registered user, I want to edit my account, so that I can update my personal information
- As a registered user, I want to sign out, so that I can protect my personal information and end my session
- As a registered user, I want to delete my account, so that I can remove all my personal data and end my relationship with the service
- As an user, I want to search QBanks, so that I can see if someone else has already created what I need
- As an user, I want to sort QBanks, so that I can easily find QBanks I need
- As an user, I want to filter QBanks by subject, so that I can filter out QBanks I am not interested in
- As an user, I want to search questions in QBanks, so that I can easily find the questions I need
- As an user, I want to sort questions in QBanks, so that I can easily find the questions I need
- As an user, I want to search users, so that I can find users I am interested in
- As an user, I want to sort users, so that I can easily find the users I am interested in
- As an user, I want to see user profiles, so that I can check them out and see all their public QBanks
- As an user, I want to see QBank ratings, so that I can make informed decisions
- As an user, I want to see the currently trending and most popular QBanks, so that I can find the most interesting QBanks

- As an user, I want to use public QBanks statelessly, so that I can try them out, even if I am not registered
- As a registered user, I want to create my own QBanks, so that I can have custom QBanks based on my needs and share my knowledge with others
- As a registered user, I want to edit my QBanks, so that I can update and improve my QBanks
- As a registered user, I want to delete my QBanks, so that I can remove QBanks that are no longer relevant
- As a registered user, I want to add questions to my QBanks, so that I can fill and update my QBanks
- As a registered user, I want to edit questions in my QBanks, so that I can update and improve my questions
- As a registered user, I want to delete questions from my QBanks, so that I can remove outdated or incorrect questions
- As a registered user, I want to favorite others public QBanks, so that I can have easy access to them
- As a registered user, I want to search in my own QBank list, so that even if I have created or favorited a lot of QBanks, I can still navigate easily in the list
- As a registered user, I want to sort my own QBank list, so that even if I have created or favorited a lot of QBanks, I can still navigate easily in the list
- As a registered user, I want to follow other users, so that I can stay updated about them
- As a registered user, I want to rate others public QBanks, so that I can provide feedback and help others make informed decisions
- As an registered user, I want to use QBanks in my own QBank list statefully with dynamic scheduling of the questions, so that I can take advantage of optimized spaced repetition
- As a registered user, I want to see recommended QBanks, so that I can easily find QBanks tailored for me

1.3 Random mode and Spaced Repetition mode

Each question gets rated by the user based on their self-assessment of how good they think did (Again, Hard, Good, Easy).

Random mode	**Spaced Repetition mode**
Stateless: rating the question has no effect	Stateful: rating the question will schedule it to a closer or farther date depending on the grade
Any question can come up	Only the questions scheduled for the day can come up

Spaced Repetition mode entails the implementation of a dynamic scheduling system.

1.4 Recommendation system

The recommendation system uses a user-based collaborative filtering technique.

1.5 Searching, filtering and sorting

Search for:

- QBanks
- Questions
- Users

Filter by:

- All (Qbanks, Questions, Users)
- Subject (QBanks)
- Due today (Qbanks)

Sort by:

- Trending (QBanks, Users)
- Popularity (QBanks, Users)
- Name (QBanks)
- Newest (Questions)
- Oldest (Questions)

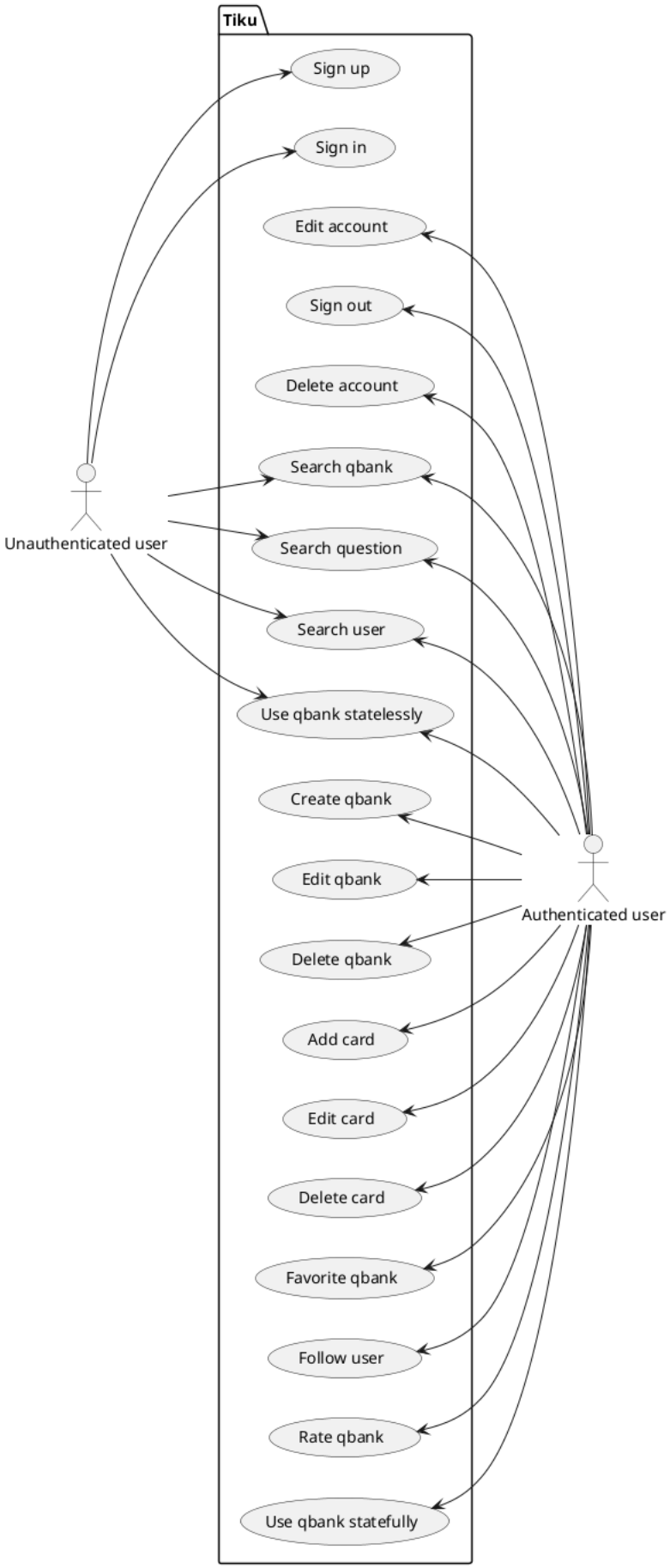
Chapter 2

General description

2.1 Introduction

Tiku is a web app built with Django and Bootstrap, designed to host question banks for efficient learning using spaced repetition and active recall techniques. Users can create, practice, and explore question banks shared by others.

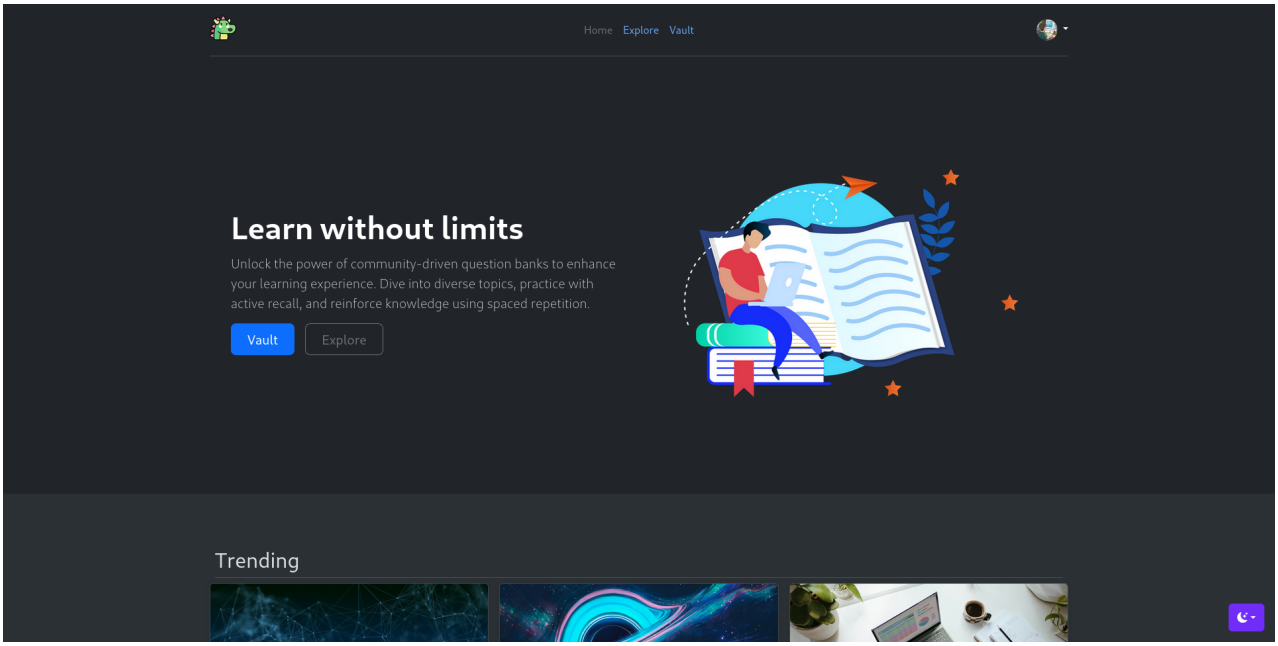
2.2 Product functions



The key difference is that authenticated users can actively contribute to the platform by creating, managing, and rating Qbanks, as well as building connections with other users. Additionally, they gain access to stateful usage of Qbanks, allowing them to track their progress.

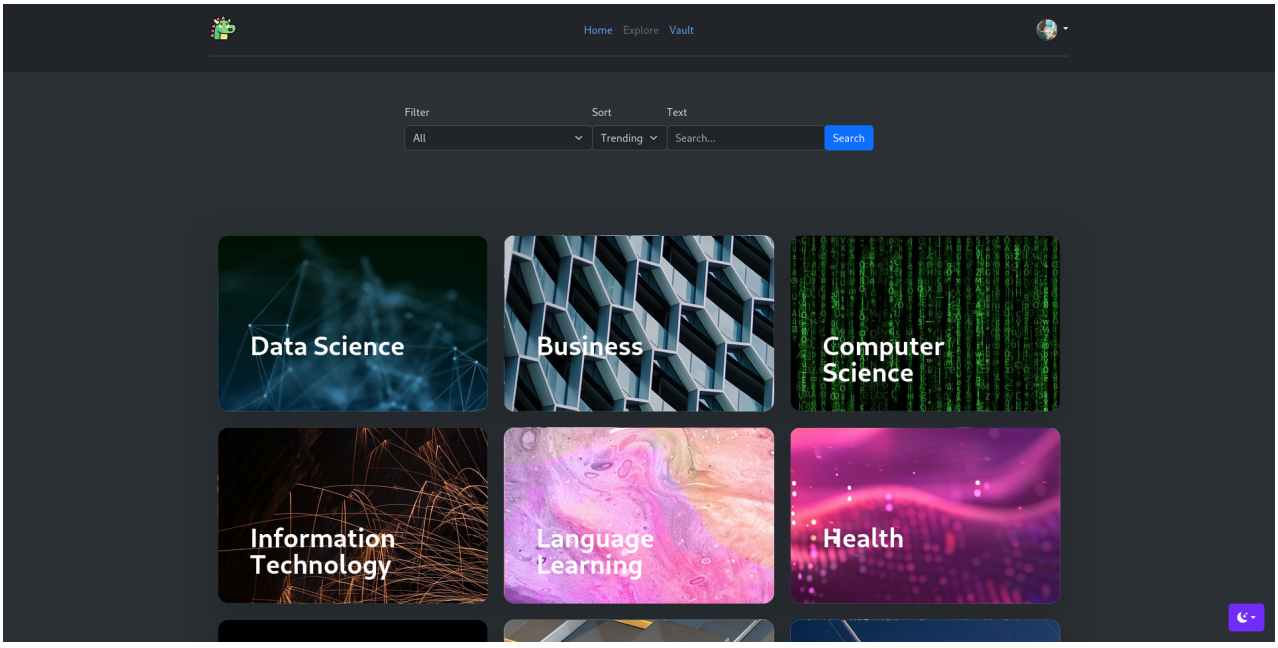
2.3 User interface

2.3.1 Home page



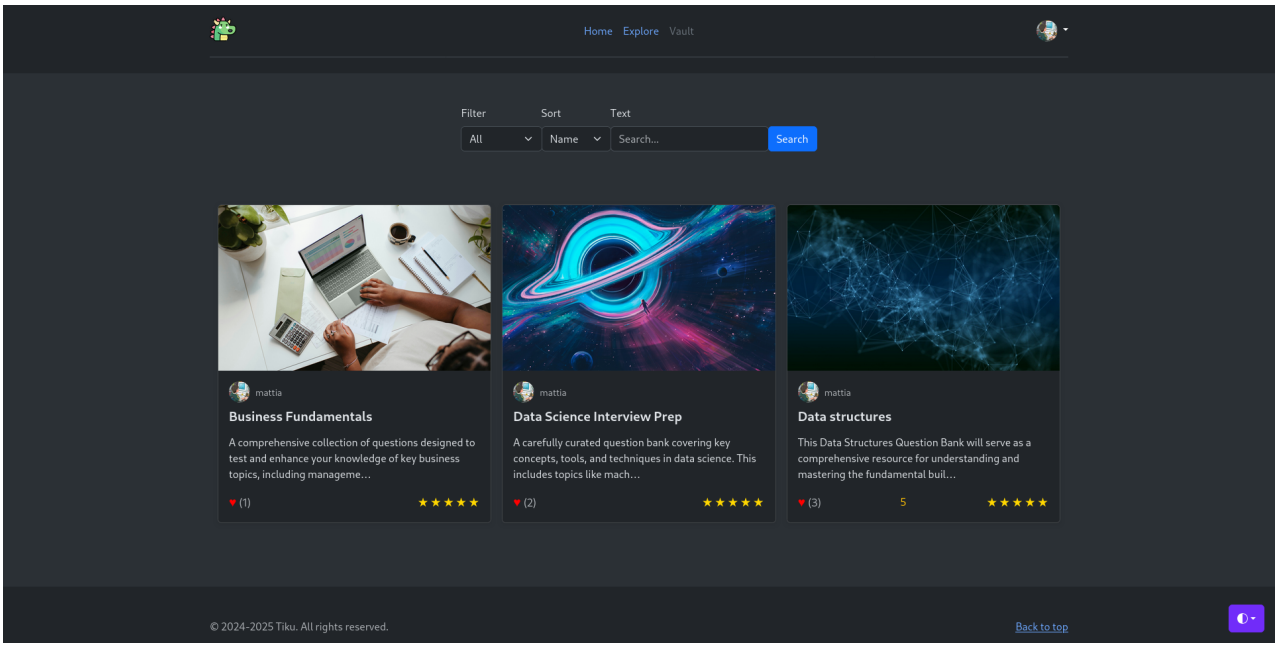
The homepage is the landing page of the websites. It highlights recommended, trending, and popular qbanks.

2.3.2 Explore page



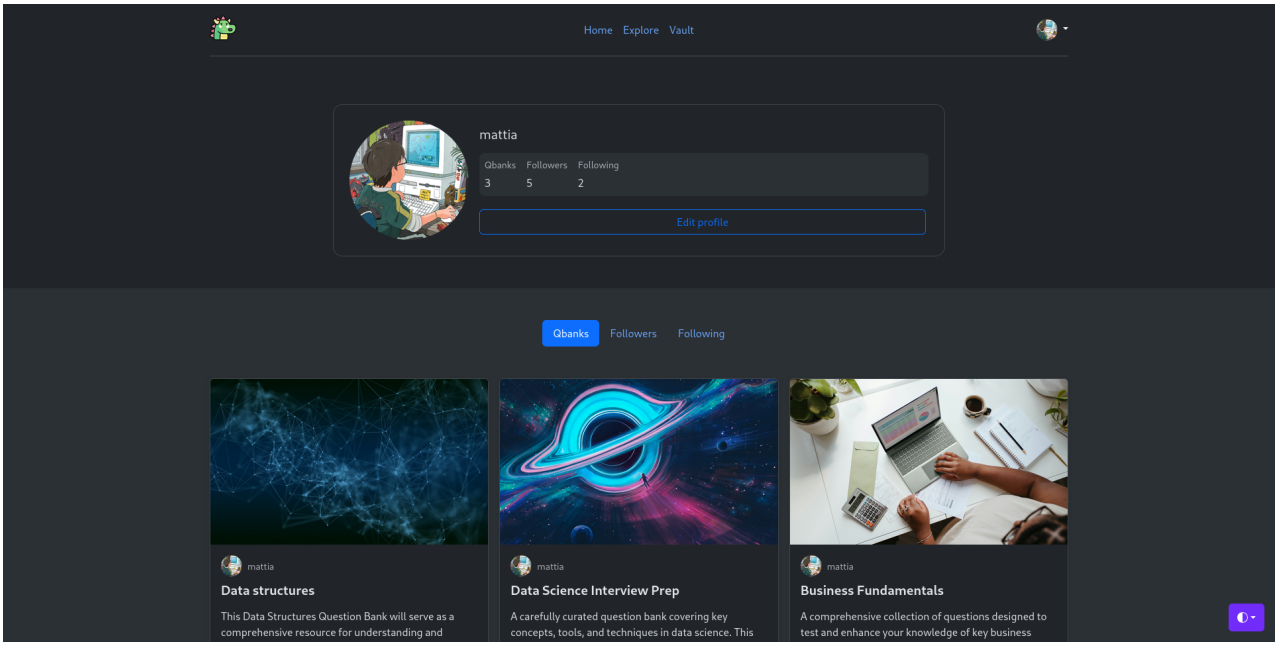
The Explore page allows users to search for qbanks by topic, featuring a search bar, to help users easily discover relevant content.

2.3.3 Vault page



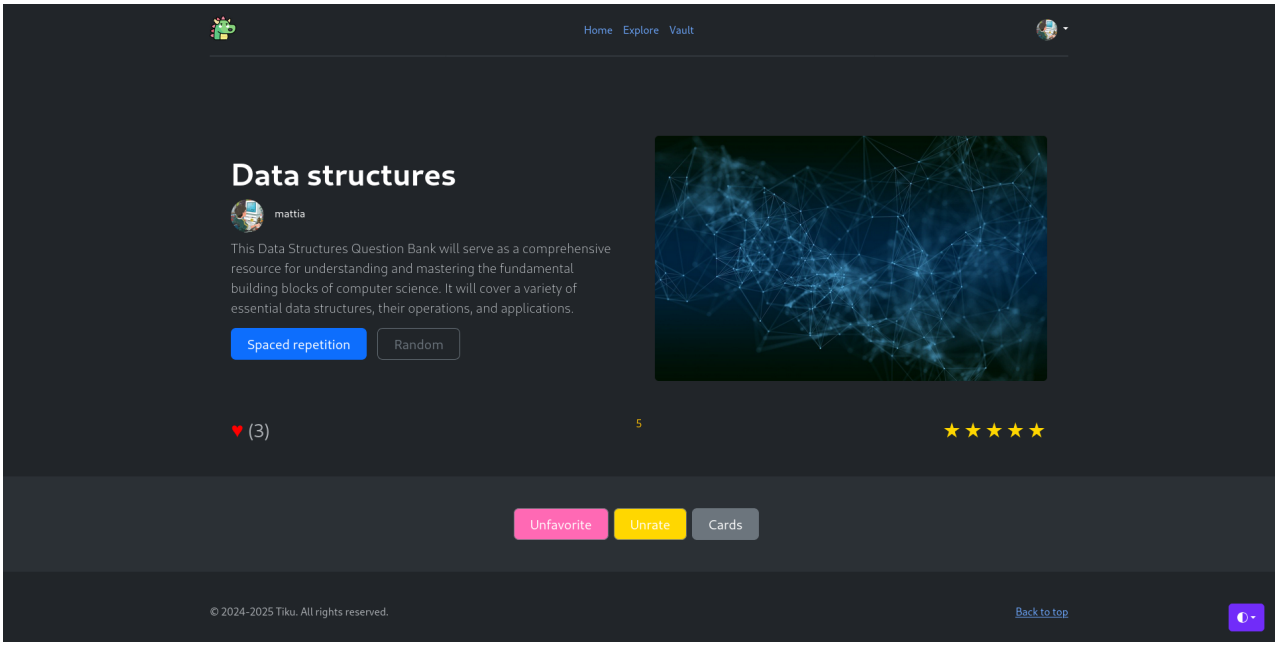
The Vault page displays all the qbanks that the user has favorited, providing quick access to their saved content in one organized location.

2.3.4 Profile page



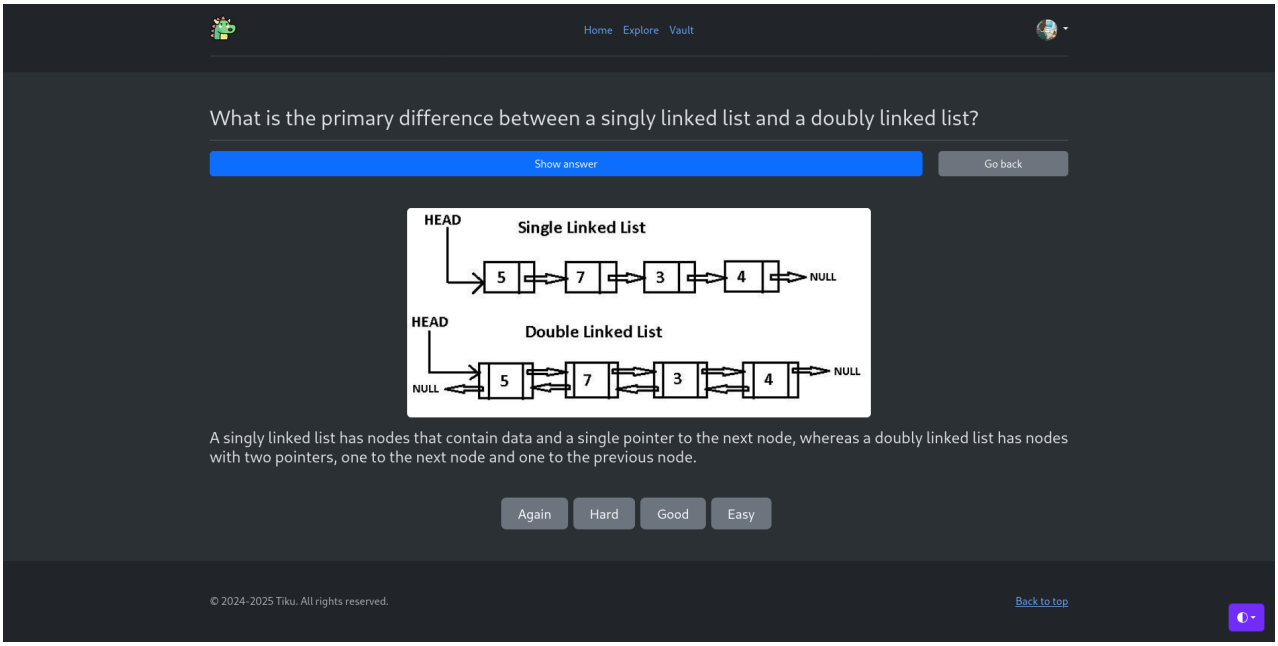
The Profile page showcases the user's personal information, a list of their created Qbanks, and details about their followers and the users they are following.

2.3.5 Qbank page



The Qbank page displays detailed information about a specific question bank.

2.3.6 Card page



The Card page presents a question and allows the user to reveal the answer. After viewing the answer, users can rate their response based on how well they recalled it, helping to adjust the card's review schedule for future sessions.

2.4 Code organization

The project is divided into 3 applications:

- qbanks
- users
- cards

The project is divided into 3 applications to ensure modularity to the project and apply the principle of separation of concerns, enhancing its readability, maintainability and scalability.

Each application has its own models, templates and views.

2.4.1 Qbanks

The qbanks application revolves around the Qbank model. It manages all logic concerned with qbanks, such as keeping track of users favoriting qbanks, keeping track of users rating qbanks, managing the recommendation system for suggesting relevant qbanks to users. It also serves as the repository for the primary templates of the website.

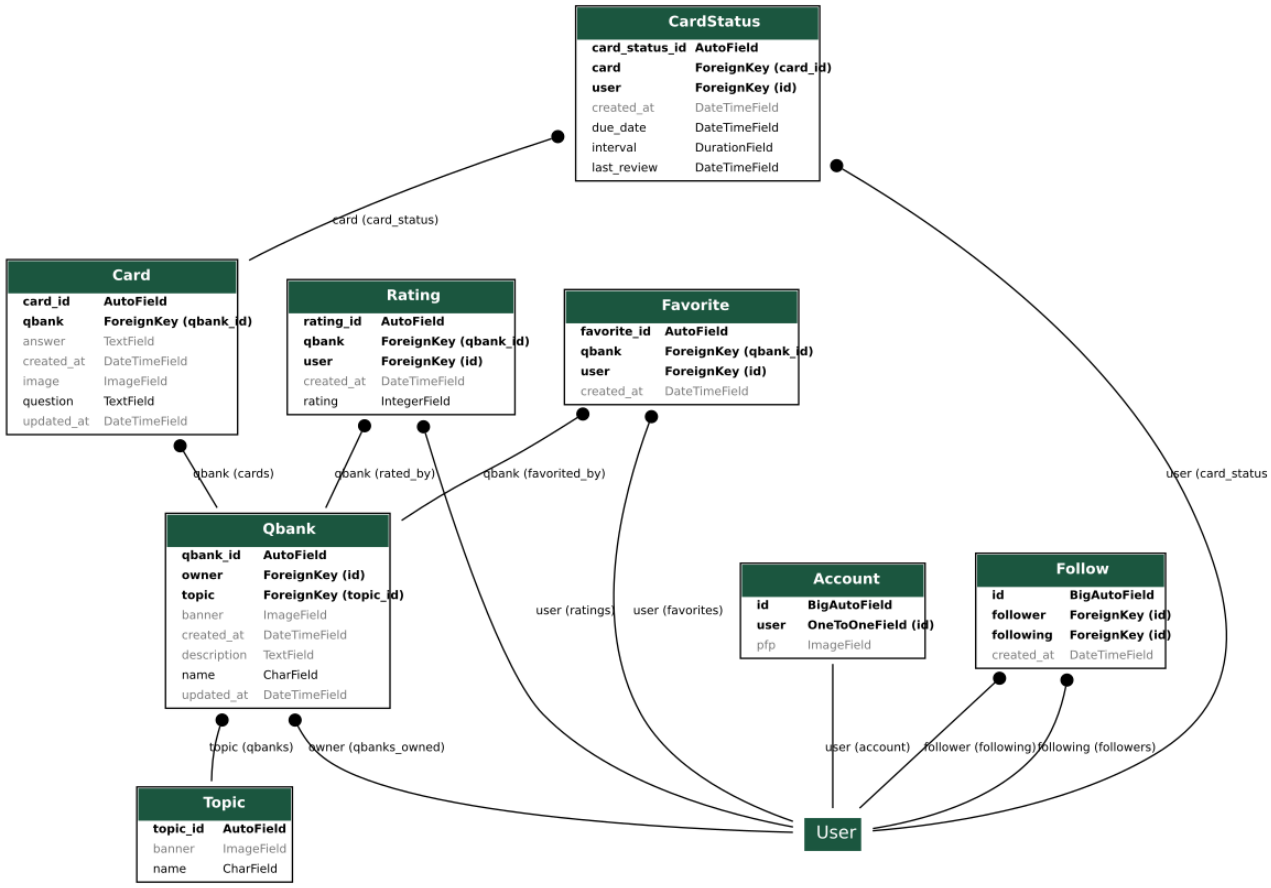
2.4.2 Cards

The cards application centers around the Card model. It implements the scheduling logic of cards for optimized spaced repetition and keeps track of user's progress with each card/question.

2.4.3 Users

The users application centers around the User/Account model. It manages core user functionality, such as user creation, profile editing and authentication logic. It also implements logic for users to follow each other.

2.5 Database



2.6 Spaced repetition algorithm

The spaced repetition algorithm is a very simplified version of what softwares like Anki do: it adjusts the scheduling interval for reviewing a question based on the user's self-assessment of how well they answered it. The algorithm dynamically updates the review interval to optimize retention.

2.6.1 Algorithm

1. Initial review: When a question is first introduced, the review interval is set to 0 days (reviewed immediately)
2. User rating categories: After answering the question, the user rates their confidence in their answer using one of four options:
   - Again: Indicates the user found the question very difficult or forgot the answer
   - Hard: Indicates the user struggled but managed to recall part of the answer
   - Good: Indicates the user answered correctly with some effort
   - Easy: Indicates the user answered effortlessly and confidently
3. Adjusting the interval: Based on the rating, the review interval for the question is updated:
   - Again: The interval resets to 0 days (review the same day to reinforce)

- Hard: The interval is halved, reducing the time until the next review
- Good: The interval remains the same, keeping the current schedule
- Easy: The interval is doubled, increasing the time until the next review

4. Repeat process: Each time the question is reviewed, the user rates it again, and the interval is updated dynamically

2.6.2 Implementation

```
def schedule(card_status, difficulty):
    if difficulty == 'easy':
        # Checks if interval is zero
        if card_status.interval.total_seconds() < 86400:
            # Sets the interval to 2 days
            card_status.interval = timedelta(days=2)
        else:
            # Doubles the interval
            card_status.interval *= 2
    elif difficulty == 'good':
        # Checks if interval is zero
        if card_status.interval.total_seconds() < 86400:
            # Sets the interval to 1 days
            card_status.interval = timedelta(days=1)
        else:
            # Keeps the same interval
            pass
    elif difficulty == 'hard':
        # Halves the interval
        card_status.interval /= 2
    elif difficulty == 'again':
        # Resets to 0
        card_status.interval = timedelta(days=0)
    else:
        raise Http404()

    card_status.last_review = timezone.now()
    card_status.due_date = timezone.now()+card_status.interval
    card_status.save()
```

The inclusion of *due_ date* as explicit field is a deliberate design choice aimed at optimizing retrieval performance.

2.7 Recommendation system

It is a simple recommendation system using user-based collaborative filtering. User-based collaborative filtering methods predict the items that a user might like on the basis of ratings given to that item by other users who have similar taste with that of the target user.

The choice to use a user-based collaborative filtering was easy because user ratings data was already available, and on the other hand qbanks are not characterized enough to differentiate them well using a content-based filtering method.

2.7.1 Algorithm

High level:

1. Find similar users based on ratings on common qbanks
2. Identify the qbanks rated high by similar users but have not been exposed to the target user
3. Calculate the weighted average score for each of them
4. Rank qbanks based on the score and pick top n items to recommend

Low level:

1. Create user-qbank matrix
2. Normalize ratings in user-bank matrix
3. Calculate similarity using Pearson correlation
4. Get top n similar users
5. Narrow down qbank pool
6. Recommend qbanks

2.7.2 Implementation

Create user-qbank matrix

```
def get_user_qbank_matrix():
    # Query data from the model
    ratings = Rating.objects.all()

    if not ratings:
        return pd.DataFrame()

    # Prepare data for DataFrame
    data = [
        {
            'username': rating.user.username,
            'qbank_id': rating.qbank.qbank_id,
            'rating': rating.rating
        }
        for rating in ratings
    ]

    # Create DataFrame
    df = pd.DataFrame(data)

    # Pivot the DataFrame to create a user-qbank matrix
    user_qbank_matrix = df.pivot_table(index='username', columns='qbank_id', values='rating')

    return user_qbank_matrix
```

User-qbank matrix is the data structure that keeps track of the ratings:

- Rows: users
- Columns: qbanks

Normalize ratings in user-bank matrix

```
def normalize(user_qbank_matrix: pd.DataFrame):
    # Calculates the mean of each row and subtracts it from each element of the corresponding row
    # This normalizes the ratings around 0
    return user_qbank_matrix.subtract(user_qbank_matrix.mean(axis=1), axis='rows')
```

This is done because some users tend to give higher ratings and other lower ratings. Qbanks with ratings lower than their users' average is negative, and ratings greater than their users' average is positive [-1, 1]

Calculate similarity using Cosine similarity

```
def get_user_similarity(user_qbank_matrix_norm: pd.DataFrame):
    # Get user similarity matrix (numpy array)
    user_similarity = cosine_similarity(user_qbank_matrix_norm.fillna(0))
    # Return user similarity matrix (dataframe)
    return pd.DataFrame(user_similarity, index=user_qbank_matrix_norm.index, columns=user_qbank_matrix_norm.
```

Cosine similarity measures the cosine of the angle between two vectors in a multi-dimensional space. This angle represents the difference in direction between the vectors:

- Angle = 0° (cosine = 1): Vectors point in the same direction → Maximum similarity
- Angle = 90° (cosine = 0): Vectors are orthogonal → No similarity
- Angle = 180° (cosine = -1): Vectors point in opposite directions → Maximum dissimilarity

Get top n similar users

```
def get_similar_users(user_similarity: pd.DataFrame, user: User, head: int, threshold: float):
    if user.username not in user_similarity:
        return pd.DataFrame()

    # Filters user that have similarity over the threashold,
    # sorts them based on their similarity score for user in descending order
    # and takes the top n most similar users
    return user_similarity[user_similarity[user.username]>threshold][user.username].sort_values(ascending=Fa
```

Narrow down qbank pool

```
def get_similar_user_qbanks(user_qbank_matrix_norm: pd.DataFrame, similar_users: pd.DataFrame, user: User):
    # Qbanks that the target user has rated
    target_user_rated_qbanks = user_qbank_matrix_norm[user_qbank_matrix_norm.index == user.username].dropna(

    # Qbanks that similar users have rated. Remove qbanks that none of the similar users have rated
    similar_user_rated_qbanks = user_qbank_matrix_norm[user_qbank_matrix_norm.index.isin(similar_users.index

    # Remove the rated qbanks by target user from the qbank list
    return similar_user_rated_qbanks.drop(target_user_rated_qbanks.columns,axis=1, errors='ignore')
```

This means removing qbanks that have been rated by the target user and keep only the qbanks that the top n similar users have rated.

Recommend qbanks

```
def get_recommended_qbanks(similar_users: pd.DataFrame, similar_user_qbanks: pd.DataFrame, user: User, head:
    # A dictionary to store qbank scores
    qbank_score = {}

    # Loop through qbanks
    for i in similar_user_qbanks.columns:
        # Get the ratings for qbank i
        qbank_rating = similar_user_qbanks[i]
        # Create a variable to store the score
        total = 0
        # Create a variable to store the number of scores
        count = 0

        # Loop through similar users
        for u in similar_users.index:
            # If the qbank has rating
            if not pd.isna(qbank_rating[u]):
                # Score is the sum of user similarity score multiply by the qbank rating
                score = similar_users[u] * qbank_rating[u]
                # Add the score to the total score for the qbank so far
                total += score
                # Add 1 to the count
                count += 1
        # Get the average score for the qbank
        qbank_score[i] = total / count

    # Convert dictionary to pandas dataframe
    qbank_score = pd.DataFrame(qbank_score.items(), columns=['qbank', 'qbank_score'])

    # Sort the qbanks by score
    ranked_qbank_score = qbank_score.sort_values(by='qbank_score', ascending=False)

    # Get list qbank_ids
    qbank_ids = ranked_qbank_score['qbank'].tolist()

    # Filter out qbanks owned by target user
    for qbank_id in qbank_ids[:]:
        qbank = Qbank.objects.get(qbank_id=qbank_id)
        if not qbank or qbank.owner == user:
            qbank_ids.remove(qbank_id)

    # Get top n qbanks
    qbank_ids = qbank_ids[:head]

    # Return queryset
    return Qbank.objects.filter(qbank_id__in=qbank_ids)
```

Calculate the weighted average of user similarity score and qbank rating and pick the top n qbanks.

2.8 Tests

This section provides an overview and explanation of a selection of the automated tests that have been implemented.

2.8.1 Spaced repetition algorithm tests

Spaced repetition algorithm

```
class CardService:
    @staticmethod
    def schedule(card_status, difficulty):
        if difficulty == 'easy':
            # Checks if interval is zero
            if card_status.interval.total_seconds() < 86400:
                # Sets the interval to 2 days
                card_status.interval = timedelta(days=2)
            else:
                # Doubles the interval
                card_status.interval *= 2
        elif difficulty == 'good':
            # Checks if interval is zero
            if card_status.interval.total_seconds() < 86400:
                # Sets the interval to 1 days
                card_status.interval = timedelta(days=1)
            else:
                # Keeps the same interval
                pass
        elif difficulty == 'hard':
            # Halves the interval
            card_status.interval /= 2
        elif difficulty == 'again':
            # Resets to 0
            card_status.interval = timedelta(days=0)
        else:
            raise Http404()

        card_status.last_review = timezone.now()
        card_status.due_date = timezone.now()+card_status.interval
        card_status.save()
```

Easy rating scheduling test (standard case)

```
def test_schedule_easy_rating(self):
    card = Card.objects.create(qbank=self.qbank, question='test_question')
    card_status = CardStatus.objects.create(
        user=self.user,
        card=card,
        interval=timedelta(days=7),
        last_review=timezone.now(),
        due_date=timezone.now()
    )
    difficulty = 'easy'
    CardService.schedule(card_status, difficulty)
```

```
    card_status.refresh_from_db()

    self.assertEqual(card_status.interval, timedelta(days=14))
```

It checks if the interval doubles.

First easy rating scheduling test (edge case)

```
def test_schedule_first_easy_rating(self):
    card = Card.objects.create(qbank=self.qbank, question='test_question')
    card_status = CardStatus.objects.create(
        user=self.user,
        card=card,
        interval=timedelta(days=0),
        last_review=timezone.now(),
        due_date=timezone.now()
    )
    difficulty = 'easy'
    CardService.schedule(card_status, difficulty)
    card_status.refresh_from_db()

    self.assertEqual(card_status.interval, timedelta(days=2))
```

It checks if the interval is set to 2 days.

Invalid rating scheduling test

```
def test_schedule_invalid_rating(self):
    card = Card.objects.create(qbank=self.qbank, question='test_question')
    card_status = CardStatus.objects.create(
        user=self.user,
        card=card,
        interval=timedelta(days=7),
        last_review=timezone.now(),
        due_date=timezone.now()
    )
    difficulty = 'ok'

    with self.assertRaises(Http404):
        CardService.schedule(card_status, difficulty)
```

It checks if invalid ratings raise Http404.

2.8.2 Card view tests

Card view

```
def card(request):
    context = {}
    context['mode'] = request.GET.get('mode') or request.POST.get('mode')
    card_id = request.GET.get('card_id') or request.POST.get('card_id')
    qbank_id = request.GET.get('qbank_id') or request.POST.get('qbank_id')
    # If it's a GET request (fetches card to show on page)
```

```
    if request.method == 'GET':
        context['next'] = request.GET.get('next', '/')

        # If request comes from cards_list (card_id, mode=selection)
        if card_id and context['mode'] and context['mode'] == 'selection':
            # Gets the specified card
            context['card'] = get_object_or_404(Card, card_id=card_id)

        # If request comes from random button (qbank_id, mode=random)
        elif qbank_id and context['mode'] and context['mode'] == 'random':
            context['qbank'] = get_object_or_404(Qbank, qbank_id=qbank_id)

            # Gets random card from qbank
            context['card'] = Card.objects.filter(qbank=context['qbank']).order_by('?').first()

        # If request comes from spaced repetition button (qbank_id, mode=spaced_repetition)
        # (must be logged in and must have favorited the qbank, if not call favorite for both cases)
        elif qbank_id and context['mode'] and context['mode'] == 'spaced_repetition':
            context['qbank'] = get_object_or_404(Qbank, qbank_id=qbank_id)
            if not request.user.is_authenticated:
                login_url = reverse('users:login')
                query_string = f"?next={context['next']}"
                return redirect(login_url+query_string)

            # If user has not favorited the qbank
            if not Favorite.objects.filter(qbank=context['qbank'], user=request.user).exists():
                # User favorites qbank
                Favorite.objects.create(user=request.user, qbank=context['qbank'])

                # Gets list of cards of the qbank
                cards = Card.objects.filter(qbank=context['qbank'])

                # For each card it creates a card status for the logged in user
                for card in cards:
                    card_status_data = {
                        'user': request.user,
                        'card': card,
                        'interval': timedelta(days=0),
                        'last_review': timezone.now(),
                        'due_date': timezone.now()
                    }
                    CardStatus.objects.create(**card_status_data)

            # Gets first of the due cards (oldest due date)
            card_status = CardStatus.objects.filter(
                user=request.user,
                card__qbank=context['qbank'],
                due_date__lte=timezone.now()
            ).first()
            if card_status:
                context['card'] = card_status.card
        else:
            raise Http404()

        return render(request, 'cards/card.html', context)
```

```
    # If it's a POST request (updates CardStatus and redirects to next page)
    elif request.method == 'POST':
        next_url = request.POST.get('next', '/')

        # If request comes from cards_list, don't update CardStatus
        if context['mode'] and context['mode'] == 'selection':
            # Redirects to cards list
            return redirect(next_url)

        # If request comes from random button, don't update CardStatus
        elif qbank_id and context['mode'] and context['mode'] == 'random':
            # Redirects to next random card
            card_url = reverse('cards:card')
            query_string = f"?mode={context['mode']}&qbank_id={qbank_id}&next={next_url}"
            return redirect(card_url+query_string)

        # If request comes from spaced repetition button
        # (must be logged in and must have favorited the qbank, if not call favorite for both cases)
        elif qbank_id and context['mode'] and context['mode'] == 'spaced_repetition':
            context['qbank'] = get_object_or_404(Qbank, qbank_id=qbank_id)

            if not request.user.is_authenticated:
                # Redirects to login page
                login_url = reverse('users:login')
                query_string = f"?next={next_url}"
                return redirect(login_url+query_string)

            if not Favorite.objects.filter(qbank=context['qbank'], user=request.user).exists():
                Favorite.objects.create(user=request.user, qbank=context['qbank'])

                # Get list of cards of the qbank
                cards = Card.objects.filter(qbank=context['qbank'])

                # For each card create a card status for the logged in user
                for card in cards:
                    card_status_data = {
                        'user': request.user,
                        'card': card,
                        'interval': timedelta(days=0),
                        'last_review': timezone.now(),
                        'due_date': timezone.now()
                    }
                    CardStatus.objects.create(**card_status_data)

            difficulty = request.POST.get('difficulty')
            if not card_id or not difficulty:
                raise Http404()

            # Schedules card based on user rating
            card = get_object_or_404(Card, card_id=card_id)
            card_status = CardStatus.objects.get(user=request.user, card=card)
            CardService.schedule(card_status, difficulty)

            # Redirects to next due card
            card_url = reverse('cards:card')
            query_string = f"?mode={context['mode']}&qbank_id={qbank_id}&next={next_url}"
```

```
            print(card_url+query_string)
            return redirect(card_url+query_string)
        else:
            raise Http404()
    else:
        raise Http404()
```

GET card in spaced repetition mode with authenticated user that has favorited the qbank test

```
def test_get_spaced_repetition_mode_authenticated_user(self):
    Favorite.objects.create(user=self.user, qbank=self.qbank)

    # Create a CardStatus for the user
    card_status = CardStatus.objects.create(
        user=self.user,
        card=self.card,
        interval=timedelta(days=0),
        last_review=timezone.now(),
        due_date=timezone.now()
    )

    response = self.client.get(self.url, {'mode': 'spaced_repetition', 'qbank_id': self.qbank.qbank_id})

    # Check response status
    self.assertEqual(response.status_code, 200)

    # Check that the due card is in the context
    self.assertEqual(response.context['card'], self.card)
```

It checks if it returns correctly with a valid card.

GET card in spaced repetition mode with authenticated user that has favorited the qbank with missing data test

```
def test_get_spaced_repetition_mode_authenticated_user_missing_data(self):
    Favorite.objects.create(user=self.user, qbank=self.qbank)

    # Create a CardStatus for the user
    card_status = CardStatus.objects.create(
        user=self.user,
        card=self.card,
        interval=timedelta(days=0),
        last_review=timezone.now(),
        due_date=timezone.now()
    )

    response = self.client.get(self.url, {'mode': 'spaced_repetition'})

    # Check response status
    self.assertEqual(response.status_code, 404)
```

It checks if missing data raise Http404.

GET card in spaced repetition mode with unauthenticated user test

```
def test_get_spaced_repetition_mode_unauthenticated_user(self):
    self.client.logout()  # Log out the user
    response = self.client.get(self.url, {'mode': 'spaced_repetition', 'qbank_id': self.qbank.qbank_id})

    # Check redirect to login page
    self.assertRedirects(response, f"{reverse('users:login')}?next=/")
```

It checks if it redirects the user to login page.

GET card in spaced repetition mode with authenticated user that has not favorited the qbank test

```
def test_get_spaced_repetition_no_favorite(self):
    extra_card = Card.objects.create(qbank=self.qbank, question='test_question')

    response = self.client.get(self.url, {'mode': 'spaced_repetition', 'qbank_id': self.qbank.qbank_id})

    # Check response status
    self.assertEqual(response.status_code, 200)

    # Check that the qbank was favorited
    self.assertTrue(Favorite.objects.filter(user=self.user, qbank=self.qbank).exists())

    # Check that CardStatus objects were created for each card in the qbank
    card_statuses = CardStatus.objects.filter(user=self.user, card__qbank=self.qbank)
    self.assertEqual(card_statuses.count(), 2)

    # Verify the attributes of a CardStatus object
    card_status = card_statuses.get(card=self.card)
    self.assertEqual(card_status.user, self.user)
    self.assertEqual(card_status.card, self.card)
    self.assertEqual(card_status.interval, timedelta(days=0))
```

It checks if it automatically favorites the qbank and creates a card status for each card in the qbank for the given user.

Appendices

Appendix A

Active recall and Spaced repetition

Active recall is a study method where you force your brain to recall the information you wish to master. This process makes the information more memorable and easier to retrieve when needed.

Spaced repetition is a method of reviewing material at systematic intervals. At the beginning of the learning process, the intervals are spaced closely together. As the material is reviewed, the intervals become systematically longer. This method allows you to review material before it is forgotten, helping you to retain material in your long term memory.

Tiku tries to combine these two through a question-based learning platform which forces active recall, with a dynamic scheduling system of said questions for optimized spaced repetition.