TRABAJO DE GRADO
Opción Seminario-Diplomado.

Seminario desarrollo Móvil

Sistema de Gestión académica

Corporación Universitaria Remington

Ingeniería de sistemas

Seminario de investigación

Mauricio Alejandro Triviño Bedoya y Santiago Torres Parrado

Guillermo Antonio Tobón Barco

Opción de Trabajo de grado Seminario-Diplomado

2025

Tabla de Contenidos

1. Resumen

Este informe técnico documenta el desarrollo e implementación de la aplicación móvil "**Gestión de Información Académica**", una solución diseñada para modernizar y optimizar la administración de datos académicos en instituciones educativas. El proyecto fue concebido para resolver ineficiencias en la gestión manual de materias, la asignación de docentes, el registro y consulta de calificaciones, y la comunicación entre la institución, profesores y estudiantes.

La aplicación se desarrolló utilizando una metodología Ágil (Scrum), lo que permitió una entrega incremental y adaptativa del software. Se estructuró en tres zonas de usuario principales:

- Administrador: Gestiona materias, docentes, estudiantes, curso o carrera y aulas, asigna docentes, materias, aulas, cursos o carreras y estudiantes a grupos.
- Profesor: Carga calificaciones parciales y definitivas para sus estudiantes dentro de plazos establecidos a la vez puede visualizar su agenda diaria.
- Estudiante: Consulta su información académica, incluyendo materias matriculadas y el detalle de sus calificaciones (parciales y ponderadas), a la vez puede visualizar su agenda diaria.

Técnicamente, el Frontend de la aplicación móvil se construyó con Ionic Framework y ReactJS. Esta elección facilitó un desarrollo eficiente, permitiendo el despliegue en dispositivos Android desde una única base de código. El Backend fue implementado con Spring Boot de Java, encargado de la lógica de negocio y la persistencia de datos en una base de datos PostgreSQL. La comunicación entre el frontend y el backend se realizó mediante APIs REST, garantizando un flujo de información seguro. La seguridad del sistema, incluyendo la autenticación y autorización por roles (Administrador, Profesor, Estudiante), se

manejó con Spring Security y JSON Web Tokens (JWT). Para la colaboración y el control de cambios, se utilizó GitHub.

Los resultados obtenidos demuestran la aplicación práctica de los conocimientos adquiridos en el seminario de aplicaciones móviles, especialmente en el desarrollo de aplicaciones móvil y la sintaxis de ReactJS. La aplicación ha logrado digitalizar y centralizar efectivamente la gestión académica, ofreciendo una herramienta segura, accesible y eficiente que mejora la transparencia y la comunicación en el entorno educativo

2. Palabras clave

- Android: Es un sistema operativo móvil, principalmente para teléfonos inteligentes y tabletas, basado en el núcleo Linux y otros componentes de código abierto.
- Ionic: Es un framework de código abierto, basado en tecnologías web como HTML, CSS y JavaScript, que permite a los desarrolladores crear aplicaciones móviles híbridas multiplataforma.
- APIs REST: Es una interfaz de programación de aplicaciones que sigue los principios del estilo arquitectónico REST. Se utiliza para la comunicación entre sistemas informáticos a través de Internet, utilizando métodos HTTP como GET, POST, PUT y DELETE para acceder y manipular recursos.
- Java: Lenguaje de programación y plataforma informática creado por Sun Microsystems (ahora propiedad de Oracle) y lanzado en 1995. Es ampliamente utilizado para desarrollar una variedad de aplicaciones, incluyendo software empresarial, aplicaciones móviles, y aplicaciones web.

- ReactJS: Es una biblioteca de JavaScript de código abierto, desarrollada por Facebook, para construir interfaces de usuario interactivas y dinámicas. Se enfoca en la creación de componentes reutilizables que permiten construir aplicaciones web complejas de manera eficiente y organizada.

3. Marco conceptual y contextual

3.1. Marco Conceptual y Contextual del Backend

El backend de la aplicación es el componente central que aloja la lógica de negocio, gestiona la persistencia de los datos y asegura la comunicación con el frontend. Su diseño y construcción se basaron en principios de arquitectura de software y tecnologías clave que garantizan su robustez, seguridad y escalabilidad.

3.1.1. Conceptos y Definiciones Requeridas:

- Concepto: Las APIs RESTful son un estilo arquitectónico para sistemas distribuidos, que permite la comunicación entre diferentes componentes de software utilizando el protocolo HTTP. Se basan en recursos ejemplo (materias, /usuarios) a los que se accede mediante métodos HTTP estándar (GET, POST, PUT, DELETE). Son fundamentales para las aplicaciones móviles, ya que proveen una interfaz estandarizada y sin estado para que el cliente (frontend) pueda solicitar y enviar datos al servidor.

- Contexto del Trabajo: En el proyecto "Gestión de Información Académica", el backend expone APIs RESTful para todas las operaciones, permitiendo que la aplicación móvil (frontend) inicie sesión, consulte calificaciones, gestione usuarios o cargue notas. Esta fue una de las decisiones arquitectónicas clave para la interconexión cliente-servidor.

3.1.2. Spring Boot (Framework de Java):

- **Concepto:** Spring Boot es un *framework* de código abierto basado en el lenguaje Java que simplifica enormemente el desarrollo de aplicaciones Spring, especialmente las APIs RESTful y los microservicios. Permite crear aplicaciones auto-ejecutables con una configuración mínima, ofreciendo una alta productividad al desarrollador.
- **Contexto del Trabajo:** La elección de Spring Boot para el backend se alinea con la necesidad de construir una API robusta y rápidamente. Su ecosistema de librerías, como Spring Data JPA para la base de datos y Spring Security para la seguridad, fue crucial para implementar eficientemente todas las funcionalidades de gestión académica y control de acceso.

3.1.3. Bases de Datos Relacionales (PostgreSQL):

- **Concepto:** Las bases de datos relacionales organizan los datos en tablas con filas y columnas, estableciendo relaciones predefinidas entre ellas. PostgreSQL es un sistema de gestión de bases de datos relacional de objetos (ORDBMS) conocido por su fiabilidad, robustez, rendimiento y soporte para estándares SQL.

- **Contexto del Trabajo:** PostgreSQL fue seleccionada como la base de datos principal para almacenar toda la información estructurada de la aplicación (usuarios, materias, calificaciones, etc.). Su capacidad para asegurar la integridad de los datos y manejar transacciones complejas fue vital para un sistema académico donde la precisión de la información es crítica.

3.1.4. Seguridad en Aplicaciones (Spring Security y JWT):

- **Concepto:** La seguridad en aplicaciones se refiere a las medidas implementadas para proteger los datos y funcionalidades del acceso no autorizado. Spring Security es un *framework* de seguridad potente para aplicaciones Java, mientras que JSON Web Tokens (JWT) son tokens seguros que permiten la autenticación y el intercambio de información de manera compacta y segura. Son esenciales para **proteger las APIs de las aplicaciones móviles**.
- **Contexto del Trabajo:** La implementación de Spring Security con JWT en el backend fue fundamental para autenticar a los usuarios (Administradores, Profesores, Estudiantes) y autorizar su acceso a funcionalidades específicas según su rol. Esto asegura que, por ejemplo, solo un profesor pueda cargar calificaciones o solo un administrador pueda crear usuarios, protegiendo la integridad de la información académica.

3.1.5. Arquitectura Hexagonal:

- **Concepto:** Un patrón arquitectónico que divide una aplicación en capas lógicas ejemplo (presentación, lógica de negocio, persistencia) con

responsabilidades específicas. Cada capa se comunica solo con las capas adyacentes, promoviendo la modularidad, la mantenibilidad y la escalabilidad.

- **Contexto del Trabajo:** El backend de la aplicación "Gestión de Información Académica" se estructuró en capas claras (Controladores, Servicios, Repositorios), lo que facilitó el desarrollo modular, el testeo aislado de la lógica de negocio y una separación nítida de responsabilidades, optimizando el trabajo en equipo.

3.1.6. Contexto del Trabajo del Backend:

- El backend de la aplicación se contextualiza como la columna vertebral digital que soporta las operaciones de una institución educativa. Su desarrollo estuvo directamente influenciado por la necesidad de centralizar y automatizar procesos que anteriormente eran manuales y propensos a errores. Se construyó para ser un punto de verdad único para los datos académicos, proveyendo un acceso seguro y eficiente a la información. La elección de Java con Spring Boot y PostgreSQL se basó en la robustez y escalabilidad necesarias para gestionar el volumen de datos de estudiantes, profesores y materias, y para asegurar que la aplicación móvil tuviera una base de datos fiable y transaccional. La seguridad implementada es vital en un entorno donde la confidencialidad de las calificaciones y la información personal es primordial.

3.2. Marco Conceptual y Contextual del Frontend

- El frontend de la aplicación "Gestión de Información Académica" es la interfaz directa con el usuario, diseñada para ser intuitiva y accesible desde

dispositivos móviles. Su desarrollo se basó directamente en los principios y tecnologías abordados en el seminario de aplicaciones móviles.

3.2.1. Ionic Framework:

- **Concepto:** Un *framework* de código abierto que permite construir aplicaciones móviles híbridas y PWA (Progressive Web Apps) utilizando tecnologías web y *frameworks* JavaScript populares como React, Angular o Vue.js. Proporciona componentes de UI pre-diseñados que simulan el aspecto y la sensación nativa. Fue una **herramienta principal enseñada en el seminario**.
- **Contexto del Trabajo:** Ionic fue la elección para la capa visual de la aplicación. Sus componentes UI listos para usar y su integración con ReactJS permitieron construir rápidamente interfaces atractivas y funcionales para las diferentes zonas de usuario (Administrador, Profesor, Estudiante), siguiendo las directrices de diseño móvil enseñadas.

3.2.2. ReactJS (Librería de JavaScript):

- **Concepto:** Una librería de JavaScript para construir interfaces de usuario declarativas y basadas en componentes. Permite crear "componentes" UI reutilizables que gestionan su propio estado, facilitando el desarrollo de aplicaciones complejas con alto grado de interactividad. La **sintaxis y el paradigma de React fueron un pilar del seminario**.

- **Contexto del Trabajo:** La familiaridad con la sintaxis y los principios de ReactJS, adquirida en el seminario, fue directamente aplicada en la construcción de cada pantalla y componente del frontend. Esto permitió estructurar la aplicación de manera modular, gestionar el estado de forma eficiente (ej., el estado de los formularios de carga de notas, el usuario autenticado) y crear una experiencia de usuario dinámica.

3.2.3. Manejo de Estado (React Hooks y Context API):

- **Concepto:** Se refiere a cómo se gestionan y actualizan los datos que controlan el comportamiento y la apariencia de la interfaz de usuario. React Hooks (useState, useEffect) manejan el estado local, mientras que la Context API de React (o librerías externas) manejan el estado global que es accesible por múltiples componentes.
- **Contexto del Trabajo:** Para "Gestión de Información Académica", el manejo de estado fue crucial. Por ejemplo, el estado del usuario logueado (su rol y token JWT) se manejó globalmente para proteger rutas, mientras que el estado de los campos de entrada al cargar calificaciones se manejó localmente en el componente del profesor, optimizando el rendimiento y la coherencia de la UI.

3.2.4. Consumo de APIs (Cliente-Side):

- **Concepto:** Proceso mediante el cual el frontend realiza peticiones HTTP a los *endpoints* expuestos por el backend para obtener, enviar, actualizar o eliminar

datos. Es el puente de comunicación entre la interfaz de usuario y la lógica de negocio del servidor.

- **Contexto del Trabajo:** La aplicación móvil interactúa constantemente con el backend a través de APIs REST. El seminario nos preparó para entender cómo realizar estas peticiones de manera eficiente y asíncrona, cómo manejar las respuestas (datos JSON) y cómo gestionar los errores, lo cual fue vital para la funcionalidad de la aplicación, como el cargue de notas o la consulta de horarios.

3.2.5. Contexto del Trabajo del Frontend:

El frontend de "Gestión de Información Académica" se contextualiza como la puerta de acceso digital a la vida académica de la institución, diseñada para ser accesible en cualquier momento y lugar a través de dispositivos móviles. El seminario de aplicaciones móviles fue el catalizador que permitió transformar los requisitos funcionales en una aplicación interactiva. La adopción de Ionic y ReactJS, directamente influenciada por la formación recibida, fue clave para:

- **Crear una UI intuitiva:** Con zonas claras para administradores (gestión), profesores (cargue de notas) y estudiantes (consulta de notas), cada una con una experiencia adaptada a sus necesidades.
- **Proporcionar retroalimentación inmediata:** Gracias al manejo de estado y el consumo asíncrono de APIs, la interfaz responde rápidamente a las acciones del usuario, mejorando la experiencia general.

En resumen, el frontend es la manifestación práctica de los conocimientos adquiridos en el seminario, enfocado en resolver las problemáticas de acceso a la información y gestión académica directamente desde el dispositivo móvil del usuario.

4. Desarrollo e implementación del aprendizaje

4.1. La arquitectura de la aplicación móvil

La arquitectura de la aplicación móvil "Gestión de Información Académica" se basa en un modelo cliente-servidor, que garantiza una separación clara de responsabilidades y facilita tanto el desarrollo como el mantenimiento.

- Aplicación Móvil: Se eligió el formato de aplicación móvil por su practicidad y accesibilidad, permitiendo a los usuarios interactuar con la plataforma desde cualquier lugar y en cualquier momento a través de sus dispositivos inteligentes.
- Frontend (Cliente): La parte de la aplicación con la que el usuario interactúa directamente se desarrolló utilizando Ionic Frameworks y ReactJS. Esta elección se basó en su capacidad para crear interfaces de usuario responsivas, lo que agiliza el desarrollo para Android desde una única base de código.
- Backend (Servidor): La lógica de negocio, la gestión de datos y la seguridad son manejadas por el Backend. Este componente fue implementado utilizando el framework Spring Boot de Java. Spring Boot es una tecnología robusta y escalable, ideal para construir servicios web eficientes.
- APIs (Application Programming Interfaces): La comunicación entre el frontend (la aplicación móvil) y el backend (el servidor) se realiza a través de APIs REST (Representational State Transfer). Estas interfaces actúan como un puente que

permite el intercambio de datos de manera estandarizada y segura, asegurando que la información fluya correctamente entre las diferentes capas de la aplicación.

- Base de Datos: Para la persistencia de la información académica (materias, docentes, estudiantes, calificaciones), se utilizó la base de datos PostgreSQL. Esta base de datos relacional es conocida por su fiabilidad, robustez y capacidad para manejar grandes volúmenes de datos de manera eficiente.

4.2. Visión General y Arquitectura

El frontend de la aplicación "Gestión de Información Académica" es la capa con la que los usuarios interactúan directamente. Su objetivo principal es proporcionar una interfaz de usuario intuitiva y fluida que permita a los administradores, profesores y estudiantes gestionar y visualizar la información académica de manera eficiente.

Arquitectura de Alto Nivel: La aplicación móvil se desarrolló con una arquitectura basada en componentes ReactJS, esta promueve la reutilización de código, la modularidad y una gestión más sencilla del estado de la aplicación.

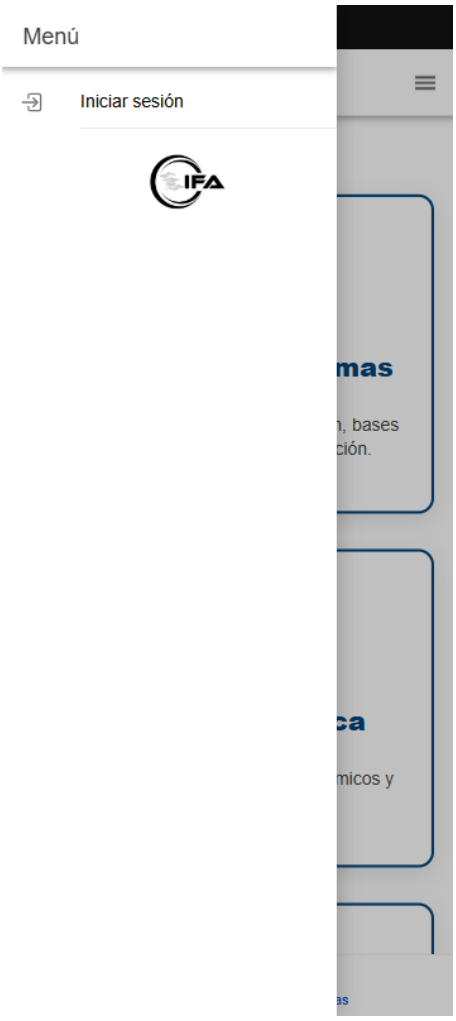
- Capa de Presentación (Vistas y Componentes): Es el nivel más visible para el usuario. Se compone de "páginas" o "vistas" ejemplo (Login, Dashboard Administrativo, Vista de Notas del Profesor, Vista de Notas del Estudiante) que a su vez están construidas a partir de componentes más pequeños y reutilizables, por ejemplo (botones, campos de entrada, tarjetas de calificaciones, tablas).

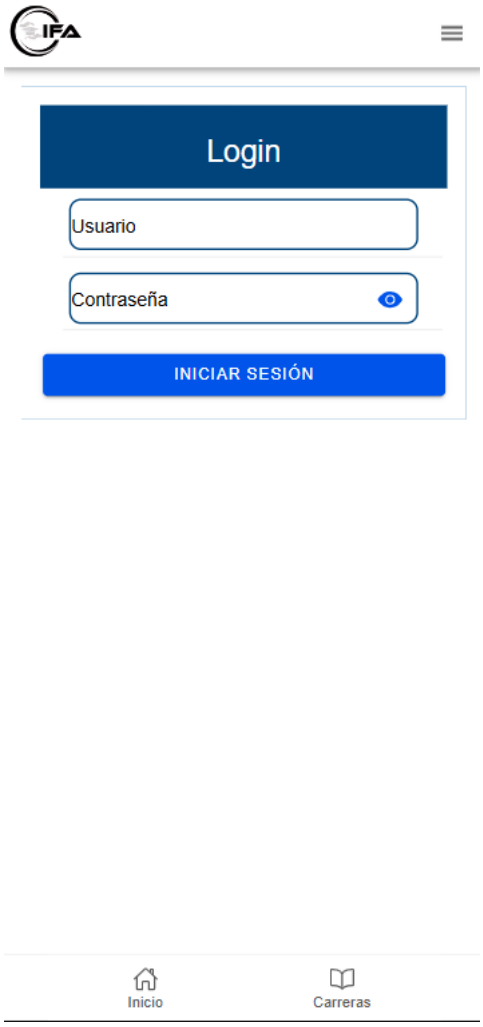
4.2.1.Home:


Corporación Universitaria Remington
Bienvenido al sistema de gestión de notas
Accede a tus calificaciones de manera rápida y segura.
Gestión Completa de Notas
Crea, edita y elimina tus notas con una interfaz simple y clara.
Inicio
Carreras

4.2.2.Home carreras

4.2.3.Login:





4.2.4.Manejo de rutas según su rol:

Aquí podemos evidenciar 7 imágenes, que corresponden al manejo de rutas según el rol del usuario que ingresa, dependiendo de esto es que le permite su funcionalidad dentro de la aplicación

4.2.5.Rutas según el rol

App.jsx

```
1   import ...
52  const routes : [{path: string, role: string[]... = [
53      {
54          id: 100,
55          icon: personOutline,
56          path: '/profile',
57          role: ['ROLE_STUDENT','ROLE_TEACHER'],
58          label: 'Perfil',
59      },
60      {
61          id: 1,
62          icon: schoolOutline,
63          path: '/student/periods',
64          role: ['ROLE_STUDENT'],
65          label: 'Calificaciones',
66      },
67      {
68          id: 2,
69          icon: calendarOutline,
70          path: '/schedule',
71          role: ['ROLE_STUDENT', 'ROLE_TEACHER'],
72          label: 'Horarios',
73      },
74      {
75          id: 3,
76          icon: schoolOutline,
77          path: '/teacher/periods',
78          role: ['ROLE_TEACHER'],
79          label: 'Asignaturas',
80      },
81      {
82          id: 4,
```

4.2.6.Rutas según el rol

App.jsx ×

```
const routes : [{path: string, role: string[]... = [
        label: 'Asignaturas',
    },
    {
        id: 4,
        icon: bookOutline,
        path: '/create/periods',
        role: ['ROLE_ADMIN'],
        label: 'Asignar materias',
    },
    {
        id: 5,
        icon: personAdd,
        path: '/create/students',
        role: ['ROLE_ADMIN'],
        label: 'Crear estudiantes',
    },
    {
        id: 6,
        icon: personAdd,
        path: '/create/teachers',
        role: ['ROLE_ADMIN'],
        label: 'Crear profesores',
    },
    {
        id: 7,
        icon: newspaper,
        path: '/create/subjects',
        role: ['ROLE_ADMIN'],
        label: 'Crear materias',
    },
    {
```

4.2.7.Rutas según el rol y boton de Logout

App.jsx

```
const routes : [{path: string, role: string[]... = [
        role: ['ROLE_ADMIN'],
        label: 'Crear materias',
    },
    {
        id: 8,
        icon: business,
        path: '/create/classRooms',
        role: ['ROLE_ADMIN'],
        label: 'Crear aulas',
    },
];

setupIonicReact();

export default function App() {  Show usages  sanTorres770 +1

    const {login,handlerLogout} = useApp()

    const handleLogout = () : void  => {  Show usages  CamiRinconR +1
        handlerLogout();
        toast.info( content: 'Sesión cerrada correctamente');

        setTimeout( handler: () : void  => {
            window.location.href = '/startPage';

        },  timeout: 1500)
    };

    const isVerified = sessionStorage.getItem( key: 'verified') === 'true'
```

4.2.8.Verificación de autentificación y muestra de rutas según el rol

```
return (
    <IonApp>
        <IonReactRouter forceRefresh={true}>
            <IonMenu contentId="main-content">
                <IonHeader>
                    <IonToolbar>
                        <IonTitle>Menú</IonTitle>
                    </IonToolbar>
                </IonHeader>
                <IonContent>
                    <IonList>

                        {(!login.isAuth || !isVerified) && (
                            <IonItem routerLink={'/login'} routerDirection="none">
                                <IonIcon slot="start" icon={logInOutline}/>
                                <IonLabel>Iniciar sesión</IonLabel>
                            </IonItem>
                        )}

                        {routes.map(route :{…}  => ((login.isAuth && isVerified) && (route.role.includes(login?.authorities[0].authority))) && (

                            <IonItem key={route.id} routerLink={route.path} routerDirection="none">
                                <IonIcon slot="start" icon={route.icon}/>
                                <IonLabel>{route.label}</IonLabel>
                            </IonItem>

                        ))}

                        {(login.isAuth && isVerified) && (
                            <IonItem button={true} onClick={() :void => handleLogout()}>
```

4.2.9.Verificación de autentificación y muestra de rutas según el rol

```
                                <IonIcon slot="start" icon={logOut} />
                                <IonLabel>Cerrar sesión</IonLabel>
                            </IonItem>
                        )}

                        <IonItem lines={'none'} className={'ion-margin-top'}>
                            <IonIcon slot="start" icon={optionsOutline} className={'ion-page-invisible'}/>
                            <IonIcon slot="start" icon={optionsOutline} className={'ion-page-invisible'}/>
                            <img src="../logo-ifa.png" alt="d" className="sk-cube sk-cube1" width={'50%'}/>
                        </IonItem>

                    </IonList>
                </IonContent>
            </IonMenu>

            <IonPage id="main-content">
                <IonHeader>
                    <IonToolbar style={{marginTop: '40px'}}>
                        <IonButtons slot="end">
                            <IonMenuButton/>
                        </IonButtons>
                        <img src="../logo-ifa.png" alt="d" className="sk-cube sk-cube1" width={'20%'}/>
                    </IonToolbar>
                </IonHeader>
                <IonContent>
                    <IonTabs>
                    <IonRouterOutlet>
```

4.2.10. Rutas

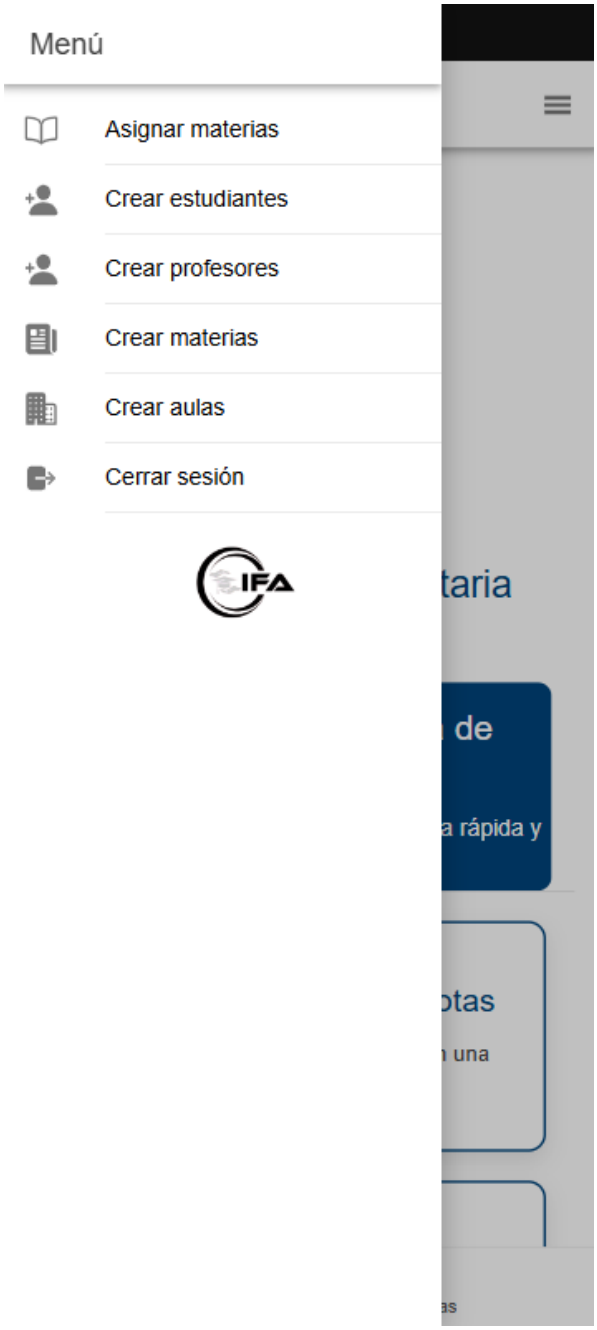
```
                        <Route exact path="/startPage" component={StartPage} />
                        <Route exact path="/courses" component={CoursePage} />
                        <Route exact path="/profile" component={ProfilePage} />
                        <Route exact path="/tabs" component={TabsPage} />
                        <Route exact path="/login" component={Login} />
                        <Route exact path="/verify" component={VerifyUser} />
                        <Route path="/student/periods" render={() => <PeriodsListStudent/>} exact={true}/>
                        <Route path="/schedule" render={() => <ScheduleView/>} exact={true}/>
                        <Route path="/teacher/periods" render={() => <PeriodsListTeacher/>} exact={true}/>
                        <Route path="/create/periods" render={() => <CreatePeriodForm/>} exact={true}/>
                        <Route path="/create/students" exact>
                            <NewStudentForm />
                        </Route>
                        <Route path="/create/teachers" exact>
                            <NewTeacherForm />
                        </Route>
                        <Route path="/create/subjects" exact>
                            <NewSubjectForm />
                        </Route>
                        <Route path="/create/classRooms" exact>
                            <NewClassRoomForm />
                        </Route>
                        <Redirect to="/startPage"/>
                    </IonRouterOutlet>
                    <IonTabBar slot="bottom">

                        <IonTabButton tab={'startPage'} href={'/startPage'}>
```

4.2.11. Rutas y fin de estructura

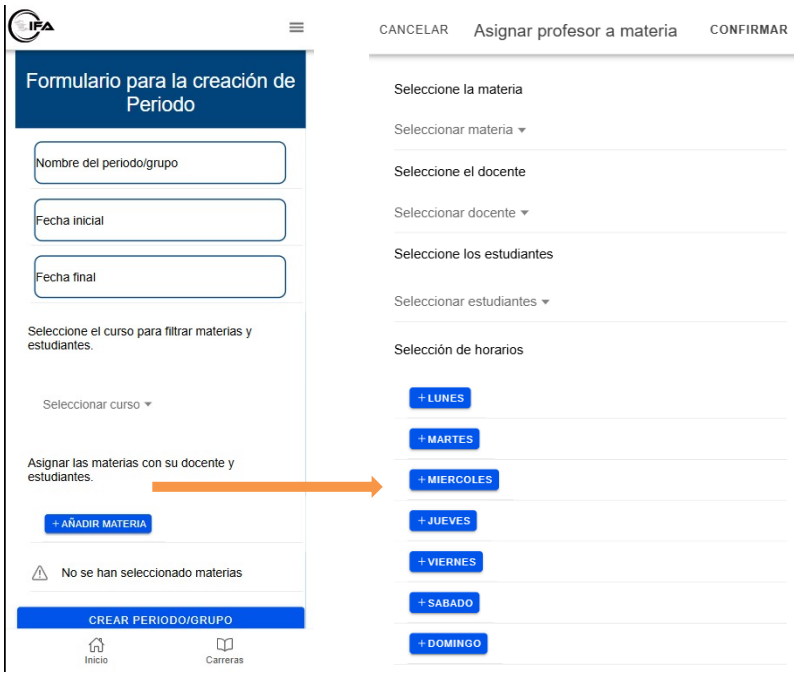
```
                            <IonTabButton tab={'startPage'} href={'/startPage'}>
                                <IonIcon icon={homeOutline}/>
                                <IonLabel>Inicio</IonLabel>
                            </IonTabButton>

                            <IonTabButton tab={'courses'} href={'/courses'}>
                                <IonIcon icon={bookOutline}/>
                                <IonLabel>Carreras</IonLabel>
                            </IonTabButton>

                        </IonTabBar>
                    </IonTabs>
                </IonContent>
            </IonPage>
        </IonReactRouter>
        <ToastContainer/>
    </IonApp>
  );
}
```

5. Administrador:

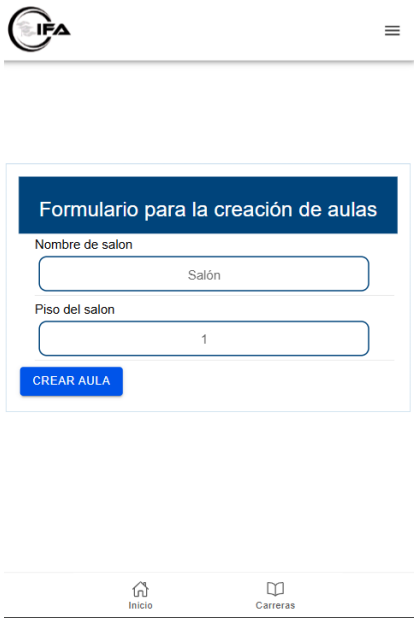
5.1. Menu:



5.2. Registro de grupos:



5.3. Registro de aulas:



5.4. Registro de materias:

5.5. Registro de estudiantes:

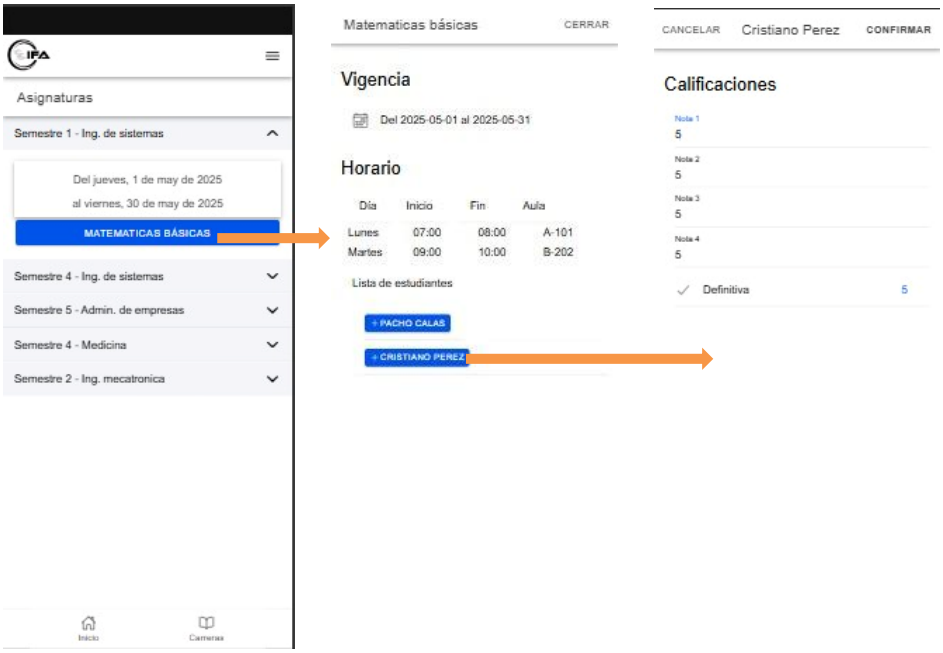
5.6. Registro de docentes:

6. Docente:

6.1. Menu:

6.2. Consultar agenda diaria:

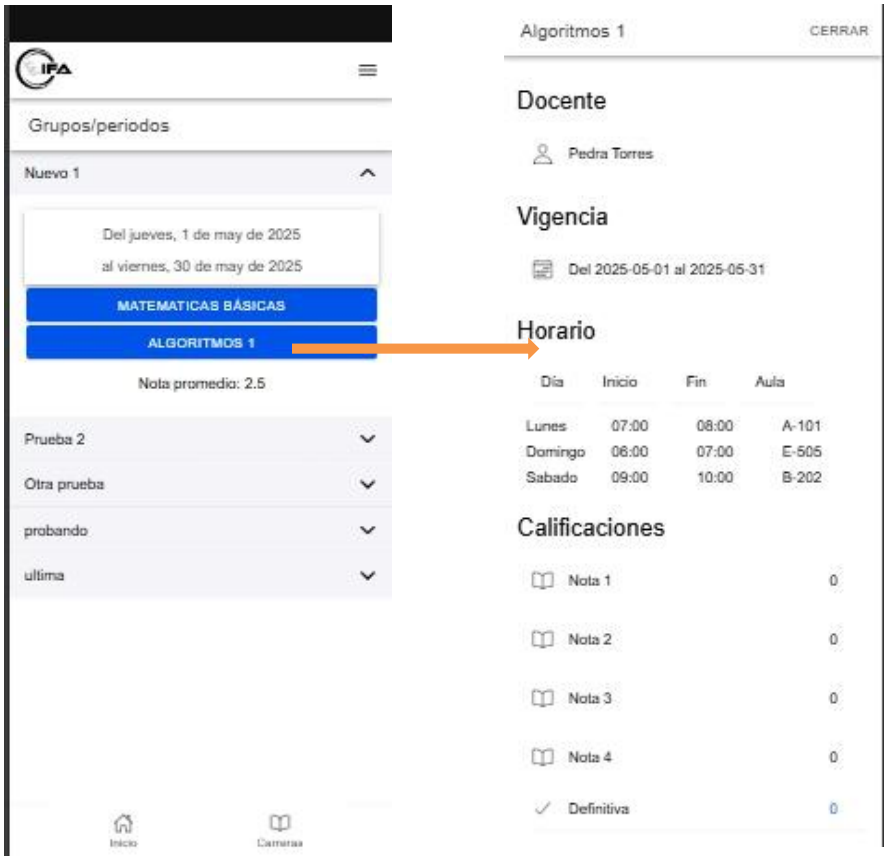
6.3. Consultar asignaturas:



7. Estudiante:

7.1. Menu:

7.2. Calificaciones:



7.3. Consultar agenda diaria:

8. Documentacion backend:

8.1. Rutas para obtención y creanción en el backend:

SpringSecurityConfig.java

```java
package com.santorres.academica.configuration;

import ...

@Configuration  sanTorres770 +2
public class SpringSecurityConfig {

    private final AuthenticationConfiguration authenticationConfiguration;  4 usages

    public SpringSecurityConfig(AuthenticationConfiguration authenticationConfiguration) {  sanTorres770
        this.authenticationConfiguration = authenticationConfiguration;
    }

    @Bean  sanTorres770
    PasswordEncoder passwordEncoder() { return new BCryptPasswordEncoder(); }

    @Bean  sanTorres770
    AuthenticationManager authenticationManager() throws Exception {
        return authenticationConfiguration.getAuthenticationManager();
    }

    @Bean  sanTorres770 +2
    SecurityFilterChain filterChain(HttpSecurity http) throws Exception{
        return http.authorizeHttpRequests( AuthorizationManagerRequestMat... authRules -> authRules
                .requestMatchers(HttpMethod.GET, "/api/users/{id}").permitAll()
                .requestMatchers(HttpMethod.GET, "/api/users").permitAll()
                .requestMatchers(HttpMethod.POST, "/api/users/verify").permitAll()
                .requestMatchers(HttpMethod.GET, "/api/students").permitAll()
                .requestMatchers(HttpMethod.GET, "/api/students/{id}").permitAll()
                .requestMatchers(HttpMethod.POST, "/api/students").permitAll()
                .requestMatchers(HttpMethod.POST, "/api/teachers").permitAll()
                .requestMatchers(HttpMethod.GET, "/api/teachers").permitAll()
```

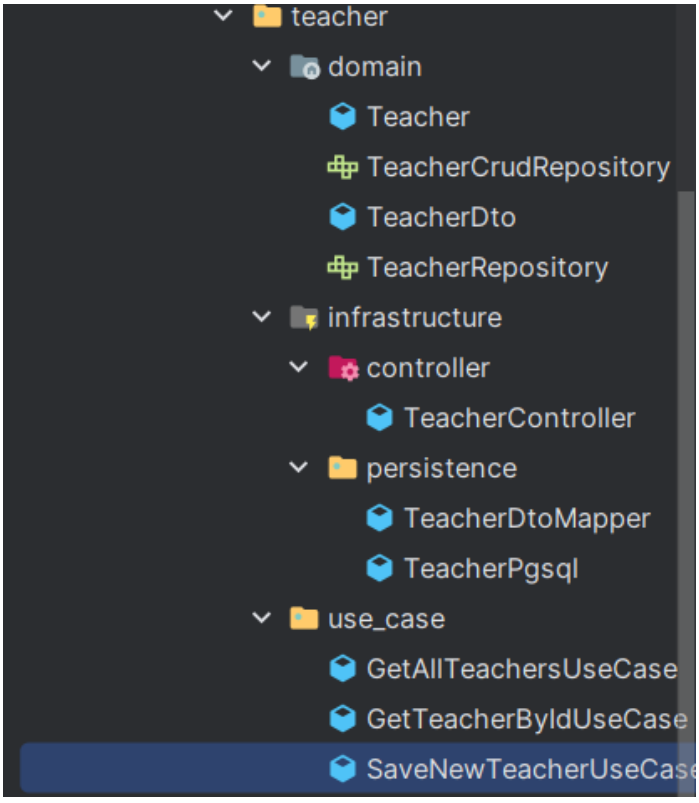
8.2. Rutas para obtención y creanción en el backend:

```java
                .requestMatchers(HttpMethod.GET, "/api/teachers").permitAll()
                .requestMatchers(HttpMethod.GET, "/api/teachers/{id}").permitAll()
                .requestMatchers(HttpMethod.POST, "/api/subjects").permitAll()
                .requestMatchers(HttpMethod.GET, "/api/courses").permitAll()
                .requestMatchers(HttpMethod.GET, "/api/periods").permitAll()
                .requestMatchers(HttpMethod.POST, "/api/periods").permitAll()
                .requestMatchers(HttpMethod.GET, "/api/periods/student/{studentId}").permitAll()
                .requestMatchers(HttpMethod.GET, "/api/periods/teacher/{studentId}").permitAll()
                .requestMatchers(HttpMethod.GET, "/api/periods/student/last/{studentId}").permitAll()
                .requestMatchers(HttpMethod.GET, "/api/periods/teacher/last/{studentId}").permitAll()
                .requestMatchers(HttpMethod.GET, "/api/classrooms").permitAll()
                .requestMatchers(HttpMethod.POST, "/api/classrooms").permitAll()
                .requestMatchers(HttpMethod.PUT, "/api/notes").permitAll()
                .requestMatchers(HttpMethod.GET, "/api/schedules/student/{studentId}").permitAll()
                .requestMatchers(HttpMethod.GET, "/api/schedules/teacher/{teacherId}").permitAll()
                .requestMatchers(HttpMethod.POST, "/confirmation/user/email").permitAll()
                .anyRequest().authenticated())
                .addFilter(new JwtAuthenticationFilter(authenticationConfiguration.getAuthenticationManager()))
                .addFilter(new JwtValidationFilter(authenticationConfiguration.getAuthenticationManager()))
                .csrf(AbstractHttpConfigurer::disable)
                .sessionManagement( SessionManagementConfigurer<HttpSecurity> management -> management.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
                .cors( CorsConfigurer<HttpSecurity> cors -> cors.configurationSource(corsConfigurationSource()))
                .build();
    }

    @Bean  sanTorres770
    CorsConfigurationSource corsConfigurationSource() {

        CorsConfiguration config = new CorsConfiguration();
```

8.3. Rutas para obtención y creanción en el backend

```
    @Bean  sanTorres770
    CorsConfigurationSource corsConfigurationSource() {

        CorsConfiguration config = new CorsConfiguration();
        config.setAllowedOrigins(Arrays.asList("http://localhost:8100"));
        config.setAllowedOriginPatterns(Arrays.asList("*"));
        config.setAllowedMethods(Arrays.asList("GET", "POST", "PUT", "DELETE"));
        config.setAllowedHeaders(Arrays.asList("Authorization", "Content-Type", "X-Requested-With"));
        config.setAllowCredentials(true);

        UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
        source.registerCorsConfiguration( pattern: "/**", config);
        return source;
    }

    @Bean  sanTorres770
    FilterRegistrationBean<CorsFilter> corsFilter() {
        FilterRegistrationBean<CorsFilter> bean = new FilterRegistrationBean<>(
                new CorsFilter(corsConfigurationSource()));
        bean.setOrder(Ordered.HIGHEST_PRECEDENCE);
        return bean;
    }

}
```

9. Estructura de carpetas para creacion de entidades:

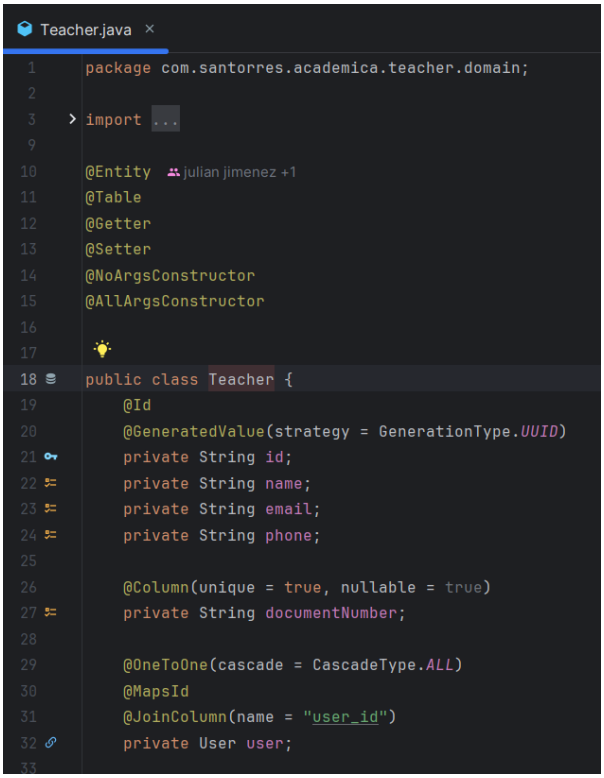


Aquí podemos evidenciar las estructuras de las carpetas para recibir las peticiones solicitadas de parte del front y que llegan mediante las rutas enseñadas anteriormente como podemos evidenciar en el use_case el cuenta con un GetAlllTeachersUseCase que sirve para obtener todos los registros de la tabla teacher. Seguido esta el GetTeacherByIdUseCase que este lo que realiza o lo que nos ayuda a gestionar es la obtención de un teacher por medio de su id. Como tercero tenemos el SaveNewTeacherUseCase que por medio de este recibimos la petición post enviada desde el front con los campos requeridos para poder realizar la creación de un teacher de forma exitosa.

9.1. Detalles de los archivos vistos en la carpeta

- Debemos tener en cuenta que antes de realizar los métodos mencionados anteriormente debemos realizar una seria de pasos que nos permiten la creación y obtención de teacher de una manera exitosa, con lo primero que empezamos es con la creación de la tabla teacher en la base de datos, esa estructura de la tabla de la base de datos la encontramos en el domain, que es allí donde le damos la estructura y las columnas que tendrá esa tabla de la base de datos

9.1.1.Clase Teacher.java:

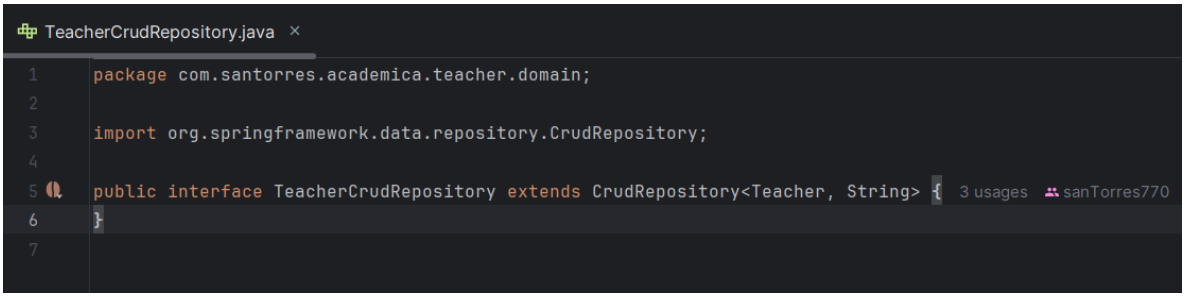


9.1.2.TeacherCrudRepository:

- Con el TeacherCrudRepository lo que se hace es extender CrudRepository para permitir las operaciones CRUD básicas, tales como lo son:

  save(), findAll, findById(), delete.

  Gracias al Spring Data JPA se genera la implementación automáticamente.



9.1.3. TeacherDto:

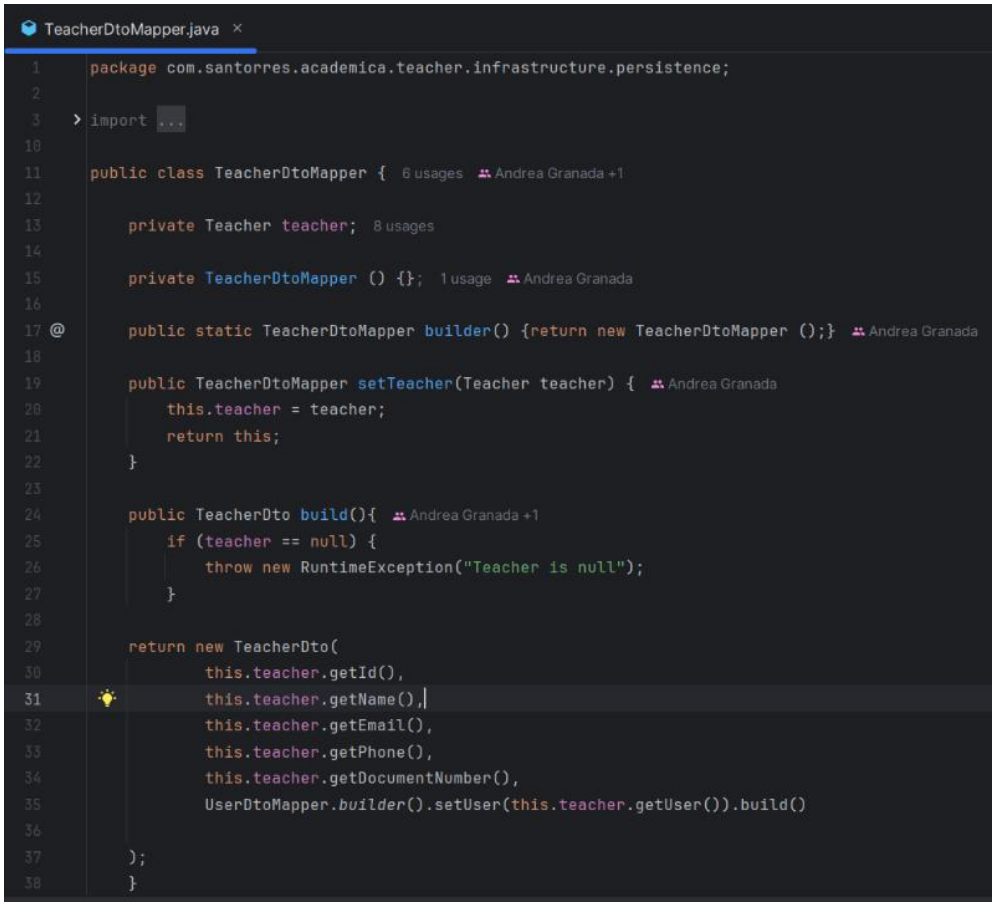
- Con TeacherDto lo que hacemos es enviar o recibir la información necesaria del front y gracias a este no se expone la entidad completa ni las relaciones innecesarias

TeacherDto.java

```java
package com.santorres.academica.teacher.domain;

import ...

@Getter  23 usages  Andrea Granada +1
@Setter
@NoArgsConstructor
@AllArgsConstructor

public class TeacherDto {
    private String id;
    private String name;
    private String email;
    private String phone;
    private String documentNumber;
    private UserDto user;
}
```

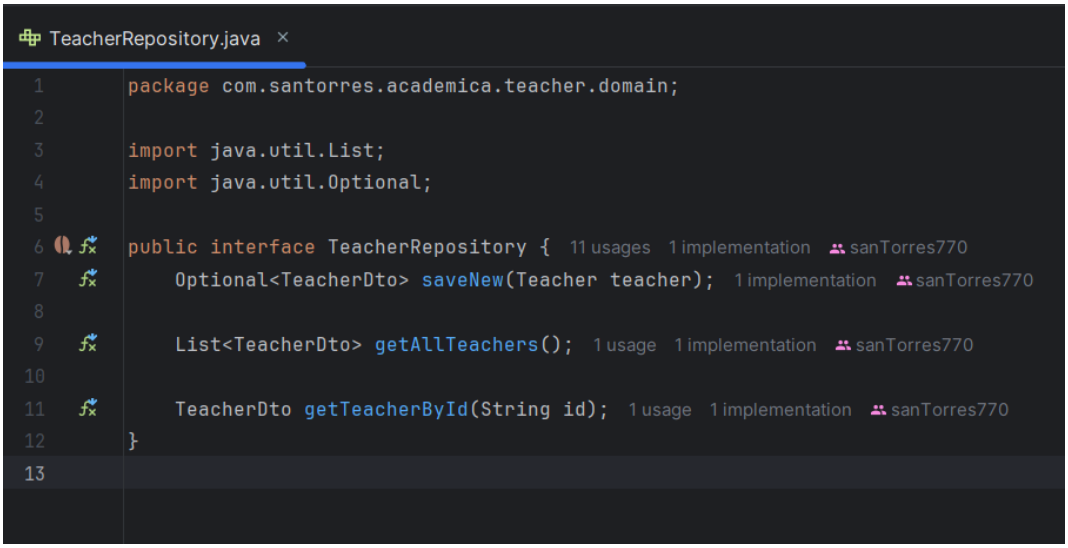
9.1.4. TeacherDtoMapper:

- El TeacherDtoMapper nos ayuda a convertir el Teacher que es una propiedad JPA a un TeacherDto y también convierte el User asignado al Teacher en un UserDto.



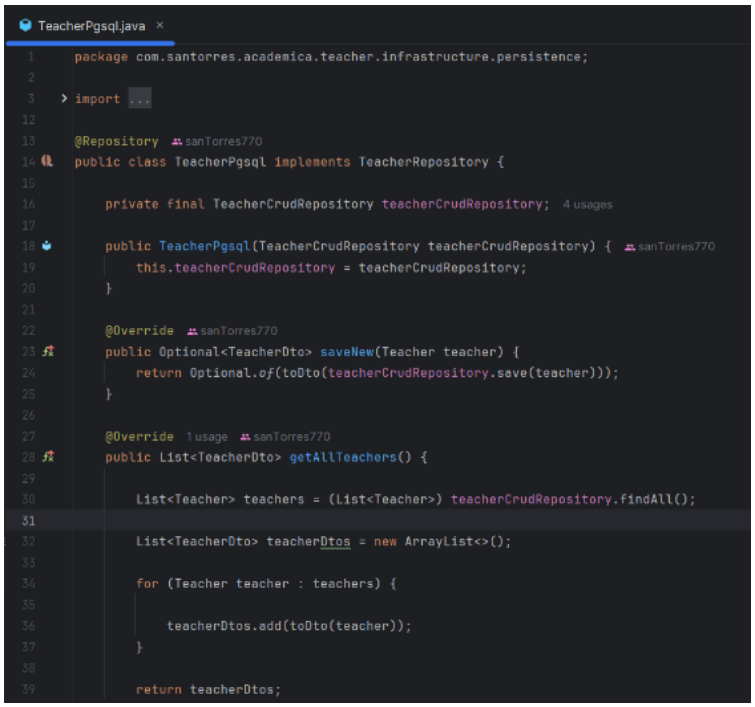
9.1.5. TeacherRepository

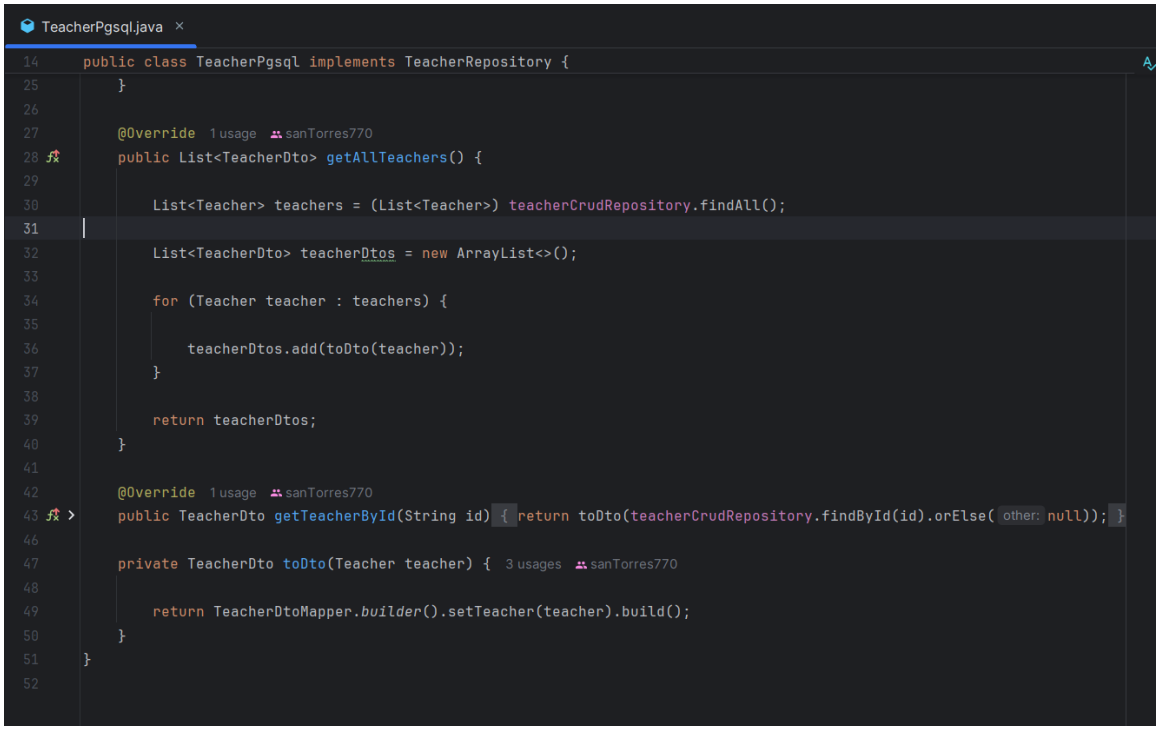
- El TeacherRepository define la interfaz que la infraestructura debe implementar. Se utiliza en los casos de uso o use:case sin saber que base de datos o tecnología hay detrás.



9.1.6. TeacherPgsql:

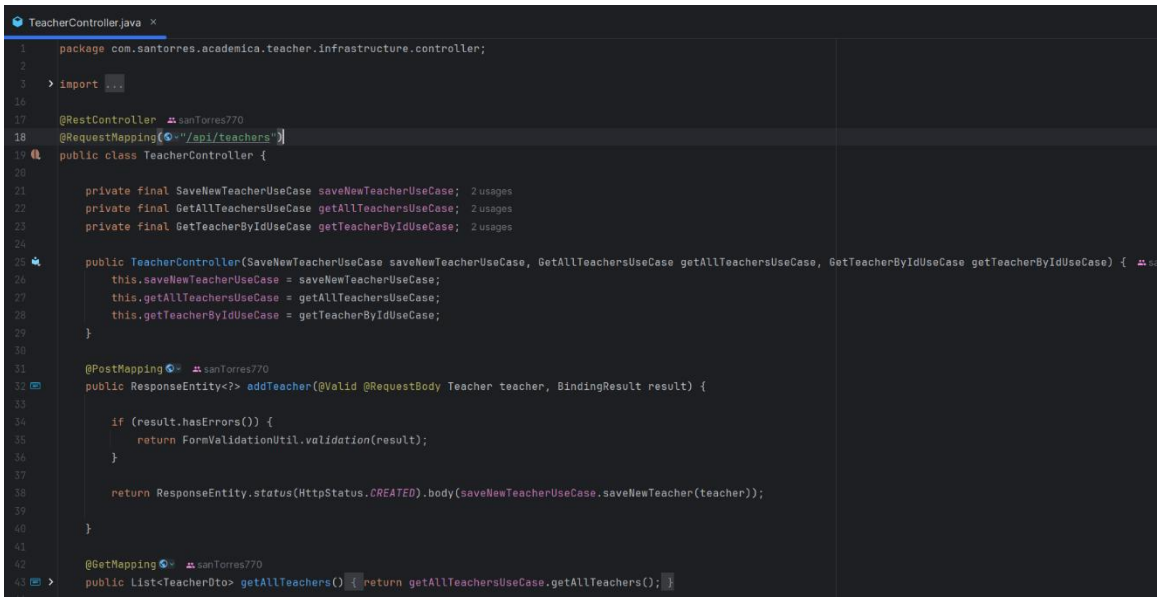
- En el TeacherPgsql se implementa el TeacherRepository utilizando el TeacherCrudRepository. Se realizan las consultas a la base de datos y se utiliza el TeacherDtoMapper para convertir los resultados a TeacherDto

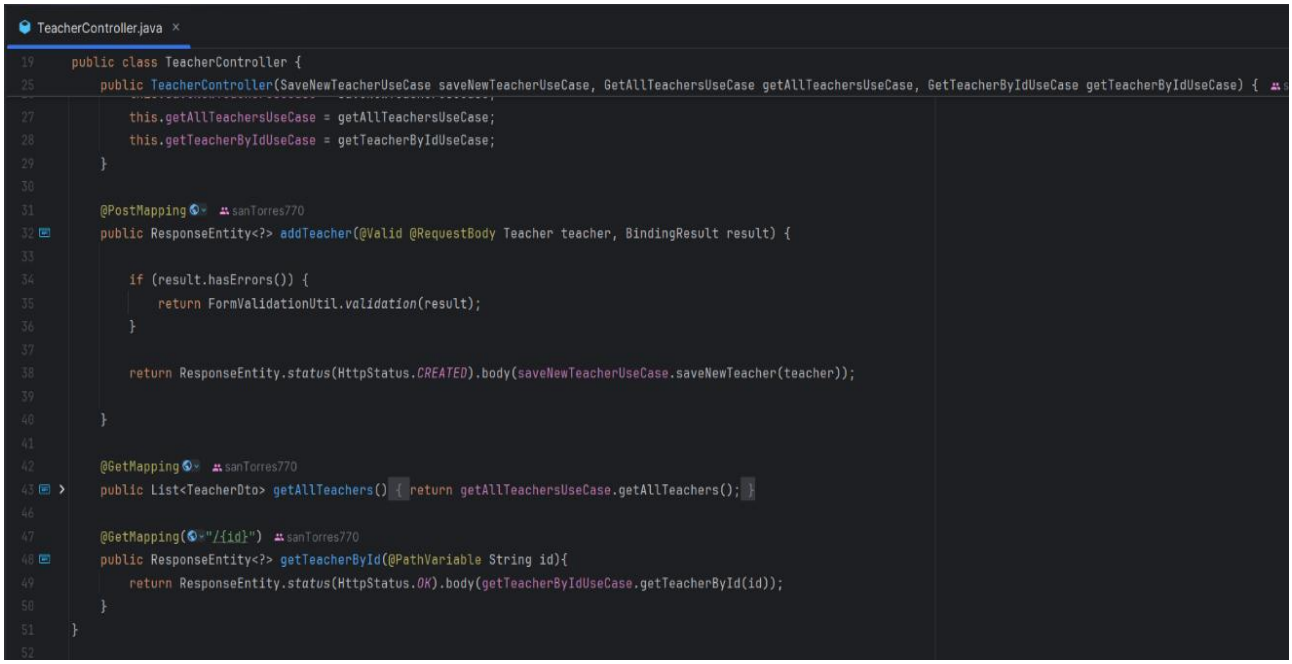




9.1.7. TeacherController:

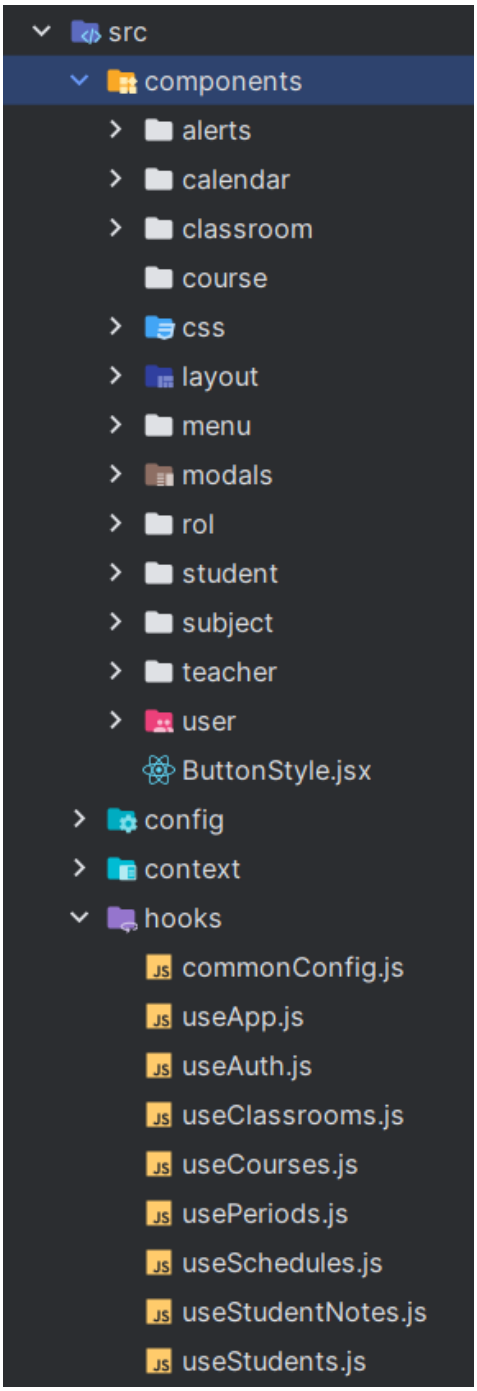
- En el TeacherController lo que se hace es recibir las peticiones HTTP y devolver respuestas JSON. En este también se llaman los casos de uso y se exponen los endpoints REST en este caso “/api/teachers”

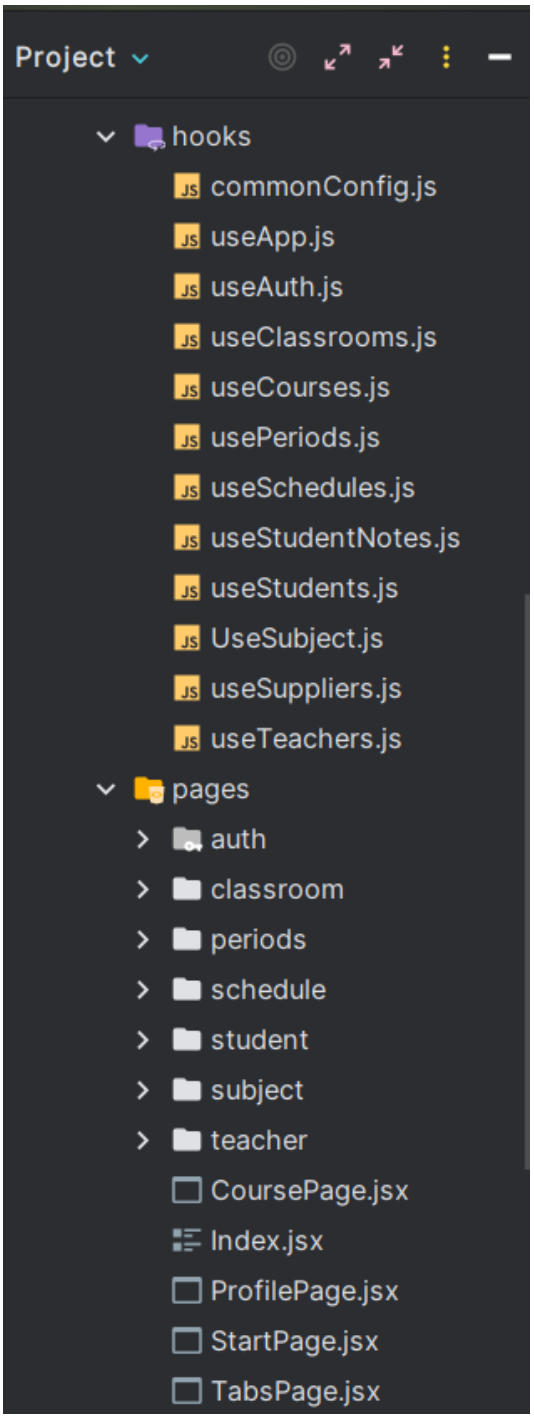




10. Patrones de Diseño y Componentes Clave:

10.1. Estructura de carpetas Front:





10.2. Componentes para la creación de un nuevo estudiante:

En el NewStudentForm.jsx lo que este permite es crear un nuevo estudiante, las imágenes correspondientes a este son las 4 que aparecen después de este texto. Podemos evidenciar como importa el formulario base reutilizable llamado StudentForm que lo trae desde components. También en la misma imagen podemos ver como se hace uso de los Refs, que son los que utilizamos para obtener acceso a los valores de los campos del formulario. Allí también evidenciamos la obtención de funciones del contexto UseApp, donde tenemos para el manejo de errores de validación (validationErrors, setValidationErrors) y la función que nos permite enviar los datos del formulario al backend (saveNewStudent). En la misma imagen tenemos los estados locale, en los cuales tenemos el setErrores y errores que nos permite guardar errores específicos de validación, también tenemos a setIsLoading e isLoading que nos indica si está esperando una respuesta del servidor.

En la misma imagen también vemos la función de handleRestartValue que lo que hacemos en esta es limpiar los datos después de la creación de un estudiante y tambien vemos el incio de la función handleSubmit lo que realizamos en esta función es recoger los datos de los Refs y construimos un objeto llamado studentForm con los campos name, email, phone, documentNumber, course y el rol, que este último lo asignamos por defecto. En esta misma función también llamamos al saveNewStudent para enviar los datos al backend, en esta función tenemos un toast de éxito o error según la respuesta del

backend. Allí también manejamos lo que son errores de validación en casos de error 400, manejo de error de red en el caso de no haber conexión.

Podemos evidenciar como se empieza a renderizar el formulario StudentForm, pasando el onSubmit que es la función de enviar, courseRef para el selector del curso, los inputs que contiene un arreglo con los datos de cada campo y el nameButton que contiene el texto que se le asigna al botón dependiendo del formulario, en este caso el de creación de estudiante. Allí ya terminaríamos con el NewStudenForm.

Seguido a esas 4 imágenes que describimos anteriormente, las 2 siguientes a estás corresponden a StudentForm, el cual es una plantilla reutilizable de un formulario para crear estudiantes, el cual cuenta con unos componentes de Ionic React (IonContent, IonItem, IonList, IonPage,

IonSelect, IonSelectOption), también tiene un InputForm que también es una plantilla reutilizable para los campos de tipo input en el formulario.

Este componente espera recibir inputs, nameButton, onSubmit, courseRef, que estamos referenciando en el NewStudentForm. En el StudentForm manejamos la estructura del formulario, renderizamos dinámicamente los inputs utilizando el componente InputForm, donde cada campo está definido desde el NewStudentForm, que es el componente padre.

En el StudentForm tenemos un IonSelect que es la lista desplegable para seleccionar el curso para el estudiante y cuando se selecciona queda guardado su Id en courseRef.current.

Elemento	NewStudentForm	StudentForm
¿Qué hace?	Lógica de negocio (manejo de datos, errores, envío)	Presentación visual del formulario
Define los inputs	Sí (con useRef, array de objetos)	No (solo los recibe y los dibuja)
Envía formulario	Define handleSubmit()	Lo ejecuta al hacer onSubmit={...}
Maneja courseRef	Define el useRef y lo pasa como prop	Actualiza su valor al seleccionar curso
Función saveNewStudent	La ejecuta desde useApp()	No tiene lógica de backend
Campos visuales (InputForm)	No los renderiza	Sí, los genera con inputs.map(...)
Select de cursos	Pasa courseRef y obtiene cursos desde contexto	Lo muestra con IonSelect

10.2.1. Componentes para la creación de un nuevo estudiante NewStudentForm:

NewStudentForm.jsx

```jsx
import StudentForm from "../../components/student/StudentForm.jsx";
import useApp from "../../hooks/useApp.js";
import { useRef, useState } from "react";
import { toast } from "react-toastify";
import LoadingAlert from "../../components/alerts/LoadingAlert.jsx";

export default function NewStudentForm() { Show usages CamiRinconR +1
    const nameRef :RefObject<null> = useRef( initialValue: null);
    const emailRef :RefObject<null> = useRef( initialValue: null);
    const phoneRef :RefObject<null> = useRef( initialValue: null);
    const courseRef :RefObject<string> = useRef( initialValue: '');
    const documentRef :RefObject<null> = useRef( initialValue: null);

    const {validationErrors,
        setValidationErrors,
        saveNewStudent} = useApp()

    const [errores, setErrores] = useState( initialState: []);
    const [isLoading :boolean , setIsLoading] = useState( initialState: false);


    const handleRestartValue = () :void => { Show usages CamiRinconR +1
        nameRef.current.value = '';
        phoneRef.current.value = '';
        emailRef.current.value = '';
        courseRef.current = '';
        documentRef.current.value = '';
    };

    const handleSubmit = (e) :void => { Show usages CamiRinconR +1
        e.preventDefault();
        setErrores( value: []);
```

10.2.2. Componentes para la creación de un nuevo estudiante NewStudentForm:

NewStudentForm.jsx

```
7   export default function NewStudentForm() {  Show usages  CamiRinconR +1
30      const handleSubmit = (e) : void  => {  Show usages  CamiRinconR +1
33          setIsLoading( value: true);
34
35          const studentForm :{...}  = {
36              name: nameRef.current.value,
37              email: emailRef.current.value,
38              phone: phoneRef.current.value,
39              user: {
40                  roles: [{
41                      id: "3"
42                  }]
43              },
44              course: {
45                  id: courseRef.current || ""
46              },
47              documentNumber: documentRef.current.value
48          };
49
50          saveNewStudent(studentForm).then(data => {
51
52              if (data.status === 200 || data.status === 201) {
53                  toast.success( content: `Se creó el estudiante ${data.data.name} correctamente`);
54                  handleRestartValue();
55              } else {
56                  toast.error( content: `Error en el proceso.`);
57              }
58          }).catch(error => {
59
60
61              if (error.code === "ERR_BAD_REQUEST") {
62                  setErrores(Object.values(error.response.data))
```

10.2.3. Componentes para la creación de un nuevo estudiante NewStudentForm:

NewStudentForm.jsx

```
7   export default function NewStudentForm() {  Show usages  CamiRinconR +1
30      const handleSubmit = (e) : void  => {  Show usages  CamiRinconR +1
58          }).catch(error => {
63                  setValidationErrors(error.response.data)
64                  toast.error( content: 'Revisa los campos que faltan por diligenciar en el formulario.')
65              }
66
67              if (error.code === "ERR_NETWORK") {
68                  toast.error( content: 'Error de conexión.')
69              }
70
71
72          }).finally( onFinally: () : void  => setIsLoading( value: false))
73
74      }
75      if (isLoading){
76          return <LoadingAlert/>
77      }
78
79
80      return (
81          <StudentForm
82              onSubmit={handleSubmit}
83              courseRef={courseRef}
84              inputs={[
85                  {
86                      id: '1',
87                      inputId: 'name',
88                      type: 'text',
89                      reference: nameRef,
90                      placeholder: 'Nombre completo',
91                      labelValue: 'Nombre completo',
```

10.2.4. Componentes para la creación de un nuevo estudiante NewStudentForm:

NewStudentForm.jsx

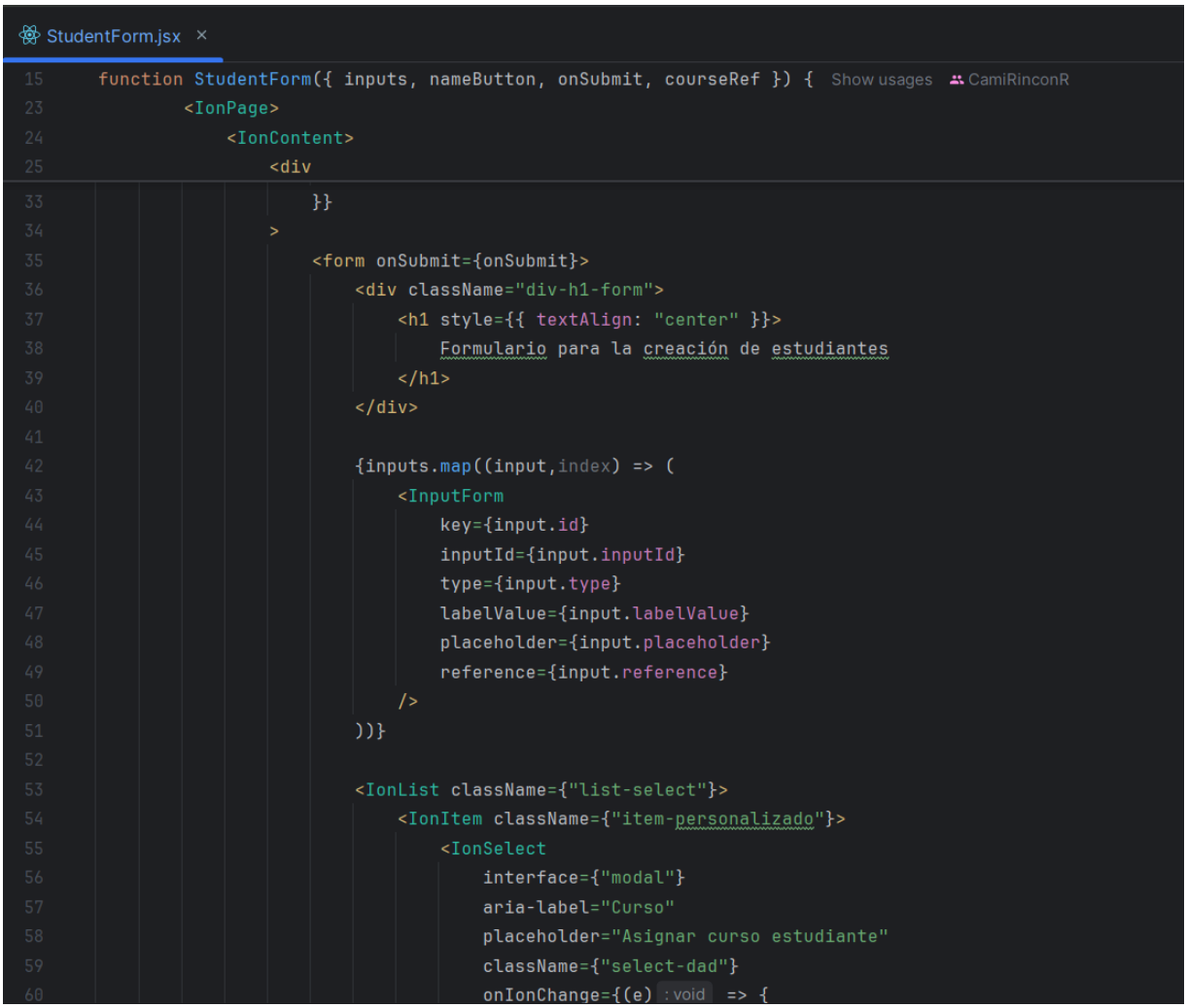
```
export default function NewStudentForm() {  Show usages  CamiRinconR +1
        <StudentForm
                },
                {
                    id: '2',
                    inputId: 'documentNumber',
                    type: 'text',
                    reference: documentRef,
                    placeholder: 'N° Documento',
                    labelValue: 'Número de documento',
                },
                {
                    id: '3',
                    inputId: 'email',
                    type: 'email',
                    reference: emailRef,
                    placeholder: 'alguien@alguien.com',
                    labelValue: 'Correo electrónico',
                },
                {
                    id: '4',
                    inputId: 'phone',
                    type: 'tel',
                    reference: phoneRef,
                    placeholder: '3105689654',
                    labelValue: 'Número de celular',
                },

            ]}
            nameButton={'Crear estudiante'}
        />
    );
```

10.2.5. Componentes para la creación de un nuevo estudiante StudentForm:

StudentForm.jsx

```jsx
import InputForm from "../layout/InputForm.jsx";
import ButtonStyle from "../ButtonStyle.jsx";
import {
    IonContent,
    IonItem,
    IonList,
    IonPage,
    IonSelect,
    IonSelectOption,
} from "@ionic/react";
import "../css/generalStyle.css";
import useApp from "../../hooks/useApp.js";
import { useEffect } from "react";

function StudentForm({ inputs, nameButton, onSubmit, courseRef }) {  Show usages  CamiRinconR
    const { allCourses, getAllCourses } = useApp();

    useEffect( effect: () : void => {
        getAllCourses();
    }, deps: []);

    return (
        <IonPage>
            <IonContent>
                <div
                    className={"div-form"}
                    style={{
                        display: "flex",
                        justifyContent: "center",
                        alignItems: "center",
                        marginLeft: "20px",
                        marginRight: "20px",
```

10.2.6. Componentes para la creación de un nuevo estudiante StudentForm:



StudentForm.jsx

```
15  function StudentForm({ inputs, nameButton, onSubmit, courseRef }) {  Show usages  CamiRinconR
23          <IonPage>
24              <IonContent>
25                  <div
33                      }}
34                  >
35                      <form onSubmit={onSubmit}>
36                          <div className="div-h1-form">
37                              <h1 style={{ textAlign: "center" }}>
38                                  Formulario para la creación de estudiantes
39                              </h1>
40                          </div>
41
42                          {inputs.map((input,index) => (
43                              <InputForm
44                                  key={input.id}
45                                  inputId={input.inputId}
46                                  type={input.type}
47                                  labelValue={input.labelValue}
48                                  placeholder={input.placeholder}
49                                  reference={input.reference}
50                              />
51                          ))}
52
53                          <IonList className={"list-select"}>
54                              <IonItem className={"item-personalizado"}>
55                                  <IonSelect
56                                      interface={"modal"}
57                                      aria-label="Curso"
58                                      placeholder="Asignar curso estudiante"
59                                      className={"select-dad"}
60                                      onIonChange={(e) : void  => {
```

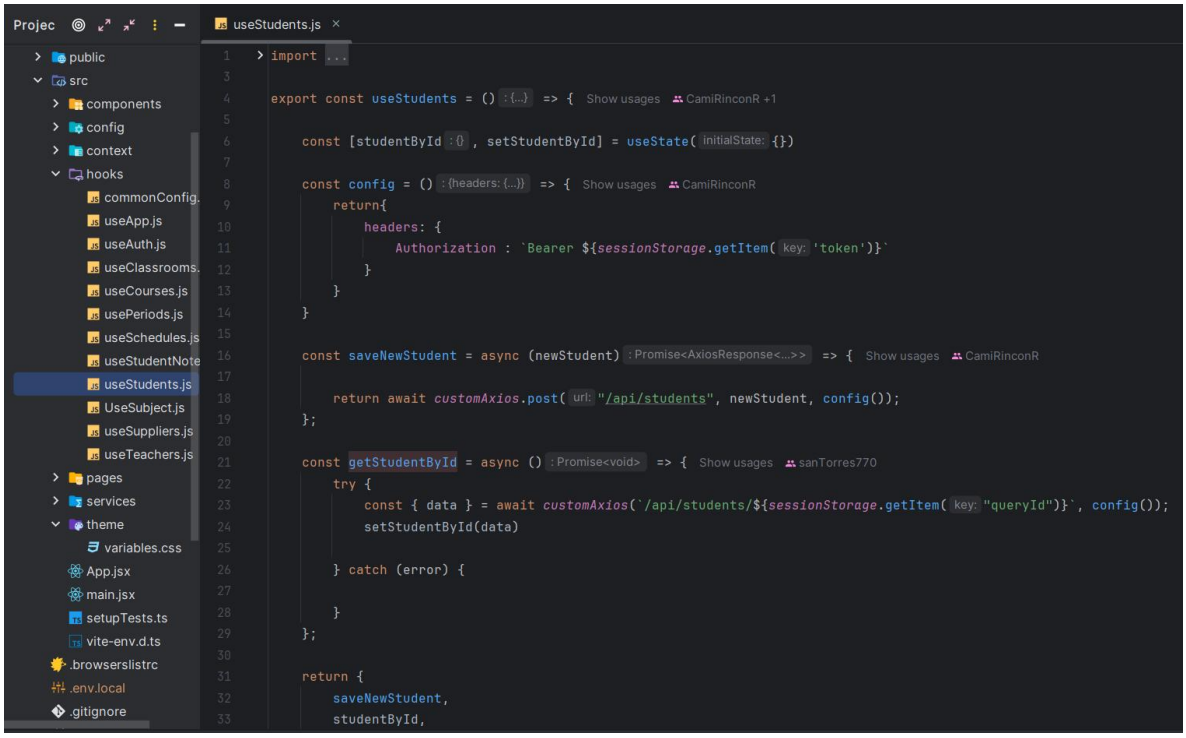
10.2.7. Componentes para la creación de un nuevo estudiante StudentForm:

StudentForm.jsx

```
15    function StudentForm({ inputs, nameButton, onSubmit, courseRef }) {  Show usages  CamiRinconR
23            <IonPage>
24                <IonContent>
25                    <div
35                        <form onSubmit={onSubmit}>
61                                                courseRef.current = e.detail.value;
62                                            }}
63                                        >
64                                            {allCourses.map((course) => (
65                                                <IonSelectOption key={course.id} value={course.id}>
66                                                    {course.name}
67                                                </IonSelectOption>
68                                            ))}
69                                        </IonSelect>
70                                    </IonItem>
71                                </IonList>
72
73                                <ButtonStyle name={nameButton} />
74                            </form>
75                    </div>
76                </IonContent>
77            </IonPage>
78        );
79    }
80
81    export default StudentForm;  Show usages  CamiRinconR
82
```

De esta forma se maneja para todos los componentes.

11. Hooks de estudiante en el front, para enviar o recibir peticiones del back:



El useStudent es el hook que encapsula la lógica relacionada con elll estudiante y dentro de esta tenemos el studentById, setStudentById que es donde se almacenará los datos de un estudiante individualmente.

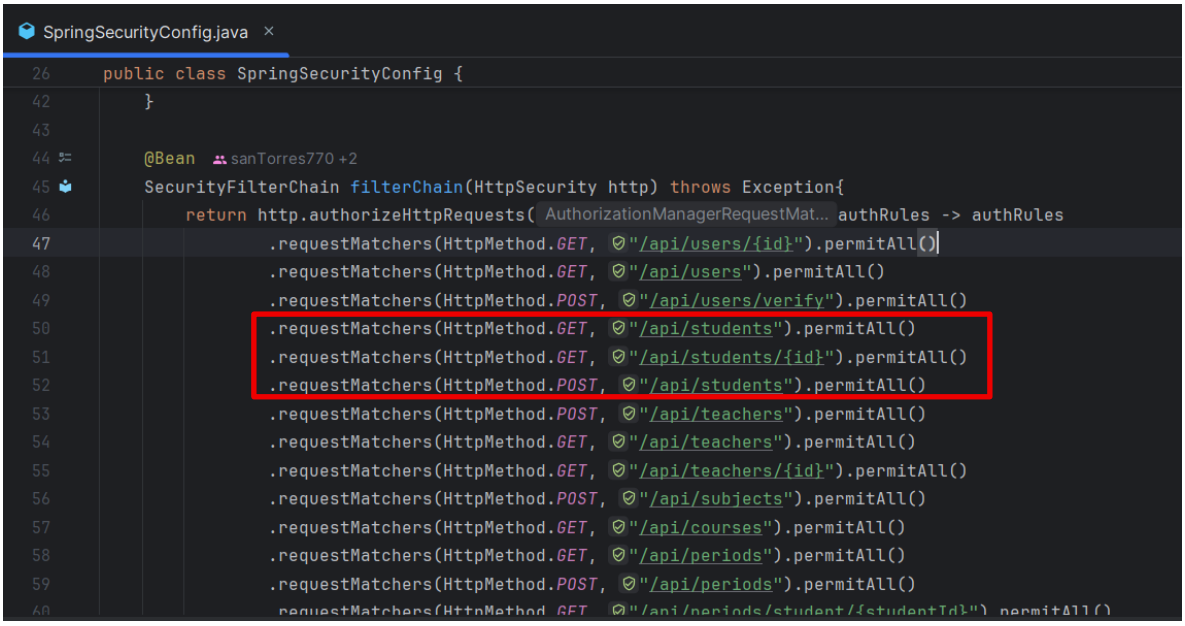
Continuando con la estructura de:

saveNewStudent que con esta lo que se logra es enviar una solicitud post al backend para crear un nuevo estudiante.

Allí ya pasamos a la estructura de:

getStudentById que por medio de esta realizamos una petición get al backend que recupera un estudiante especifico. Y continuamos con lo que expone este hook que es el saveNewStudent -> Para guardar un nuevo estudiante, studentById -> Obtener un estudiante por Id, getStudentById -> Estudiante actualmente guardado en la base de datos.

11.1. Recibe la petición del Student del Front.

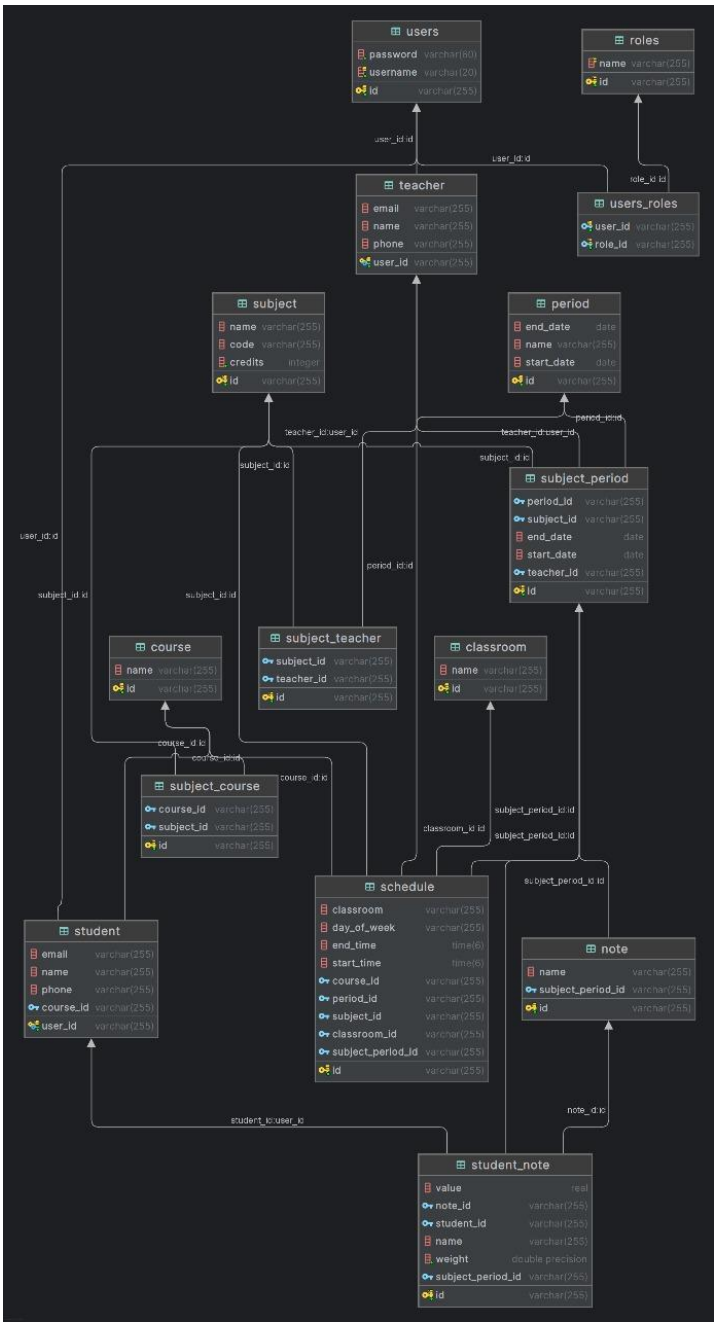


De esta manera se manejan todos los componentes.

12. Base de Datos

Para la implementación de la base de datos utilizamos el motor de PostgreSQL.

12.1. Modelo de la base de datos



13. Tabla de herramientas

Categoría	Herramienta / Tecnología	Descripción
Frontend (UI)	ReactJS	Biblioteca de JavaScript para construir interfaces de usuario.
	Ionic Framework	Framework para apps híbridas con componentes móviles y soporte multiplataforma.
	Capacitor	Puente nativo para acceder a funcionalidades del dispositivo.
Backend (API)	Java Spring Boot	Framework para construir microservicios y APIs REST robustas.
	Maven / Gradle	Herramientas de construcción y gestión de dependencias para Java.
Base de Datos	PostgreSQL	Sistema de gestión de bases de datos relacional, potente y de código abierto.
	pgAdmin	Interfaz gráfica para administrar bases de datos PostgreSQL.
Autenticación / Seguridad	Spring Security	Módulo de seguridad para autenticación y autorización en Spring Boot.
	JWT (JSON Web Tokens)	Estándar para autenticación basada en tokens.
Comunicación API	REST	Protocolos para la comunicación entre frontend y backend.
Desarrollo y Testing	Postman	Herramienta para probar APIs REST.
	IntellijIdea / Webstorm	Editores de código y entornos de desarrollo
Control de versiones	GitHub	Control de versiones y colaboración en el código fuente.

14. Conclusiones

La culminación de la aplicación "Gestión de Información Académica" no es solo la entrega de una herramienta funcional, sino, y quizás más importante, la validación empírica y significativa de los conocimientos adquiridos en el seminario de aplicaciones móviles. Poner en práctica la teoría en un proyecto real y complejo como este permitió extraer conclusiones esenciales sobre la efectividad de las metodologías y tecnologías estudiadas:

- Dominio del Desarrollo Híbrido: La implementación exitosa del frontend con Ionic Framework y ReactJS demostró la eficiencia superior del desarrollo híbrido. La capacidad de construir una única base de código para Android, tal como se enfatizó en el seminario, se tradujo directamente en una optimización drástica de tiempos y recursos, confirmando este enfoque como una solución pragmática y potente para la distribución móvil.
- Consolidación de Habilidades en ReactJS: La sintaxis, la arquitectura basada en componentes y el manejo de estado con React Hooks y Context API, pilares del seminario, se afianzaron completamente en la práctica. La construcción de interfaces de usuario interactivas para diferentes roles y la gestión dinámica de los datos del backend solidificaron una comprensión profunda de ReactJS, más allá de lo teórico.
- Comprensión Integral de la Arquitectura Cliente-Servidor: El seminario proporcionó las bases para entender la comunicación entre frontend y backend. La experiencia de integrar la aplicación móvil con una API RESTful desarrollada en Spring Boot permitió comprender de forma tangible cómo se orquestan las peticiones, se manejan las respuestas y se asegura la comunicación, cerrando el ciclo de la arquitectura distribuida.
- Relevancia de la Seguridad en el Desarrollo Móvil: La implementación de la autenticación con JWT y Spring Security resaltó la importancia crítica de la seguridad, un tema que, aunque a veces subestimado en la teoría inicial, se manifestó como una necesidad imperativa en un

entorno real. La práctica reforzó la necesidad de proteger la información del usuario y los recursos del sistema de accesos no autorizados.

- Impacto de las Metodologías Ágiles: Aunque el seminario se centró en aspectos técnicos, la aplicación de una metodología Ágil (Scrum) fue un aprendizaje transversal. La iteración constante, la adaptabilidad a los requisitos cambiantes y la colaboración facilitaron la gestión de un proyecto complejo, validando la eficacia de estos marcos de trabajo para el desarrollo de software moderno.

En definitiva, este proyecto no fue solo un ejercicio técnico, sino una experiencia de aprendizaje transformadora que demostró la viabilidad, la eficiencia y el poder de las herramientas y conceptos vistos en el seminario. La teoría se convirtió en una guía sólida para resolver problemas del mundo real, equipándonos con las habilidades necesarias para enfrentar futuros desafíos en el desarrollo de aplicaciones móviles.

15. Referencias

https://platzi.com/blog/arquitectura-hexagonal/

https://aws.amazon.com/es/what-is/restful-api/

https://ionicframework.com/docs/

https://spring.io/projects/spring-framework

https://administraciondesistemas.com/arquitectura-tecnologica-netflix/