INSTITUTO TECNOLÓGICO DE ESTUDIOS SUPERIORES DE MONTERREY

Ingeniería en Tecnologías Computacionales (ITC)

Desarrollo e Implantación de Sistemas de Software (TC3005B.501)

Servicios Web

Equipo: CodeBreakers

Marco Flamenco Andrade	A01732313
Daniela Hernández y Hernández	A01730397
María José Burguete Euán	A01730344
Aarón Cortés García	A01730451

18 de mayo del 2022
Puebla, Pue.

Funcionalidades que atienden los servicios web del proyecto.

La plataforma web para el Programa Asesor Estudiante (PAE), requerirá de un servidor local configurado con todas las dependencias para poder desplegar tanto el Front End como el Back End de la página, incluyendo también la Base de Datos en PostgreSQL con todos los registros del dominio.

Por medio de las herramientas provenientes del *Object Relational Mapping* integrado con Django, es posible estructurar los métodos para la REST API que maneja la plataforma. Es por ello que, de manera general, las funcionalidades que atienden los servicios web del proyecto se resumen en operaciones de petición y respuesta para leer, actualizar, eliminar y crear nuevos registros en la Base de Datos (CRUD). Asimismo, para controlar la navegación en la interfaz, el acomodo y respuesta de sus elementos visuales, se requiere la intervención del servidor de acuerdo a los inputs del usuario.

Las funcionalidades más importantes que se están desarrollando para el proyecto en términos de servicios web, son las siguientes:

- ➔ Control de registro de usuarios en el sistema PAE (estudiantes, asesores y administradores).
- ➔ Manejo y mantenimiento de asesorías agendadas con todos sus parámetros necesarios.
- ➔ Reconocimiento de horarios disponibles ofrecidos por los asesores, junto con su banco de materias.
- ➔ Control de las encuestas de retroalimentación para cada tipo de usuario en el sistema.
- ➔ Reconocimiento simplificado de las horas de servicio social presentado por los actores de la plataforma.
- ➔ Mantenimiento de las carreras y materias disponibles en la oferta del sistema.
- ➔ Despliegue y navegación a través de la vista de la página.
- ➔ Modificación de los elementos visuales en la vista de acuerdo a ciertos inputs del usuario.

Determinar los diversos endpoints, métodos HTTP, y estructura del body para cada funcionalidad.

Hablando a nivel de red, los endpoints que se consideran para este sistema son el servidor en el cual se encontrarán los archivos web que componen la plataforma (frontend, backend y base de datos) y los dispositivos desde los cuáles el sitio sea accedido, es decir, las computadoras, laptops, celulares, tablets, etc, de los estudiantes, tutores y administrativos.

Por otra parte, teniendo en cuenta que este proyecto se maneja con una REST API desarrollada en Django, los endpoints en este caso equivaldrían a los puntos o canales de comunicación del sistema. De acuerdo a la arquitectura del Back End, las distintas funcionalidades requieren de un *serializer*, que 'mapea' cómo va a ser estructura la información de la petición web, esto es empleado dentro de un *viewset*, que define cómo se va a manejar la petición, en otras palabras, la query que será realizada a la base de datos. Finalmente, se liga el viewset correspondiente a un url específico, el cual es el lugar donde la API mandará y resolverá las peticiones web, siendo entonces estos últimos los endpoints.

ID	Funcionalidad	Endpoint	Método HTTP
1	Control, manejo y mantenimiento de las carreras cargadas en el sistema PAE	careers	POST, PUT, GET
2	Control, manejo y mantenimiento de los usuarios cargados en el sistema PAE	users	POST, PUT, GET, DELETE
3	Control, manejo y mantenimiento de los usuarios cargados en el sistema PAE especificando su tipo de cuenta (estudiantes, asesores y administradores)	pae_users	POST, PUT, GET, DELETE
4	Control, manejo y mantenimiento de las materias cargadas en el sistema PAE	subjects	POST, PUT, GET, DELETE
5	Control, manejo y mantenimiento de las materias ofrecidas por cada asesor dentro del sistema PAE	tutor_subjects	POST, PUT
6	Control, manejo y mantenimiento de las asesorías agendadas en el sistema PAE	sessions	POST, PUT, GET, DELETE
7	Mostrar las horas de asesoría disponibles de acuerdo a la materia seleccionada por el usuario	available_sessions	GET
8	Mostrar la información más relevante de asesorías asociadas a un estudiante en específico	sessions_of_specific_student	GET
9	Mostrar la información más relevante de asesorías asociadas a un tutor en específico	sessions_of_specific_tutor	GET
10	Mostrar la información más relevante de asesorías marcadas con estatus pendiente y que necesitan ser aprobadas por un administrador	pending_sessions	GET

11	Recuperar los registros de potenciales asesores para una asesoría en específico (materia y hora concreta), ordenados de menor a mayor horas de servicio	ordered_tutors_for_session	GET
12	Recuperar los registros de usuarios señalados como estudiantes dentro del sistema	students	GET
13	Recuperar los registros de usuarios señalados como asesores dentro del sistema	tutors	GET
14	Recuperar los registros de usuarios señalados como administradores dentro del sistema	admins	GET
15	Mostrar las horas de servicio acumuladas por un asesor en específico	service_hours	GET
16	Mostrar la oferta de materias indicada por un asesor en específico	subjects_by_tutor	GET
17	Mostrar las horas elegidas por un asesor en específico para impartir asesorías (señalando si se encuentran disponibles o no).	schedule_by_tutor	GET

Estructura del body de cada funcionalidad.

1. Serializer:

```
class CareerSerializer(ModelSerializer):
    class Meta:
        model = Career
        fields = ALL_FIELDS
```

ViewSet:

```
class CareersViewSet(ModelViewSet):
    queryset = Career.objects.all()
    serializer_class = CareerSerializer
    permission_classes = (AllowAny, )
```

2. Serializer:

```
class UserSerializer(ModelSerializer):
    class Meta:
        model = User
        fields = ALL_FIELDS
        extra_kwargs = {'password':{'required':True, 'write_only':True}}

    def create(self, validated_data):
        user = User.objects.create_user(**validated_data)
        Token.objects.create(user=user)
        return user
```

ViewSet:

```
class UsersViewSet(ModelViewSet):
    queryset = User.objects.all()
    serializer_class = UserSerializer
    authentication_classes = (TokenAuthentication, )
    permission_classes = (AllowAny, )
```

3. Serializer:

```
class PaeUserSerializer(ModelSerializer):
    class Meta:
        model = PaeUser
        fields = ALL_FIELDS
```

ViewSet:

```
class PaeUsersViewSet(ModelViewSet):
    queryset = PaeUser.objects.all()
    serializer_class = PaeUserSerializer
    authentication_classes = (TokenAuthentication, )
    permission_classes = (AllowAny, )
```

4. Serializer:

```
class SubjectSerializer(ModelSerializer):
    class Meta:
        model = Subject
        fields = ALL_FIELDS
```

ViewSet:

```
class SubjectsViewSet(ModelViewSet):
    queryset = Subject.objects.all()
    serializer_class = SubjectSerializer
    permission_classes = (AllowAny, )
```

5. Serializer:

```
class TutorSubjectSerializer(ModelSerializer):
    class Meta:
        model = TutorSubject
        fields = ALL_FIELDS
```

ViewSet:

```
class TutorSubjectsViewSet(ModelViewSet):
    queryset = TutorSubject.objects.all()
    serializer_class = TutorSubjectSerializer
    authentication_classes = (TokenAuthentication, )
    permission_classes = (AllowAny, )
```

6. Serializer:

```
class SessionAvailabilitySerializer(Serializer):
    id = IntegerField()
    id_tutor__id__username = CharField()
    id_tutor__schedule__day_hour = CharField()
    service_hours = IntegerField()
```

ViewSet:

```
class SessionsViewSet(ModelViewSet):
    queryset = Session.objects.all()
    serializer_class = SessionSerializer
    authentication_classes = (TokenAuthentication, )
    permission_classes = (AllowAny, )
```

7. Serializer:

```
class SessionAvailabilitySerializer(Serializer):
    id = IntegerField()
    id_tutor__id__username = CharField()
    id_tutor__schedule__day_hour = CharField()
    service_hours = IntegerField()
```

ViewSet:

```
class AvailableSessionsViewSet(mixins.ListModelMixin, viewsets.GenericViewSet):
    permission_classes = (AllowAny, )
    authentication_classes = (TokenAuthentication, )
    model = TutorSubject
    serializer_class = SessionAvailabilitySerializer
    def get_queryset(self):
        queryset = TutorSubject.objects.filter(id_tutor__schedule__available = True).annotate
        (service_hours = Count('id_tutor__session', filter=Q(id_tutor__session__status = 1))).
        order_by('service_hours').values('id', 'id_tutor__id__username',
        'id_tutor__schedule__day_hour', 'service_hours')
        subject = self.request.query_params.get('subject')
        if subject:
            queryset = queryset.filter(id_subject = subject)
        return queryset
```

8. Serializer:

```
class SessionCardSerializer(Serializer):
    id = IntegerField()
    id_subject__name = CharField()
    id_tutor__id__first_name = CharField()
    id_tutor__id__email = EmailField()
    id_student__id__first_name = CharField()
    id_student__id__email = EmailField()
    date = DateTimeField()
    spot = CharField()
    status = IntegerField()
```

ViewSet:

```python
class SessionsOfSpecificStudentViewSet(mixins.ListModelMixin, viewsets.GenericViewSet):
    permission_classes = (AllowAny, )
    authentication_classes = (TokenAuthentication, )
    model = Session
    serializer_class = SessionCardSerializer
    def get_queryset(self):
        queryset = Session.objects.all().values('id', 'id_subject__name',
        'id_tutor__id__first_name', 'id_tutor__id__email', 'id_student__id__first_name',
        'id_student__id__email', 'date', 'spot', 'status')
        student = self.request.query_params.get('student')
        if student:
            queryset = queryset.filter(id_student__id = student)
        return queryset
```

9. Serializer: comparte serializer con la funcionalidad 8.
   ViewSet:

```python
class SessionsOfSpecificTutorViewSet(mixins.ListModelMixin, viewsets.GenericViewSet):
    permission_classes = (AllowAny, )
    authentication_classes = (TokenAuthentication, )
    model = Session
    serializer_class = SessionCardSerializer
    def get_queryset(self):
        queryset = Session.objects.all().values('id', 'id_subject__name',
        'id_tutor__id__first_name', 'id_tutor__id__email', 'id_student__id__first_name',
        'id_student__id__email', 'date', 'spot', 'status')
        tutor = self.request.query_params.get('tutor')
        if tutor:
            queryset = queryset.filter(id_tutor__id = tutor)
        return queryset
```

10. Serializer: comparte serializer con la funcionalidad 8.
    ViewSet:

```python
class PendingSessionsViewSet(mixins.ListModelMixin, viewsets.GenericViewSet):
    permission_classes = (AllowAny, )
    authentication_classes = (TokenAuthentication, )
    model = Session
    serializer_class = SessionCardSerializer
    def get_queryset(self):
        queryset = Session.objects.filter(status = 0).values('id', 'id_subject__name',
        'id_tutor__id__first_name', 'id_tutor__id__email', 'id_student__id__first_name',
        'id_student__id__email', 'date', 'spot', 'status')
        return queryset
```

11. Serializer:

```python
class OrderedTutorsForSpecificSessionSerializer(Serializer):
    id_tutor__id__first_name = CharField()
    service_hours = IntegerField()
    id_subject = CharField()
    id_tutor__schedule__day_hour = CharField()
```

ViewSet:

```
class OrderedTutorsForSessionViewSet(mixins.ListModelMixin, viewsets.GenericViewSet):
    permission_classes = (AllowAny, )
    authentication_classes = (TokenAuthentication, )
    model = TutorSubject
    serializer_class = OrderedTutorsForSpecificSessionSerializer
    def get_queryset(self):
        subject = self.request.query_params.get('subject')
        dayHour = self.request.query_params.get('dayHour')
        if subject and dayHour:
            queryset = TutorSubject.objects.filter(id_tutor__schedule__available = True,
            id_subject = subject, id_tutor__schedule__day_hour = dayHour).annotate(service_hours
            = Count('id_tutor__session', filter=Q(id_tutor__session__status = 1))).order_by
            ('service_hours').values('id_tutor__id__first_name', 'service_hours', 'id_subject',
            'id_tutor__schedule__day_hour')
            return queryset
```

12. Serializer:

```
class UserDataSerializer(Serializer):
    id = IntegerField()
    id__first_name = CharField()
    career = CharField()
    semester = IntegerField()
```

ViewSet:

```
class StudentsViewSet(mixins.ListModelMixin, viewsets.GenericViewSet):
    permission_classes = (AllowAny, )
    authentication_classes = (TokenAuthentication, )
    model = PaeUser
    serializer_class = UserDataSerializer
    def get_queryset(self):
        queryset = PaeUser.objects.filter(user_type = 0).values('id', 'id__first_name', 'career',
        'semester')
        return queryset
```

13. Serializer: comparte serializer con la funcionalidad 12.

ViewSet:

```
class TutorsViewSet(mixins.ListModelMixin, viewsets.GenericViewSet):
    permission_classes = (AllowAny, )
    authentication_classes = (TokenAuthentication, )
    model = PaeUser
    serializer_class = UserDataSerializer
    def get_queryset(self):
        queryset = PaeUser.objects.filter(user_type = 1).values('id', 'id__first_name', 'career',
        'semester')
        return queryset
```

14. Serializer:

```
class AdminsSerializer(Serializer):
    id = IntegerField()
    first_name = CharField()
```

ViewSet:

```
class AdminsViewSet(mixins.ListModelMixin, viewsets.GenericViewSet):
    permission_classes = (AllowAny, )
    authentication_classes = (TokenAuthentication, )
    model = User
    serializer_class = AdminsSerializer
    def get_queryset(self):
        queryset = User.objects.filter(is_superuser = True).values('id', 'first_name')
        return queryset
```

15. Serializer:

```
class ServiceHoursSerializer(Serializer):
    id_tutor__id__first_name = CharField()
    service_hours = IntegerField()
```

ViewSet:

```
class ServiceHoursViewSet(mixins.ListModelMixin, viewsets.GenericViewSet):
    permission_classes = (AllowAny, )
    authentication_classes = (TokenAuthentication, )
    model = TutorSubject
    serializer_class = ServiceHoursSerializer
    def get_queryset(self):
        tutor = self.request.query_params.get('tutor')
        if tutor:
            queryset = TutorSubject.objects.filter(id_tutor = tutor).distinct().annotate
            (service_hours = Count('id_tutor__session', filter=Q(id_tutor__session__status = 1))).
            values('id_tutor__id__first_name', 'service_hours')
            return queryset
```

16. Serializer:

```
class SubjectsByTutorSerializer(Serializer):
    id_subject__name = CharField()
```

ViewSet:

```
class SubjectsByTutorViewSet(mixins.ListModelMixin, viewsets.GenericViewSet):
    permission_classes = (AllowAny, )
    authentication_classes = (TokenAuthentication, )
    model = TutorSubject
    serializer_class = SubjectsByTutorSerializer
    def get_queryset(self):
        tutor = self.request.query_params.get('tutor')
        queryset = TutorSubject.objects.filter(id_tutor = tutor).values('id_subject__name')
        return queryset
```

17. Serializer:

```
class ScheduleByTutorSerializer(Serializer):
    day_hour = CharField()
    available = BooleanField()
```

ViewSet:

```python
class ScheduleByTutorViewSet(mixins.ListModelMixin, viewsets.GenericViewSet):
    permission_classes = (AllowAny, )
    authentication_classes = (TokenAuthentication, )
    model = Schedule
    serializer_class = ScheduleByTutorSerializer
    def get_queryset(self):
        tutor = self.request.query_params.get('tutor')
        queryset = Schedule.objects.filter(id_user = tutor).values('day_hour', 'available')
        return queryset
```

Establecer validaciones, códigos de error, y respuestas de los servicios.

Dentro del sistema PAE, las principales validaciones y manejo de errores se relacionan directamente con las operaciones hechas en la Base de Datos, tratando de reducir el número de peticiones fallidas y la inserción de información inconsistente. El primer filtro para validar los datos, es revisar la verificación con herramientas puras de HTML y Bootstrap en los formularios, inclusive permitiendo la escritura de expresiones regulares específicas para cada campo. Gracias a esto, es posible comprobar que los nombres de los usuarios no contengan números o caracteres especiales; que las contraseñas contengan al menos una minúscula, una mayúscula y un número; que la matrícula y el email guarden el formato de la institución, etc.

El siguiente paso es hacer la solicitud directa a la BD con los inputs escritos en el formulario, donde entra en acción la validación propia del protocolo HTTP con sus códigos de estado de respuesta. Dichos códigos pueden resultar en cualquiera de las variantes y clasificaciones, logrando indicar respuestas satisfactorias, respuestas informativas, redirecciones, errores de cliente y errores de servidor. A continuación se presentan los códigos de estado más frecuentes durante el desarrollo y uso de la plataforma web:

- ➔ **200 - Resolved request successfully.** Se hace una solicitud al servidor con la dirección correcta y los parámetros adecuados si es el caso, resultando en la aplicación de un método satisfactoriamente.
- ➔ **300 - Redirect / Not authenticated for that url.** Se comprueba la existencia de la dirección solicitada, pero esta ha sufrido cambios para su visualización o el usuario no cuenta con las credenciales necesarias para acceder a ella.
- ➔ **400 - Bad request.** Se comprueba la existencia de la dirección solicitada, pero los parámetros ingresados para aplicar el método no son los adecuados. Esto puede pasar, por ejemplo, a la hora de solicitar el POST de un nuevo usuario, pero con campos vitales vacíos, como la matrícula o la contraseña.
- ➔ **404 - Not found.** Se hace una solicitud a la API con un url que no existe, ya sea por que fue eliminado o la sintaxis de la petición fue incorrecta.
- ➔ **500 - Internal server error.** Ocurre cuando el servidor está fuera de línea o inaccesible desde el endpoint donde se realizó la petición..