# Backend Documentation

Note that this documentation will change as more features are added, and the components properly connect to the front end.

## Main Function

```python
import os
import sys


def main():
    """Run administrative tasks."""
    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'codenow_api.settings')
    try:
        from django.core.management import execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and "
            "available on your PYTHONPATH environment variable? Did you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)


if __name__ == '__main__':
    main()
```

Djangos' main function executes the command line utility.

## Pipfile

```
[[source]]
url = "https://pypi.org/simple"
verify_ssl = true
name = "pypi"

[packages]
django = "*"
djangorestframework = "*"
djangorestframework-simplejwt = "*"
django-cors-headers = "*"

[dev-packages]

[requires]
python_version = "3.10"
```

Pipfile that contains the requirements and packages needed to use the functionality of the backend.

# URL Paths

```
from django.contrib import admin
from django.urls import include, path
from rest_framework import routers
from codenow_api.accounts import views
from .accounts.views import LoginView


router = routers.DefaultRouter()
router.register(r'users', views.UserViewSet)

# Wire up our API using automatic URL routing.
# Additionally, we include login URLs for the browsable API.
urlpatterns = [
    path('', include(router.urls)),
    path('api-auth/', include('rest_framework.urls', namespace='rest_framework')),
    path('login', LoginView.as_view(), name='login')
]
```

URL path folder contains all endpoints which can be reached via the frontend. Currently, it contains API-auth with a rest framework and a login endpoint.

# Serializers

```python
from django.contrib.auth.models import User
from rest_framework import serializers


class UserSerializer(serializers.HyperlinkedModelSerializer):
    class Meta:
        model = User
        fields = ['url', 'username', 'password', 'email']
```

Contains a serialization for view sets to use the fields when sending post/get requests through Django's local service. Currently, only a UserSerializer used for the login and sign-in functionality.

# Views

```python
class UserViewSet(viewsets.ModelViewSet):
    """
    API endpoint that allows users to be viewed or edited.
    """
    queryset = User.objects.all().order_by('-date_joined')
    serializer_class = UserSerializer


class LoginView(APIView):
    permission_classes = [AllowAny]

    @csrf_exempt
    def post(self, request):
        """
        API endpoint that authenticates user and returns a token
        """
        username = request.data.get("username")
        password = request.data.get("password")

        if username is None or password is None:
            return Response({'error': 'Please provide both username and password'},
                            status=HTTP_400_BAD_REQUEST)

        user = authenticate(username=username, password=password)

        if not user:
            return Response({'error': 'Invalid Credentials'},
                            status=HTTP_200_OK)

        token, _ = Token.objects.get_or_create(user=user)
        return Response({'token': token.key},
                        status=HTTP_200_OK)
```

View set file contains different classes of model view sets allowing users to log in or be viewed/edited and also generate auth tokens for users to use when login into an account.

```python
class AddProblemView(APIView):
    """
    Add Problem View
    """
```

```python
    @csrf_exempt
    def post(self, request):
        """
        API endpoint that will add a question to the prequiz table.
        """
        question_id = request.data.get("question_id")
        problem_name = request.data.get("problem_name")
        difficulty_level = request.data.get("difficulty_level")
        leetcode_url = request.data.get("leetcode_url")
        if None in (question_id, problem_name, difficulty_level, leetcode_url):
            return Response({'error': 'Please provide all necessary data'}, status=HTTP_400_BAD_REQUEST)

        prequiz_problem = PrequizProblem(question_id=question_id, difficulty_level=difficulty_level,
                                         problem_name=problem_name, leetcode_url=leetcode_url)
        prequiz_problem.save()
        return Response(status=200)
```

The endpoint outlined sends a post request to add a problem tot he prequiz problem database for further usage.

```python
class PrequizProblemsView(APIView):
    """
    Prequiz Problems Endpoint
    """

    @csrf_exempt
    def get(self, request):
        easy_problems = PrequizProblem.objects.order_by('?').filter(difficulty_level='easy')[:3]
        easy_problems_json = [{'question_id': p.question_id, 'problem_name': p.problem_name,
                               'difficulty_level': p.difficulty_level,
                               'leetcode_url': p.leetcode_url} for p in easy_problems]
        return Response({'problems': easy_problems_json})

    @csrf_exempt
    def post(self, request):
        problem_id = request.data.get('question_id')
        perceived_difficulty = request.data.get('perceived_difficulty')
        completion_time = request.data.get('completion_time')
        problem = PrequizProblem.objects.get(question_id=problem_id)
        problem.perceived_difficulty = perceived_difficulty
        problem.completion_time = completion_time
        problem.save()
        return HttpResponse(status=200)
```

The endpoint above is used for the quiz page to grab three random problems from the database and send them as JSON information on a GET request. The post request is to update their perceived difficulty and completion time.3