**CS673 Software Engineering**

**Team 3 - Rhettoric**

**Software Design Document**

| Team Member | Role(s) | Signature | Date |
| --- | --- | --- | --- |
| Magnus Urosev | Team Leader | *Magnus Urosev* | 5/28 |
| Adrian Ortiz | Configuration Leader / Security Leader |  |  |
| Xi Zeng | QA Leader | *Xi Zeng* | 6/17 |
| Jack Cairns | Requirements Leader |  |  |
| Jianing Li |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Revision history**

| **Version** | **Author** | **Date** | **Change** |
| --- | --- | --- | --- |
| **V 1.0** | **Magnus Urosev** | **5/28** | **Initial docs** |
| **V 1.1** | **Xi Zeng** | **5/29** | **ERD initial draft** |
| **V 1.2** | **Xi Zeng** | **6/5** | **Update new version of ERD and separate to make it clear to see** |
| **V 1.3** | **Magnus Urosev** | **6/10** | **Addressing previous feedback from Dr. Zhang** |
| **V 1.4** | **Xi Zeng** | **6/10** | **Revise ERD for DBs** |
| **V 1.5** | **Magnus Urosev** | **6/16** | **Added ORM design for survey** |
| **V 1.6** | **Xi Zeng** | **6/17** | **Specify the DB we are using** |

[Introduction](#_heading=h.gjdgxs)

[Software Architecture](#_heading=h.30j0zll)

[Class Diagram](#_heading=h.1fob9te)

[UI Design (if applicable)](#_heading=h.3znysh7)

[Database Design (if applicable)](#_heading=h.2et92p0)

[Security Design](#_heading=h.tyjcwt)

[Business Logic and/or Key Algorithms](#_heading=h.3dy6vkm)

[Design Patterns](#_heading=h.1t3h5sf)

[Any Additional Topics you would like to include.](#_heading=h.4d34og8)

[References](#_heading=h.2s8eyo1)

[Glossary](#_heading=h.17dp8vu)

# Introduction

In this section, give an overview of this document, and also address the design goals of your software system.

Rhettoric is a project focused on creating a survey. The project aims to provide accurate and reliable insight to a user's experience and feedback on a given course.

This is a project for the CS 673 class with Dr. Zhang. The objective is to create a survey for a course learning platform where at the end of the course, a survey is generated and distributed to the users to answer and provide valuable feedback to the course creator.

# Software Architecture

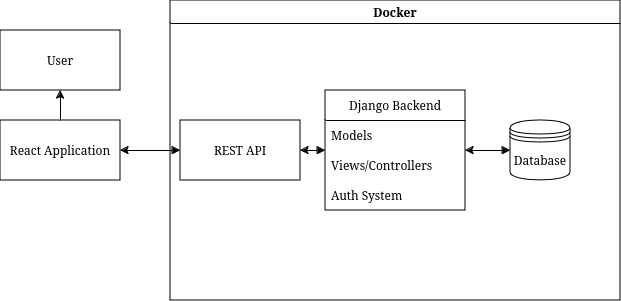
In our application architecture, Docker acts as the primary containerization framework, encapsulating the Django backend, REST API, and the database, ensuring consistency across development, testing, and production environments.

The backend is structured around Django, where it manages data models for users, surveys, and questions, along with the associated views and controllers for handling business logic and data manipulation.

The React frontend is designed to interact with the backend via REST API calls, facilitating dynamic data exchange necessary for real-time survey manipulation and user interaction.

This REST API allows communication between the React frontend and the Django backend. Authentication is handled through Django’s auth system, securing user interactions and enforcing role-based access controls.

The database serves as a placeholder but could be integrated within the Docker environment.



.

├── code

│ ├── docker-compose.yml

│ ├── Dockerfile.app

│ ├── Dockerfile.test

│ ├── README.md

│ └── survey\_app

│ ├── manage.py

│ ├── requirements.txt

│ ├── survey\_app

│ ├── surveys

│ └── users

├── demo

│ ├── CS673\_finaldemo\_team3.md

│ ├── CS673\_finalpresentation\_team3.pptx

│ └── Readme.md

├── doc

│ ├── CS673\_MeetingMinutes\_team3.docx

│ ├── CS673\_ProgressReport\_team3.xlsx

│ ├── CS673\_SDD\_team3.docx

│ ├── CS673\_SPPP\_RiskManagement.xlsx

│ ├── CS673\_SPPP\_team3.docx

│ ├── CS673\_STD\_team3.docx

│ ├── DB\_updated.png

│ ├── exam.drawio

│ ├── ProjectDescription.docx

│ ├── pull\_request\_template.md

│ └── Readme.md

├── iteration

│ ├── iteration0

│ │ ├── CS673\_presentation0\_team3

│ │ ├── CS673\_presentation0\_team3.pptx

│ │ └── Iteration 0 Team 3.pptx

│ ├── iteration1

│ │ ├── CS673\_iteration1demo\_team3 (video)

│ │ ├── CS673\_presentation1\_teamX

│ │ └── readme.md

│ └── iteration2

│ ├── CS673\_presentation2\_team3.pptx

│ ├── README.md

│ └── Team3\_Rhetoric\_Iteration2\_Presentation.mp4

├── README.md

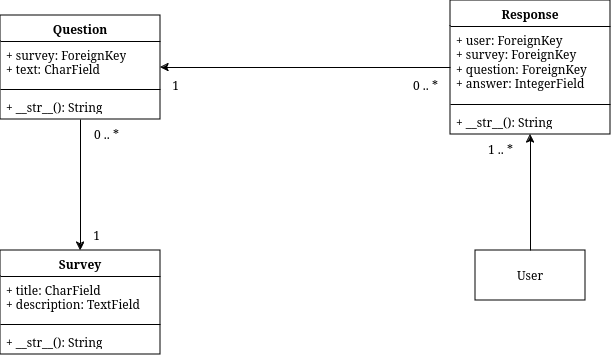
└── team.md

# Class Diagram

This diagram represents the UML class structure for Rhettoric handling surveys with Django models. The Survey model includes fields for title and description, each with corresponding data types CharField and TextField.

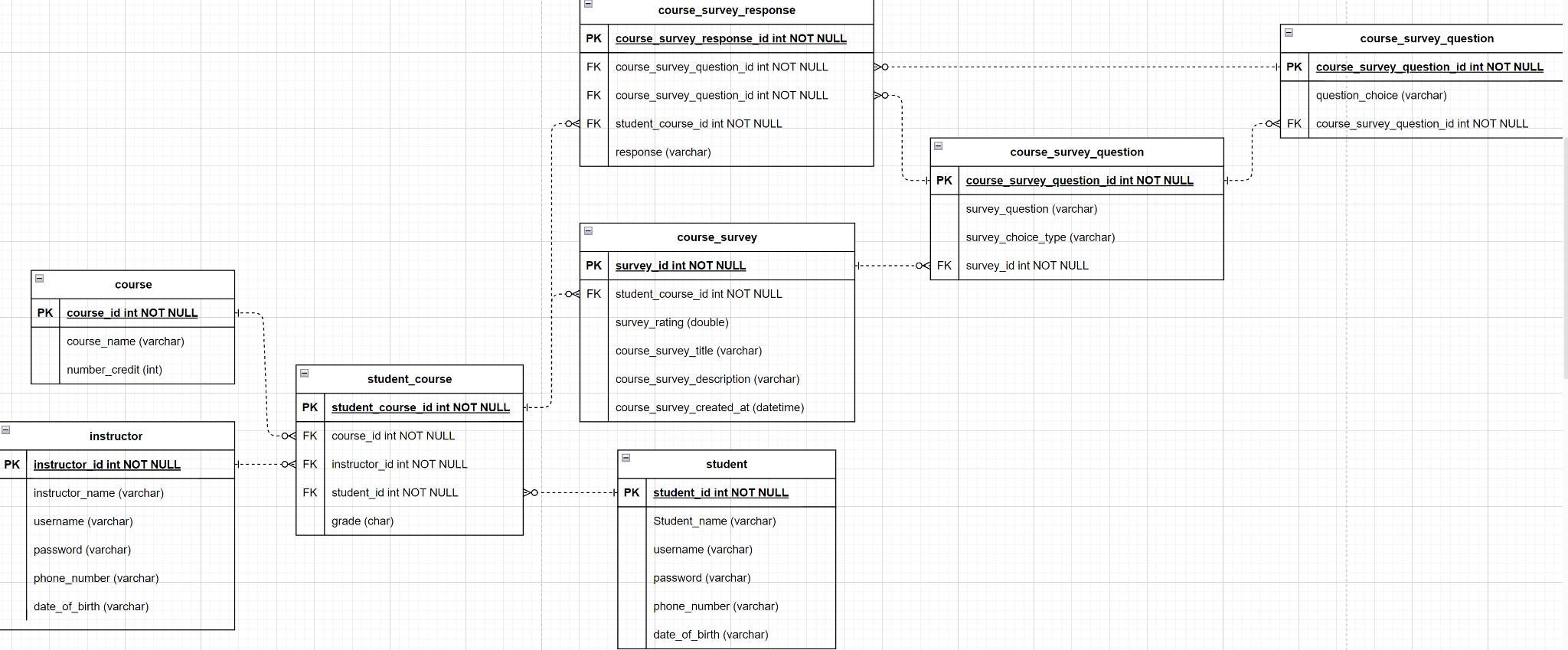
This model can associate with multiple Question models through a one-to-many relationship indicated by a ForeignKey. Each Question includes a text field of type CharField. Similarly, the Response model maps to User, Survey, and Question through ForeignKey relations, indicating that each response is linked to a specific user, survey, and question.

It also includes an answer field defined as an IntegerField with predefined choices. The relationships are set up to ensure that deleting a Survey cascades to delete its associated Questions and Responses, safeguarding data integrity and consistency across the application.



# Database Design

Below is a diagram showing the Database design. This is a hypothetical implementation.



# 

# Security Design

**1. Security Issues**

- Data Protection: Ensuring that user data, especially survey responses and personal information, is protected from unauthorized access and breaches.

- Authentication and Authorization: Ensuring only authorized users have access to specific parts of the application.

- Secure Data Transmission: Ensuring data transmitted between the client and server is encrypted to prevent interception.

- SQL Injection and Other Attacks: Preventing common web application attacks such as SQL injection, XSS, CSRF, etc.

**2. Security Requirements**

- User Authentication: Implement secure authentication mechanisms such as multi-factor authentication.

- Role-Based Access Control (RBAC): Implement RBAC to restrict access to different parts of the application based on user roles.

- Data Encryption: Use encryption for sensitive data storage and transmission.

- Input Validation: Implement strict input validation to prevent injection attacks.

- Regular Security Audits: Conduct regular security audits and vulnerability assessments.

**3. Security Design**

- Authentication and Authorization

- Implement Django’s built-in authentication system.

- Use libraries such as `django-allauth` for enhanced authentication features.

- Data Encryption

- Use HTTPS for secure data transmission.

- Store sensitive data encrypted in the database.

- Input Validation

- Use Django’s form validation features to ensure inputs are sanitized.

- Security Headers

- Implement security headers like Content Security Policy (CSP), X-Content-Type-Options, X-Frame-Options, and X-XSS-Protection.

**4. Security Tools**

- Django Security Features: Utilize built-in Django security features.

- SSL/TLS Certificates: Use services like Let's Encrypt to provide SSL/TLS certificates for HTTPS.

- Security Libraries: Use security libraries like `django-secure`, `django-axes`, and `djangorestframework-jwt`.

- Static Code Analysis Tools: Use tools like Bandit and SonarQube for static code analysis to detect security vulnerabilities.

**5. Implementation**

- Enable HTTPS

- Ensure the server is configured to serve the application over HTTPS.

- Authentication and Authorization

- Implement Django’s authentication and permission system.

- Example:

```python

settings.py

AUTHENTICATION\_BACKENDS = (

'django.contrib.auth.backends.ModelBackend',

'allauth.account.auth\_backends.AuthenticationBackend',

)

views.py

from django.contrib.auth.decorators import login\_required

from django.http import HttpResponse

@login\_required

def my\_view(request):

return HttpResponse("Hello, authenticated user!")

- Data Encryption

- Ensure sensitive data is encrypted.

- Example:

```python

from django.db import models

from cryptography.fernet import Fernet

key = Fernet.generate\_key()

cipher\_suite = Fernet(key)

class SensitiveData(models.Model):

encrypted\_data = models.BinaryField()

def save(self, args, kwargs):

self.encrypted\_data = cipher\_suite.encrypt(self.encrypted\_data.encode())

super().save(args, kwargs)

- Input Validation

- Use Django’s form validation features.

- Example:

```python

from django import forms

class SurveyForm(forms.Form):

question = forms.CharField(max\_length=200)

answer = forms.CharField(widget=forms.Textarea)

**6. Testing**

- Automated Security Tests

- Use tools like OWASP ZAP or Burp Suite for automated security testing.

- Penetration Testing

- Conduct regular penetration testing to identify vulnerabilities.

- Unit Tests

- Implement unit tests to ensure security features work as expected.

- Example:

```python

from django.test import TestCase

from django.urls import reverse

class SecurityTests(TestCase):

def test\_login\_required(self):

response = self.client.get(reverse('my\_view'))

self.assertRedirects(response, '/accounts/login/?next=/my\_view/')

# Business Logic and/or Key Algorithms

Algorithm: Survey Creation

* Facilitator Input:
  + Facilitator provides the survey title and description.
  + Facilitator provides a list of questions, each with text and choice type.
* Survey Creation:
  + Create a new survey instance with the given title and description.
  + Save the survey instance to the database.
* Question Assignment:
  + For each question provided by the facilitator. See class diagram for implementation:
    - Create a new question instance with the given text and choice type.
    - Assign the question to the created survey.
    - Save the question instance to the database.

# Design Patterns

As seen in the class diagram, the ORM (Object-Relational Mapping) design pattern utilized in the application efficiently maps the database schema to the object model within the application. This approach abstracts the database interactions into Python classes, allowing for more intuitive interactions with the database. In our implementation, the Survey, Question, and Response classes represent tables in the database.

Each class corresponds to a model in Django, where fields like CharField, TextField, and IntegerField map directly to database columns, and relationships between tables are managed through ForeignKey fields. This setup allows each Question to be linked to a specific Survey, and each Response to be connected to specific User, Survey, and Question instances, demonstrating a clear one-to-many relationship.

# 

# References

* Django Documentation: Official documentation for Django [Django Documentation](https://docs.djangoproject.com/)
* Django REST Framework: Documentation for using Django REST framework for creating REST APIs [DRF Documentation](https://www.django-rest-framework.org/)
* JSON Web Tokens (JWT): Documentation on JWT, an open standard for securely transmitting information. Available at: JWT Documentation

# Glossary

* Algorithm: A step-by-step procedure for calculations, data processing, and automated reasoning tasks.
* API (Application Programming Interface): A set of functions and protocols for building and interacting with software applications.
* Authentication: The process of verifying the identity of a user or process.
* Authorization: The process of verifying what a user is allowed to do.
* Django: A high-level Python web framework that encourages rapid development and clean, pragmatic design.
* JWT (JSON Web Token): A compact, URL-safe means of representing claims to be transferred between two parties.
* REST (Representational State Transfer): An architectural style for designing networked applications.
* Token: A piece of data that represents the right to perform some operation.