

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.....	4
2.	Palabras clave.....	5
3.	Marco conceptual y contextual	6
3.1.	Marco Conceptual y Contextual del Backend	6
3.1.1.	Conceptos y Definiciones Requeridas:	6
3.1.2.	Spring Boot (Framework de Java):	7
3.1.3.	Bases de Datos Relacionales (PostgreSQL):	7
3.1.4.	Seguridad en Aplicaciones (Spring Security y JWT):	8
3.1.5.	Arquitectura Hexagonal:	8
3.1.6.	Contexto del Trabajo del Backend:.....	9
3.2.	Marco Conceptual y Contextual del Frontend	9
3.2.1.	Ionic Framework:.....	10
3.2.2.	ReactJS (Librería de JavaScript):.....	10
3.2.3.	Manejo de Estado (React Hooks y Context API):	11
3.2.4.	Consumo de APIs (Cliente-Side):.....	11
3.2.5.	Contexto del Trabajo del Frontend:	12
4.	Desarrollo e implementación del aprendizaje.....	13
4.1.	La arquitectura de la aplicación móvil.....	13
4.2.	Visión General y Arquitectura	14
4.2.1.	Home:.....	15
4.2.2.	Home carreras	16
4.2.3.	Login:.....	17
4.2.4.	Manejo de rutas según su rol:	17
4.2.5.	Rutas según el rol.....	18
4.2.6.	Rutas según el rol.....	19
4.2.7.	Rutas según el rol y boton de Logout	20
4.2.8.	Verificación de autenticación y muestra de rutas según el rol	21
4.2.9.	Verificación de autenticación y muestra de rutas según el rol	21
4.2.10.	Rutas	22
4.2.11.	Rutas y fin de estructura	22
5.	Administrador:	23
5.1.	Menu:	23
5.2.	Registro de grupos:	24
5.3.	Registro de aulas:	24
5.4.	Registro de materias:.....	25
5.5.	Registro de estudiantes:	25
5.6.	Registro de docentes:	26
6.	Docente:	26
6.1.	Menu:	26
6.2.	Consultar agenda diaria:	27

	3
6.3. Consultar asignaturas:.....	28
7. Estudiante:.....	29
7.1. Menu:	29
7.2. Calificaciones:.....	30
7.3. Consultar agenda diaria:	30
8. Documentacion backend:.....	31
8.1. Rutas para obtención y creación en el backend:	31
8.2. Rutas para obtención y creación en el backend:	31
8.3. Rutas para obtención y creación en el backend	32
9. Estructura de carpetas para creacion de entidades:.....	32
9.1. Detalles de los archivos vistos en la carpeta	33
9.1.1. Clase Teacher.java:.....	34
9.1.2. TeacherCrudRepository:.....	34
9.1.3. TeacherDto:.....	35
9.1.4. TeacherDtoMapper:	36
9.1.5. TeacherRepository	37
9.1.6. TeacherPgsq!:.....	38
9.1.7. TeacherController:	39
10. Patrones de Diseño y Componentes Clave:	40
10.1. Estructura de carpetas Front:	40
10.2. Componentes para la creación de un nuevo estudiante:	41
10.2.1. Componentes para la creación de un nuevo estudiante NewStudentForm:	44
10.2.2. Componentes para la creación de un nuevo estudiante NewStudentForm:	45
10.2.3. Componentes para la creación de un nuevo estudiante NewStudentForm:	46
10.2.4. Componentes para la creación de un nuevo estudiante NewStudentForm:	47
10.2.5. Componentes para la creación de un nuevo estudiante StudentForm:	48
10.2.6. Componentes para la creación de un nuevo estudiante StudentForm:	49
10.2.7. Componentes para la creación de un nuevo estudiante StudentForm:	50
11. Hooks de estudiante en el front, para enviar o recibir peticiones del back:.....	51
11.1. Recibe la petición del Student del Front.	52
12. Base de Datos.....	53
12.1. Modelo de la base de datos	53
13. Tabla de herramientas	54
14. Conclusiones	55
15. Referencias.....	57

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 testeado 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



Nuestros cursos



Ingeniería de sistemas

Formación integral en programación, bases de datos, y sistemas de información.



Ingeniería Química

Estudio avanzado de procesos químicos y producción industrial.

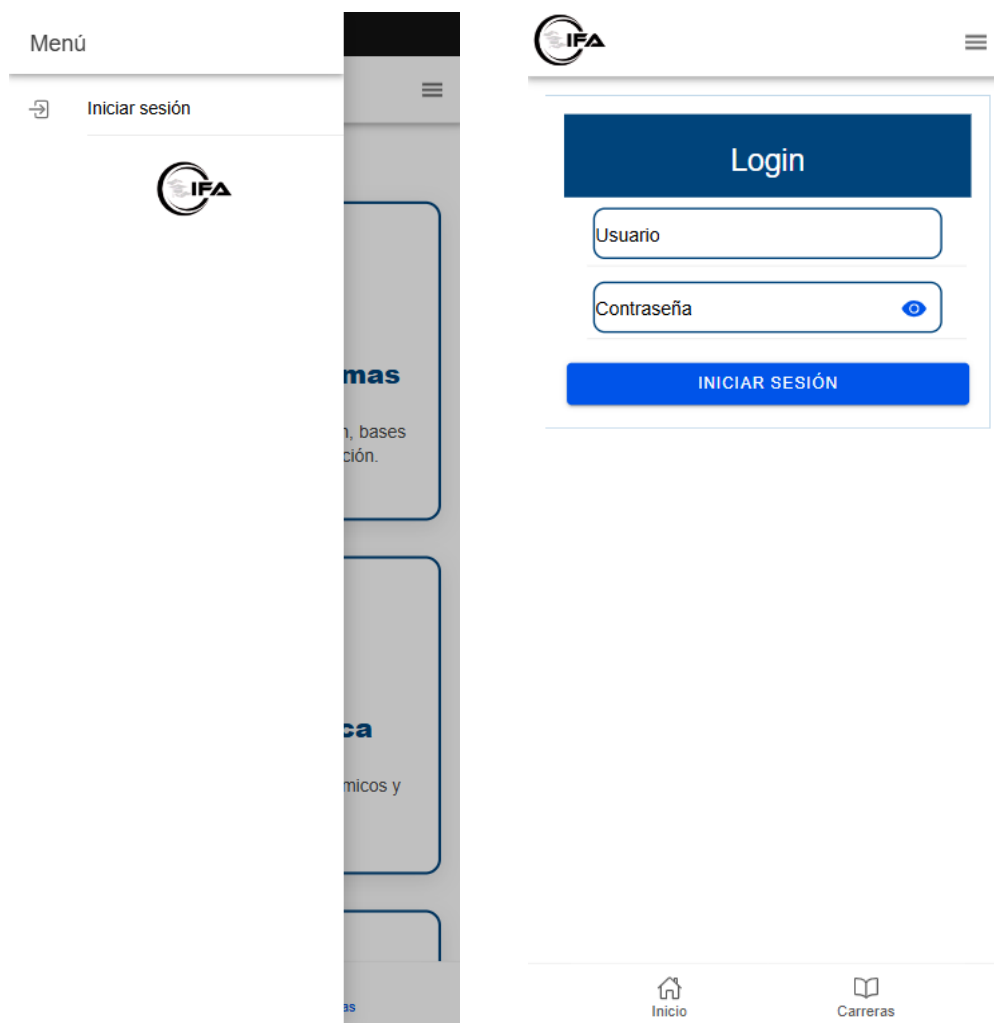


Inicio



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 ×
52  const routes : [{path: string, role: string[]...} = [
79      label: 'Asignaturas',
80    },
81    {
82      id: 4,
83      icon: bookOutline,
84      path: '/create/periods',
85      role: ['ROLE_ADMIN'],
86      label: 'Asignar materias',
87    },
88    {
89      id: 5,
90      icon: personAdd,
91      path: '/create/students',
92      role: ['ROLE_ADMIN'],
93      label: 'Crear estudiantes',
94    },
95    {
96      id: 6,
97      icon: personAdd,
98      path: '/create/teachers',
99      role: ['ROLE_ADMIN'],
100     label: 'Crear profesores',
101   },
102   {
103     id: 7,
104     icon: newspaper,
105     path: '/create/subjects',
106     role: ['ROLE_ADMIN'],
107     label: 'Crear materias',
108   },
109   {
```

4.2.7. Rutas según el rol y botón de Logout

```
App.jsx x
52  const routes : {{path: string, role: string[]... = [
106      role: ['ROLE_ADMIN'],
107      label: 'Crear materias',
108    },
109    {
110      id: 8,
111      icon: business,
112      path: '/create/classRooms',
113      role: ['ROLE_ADMIN'],
114      label: 'Crear aulas',
115    },
116  ];
117
118  setupIonicReact();
119
120  export default function App() { Show usages sanTorres770 +1
121
122    const {login, handlerLogout} = useApp()
123
124    const handleLogout = () :void => { Show usages CamiRinconR +1
125      handlerLogout();
126      toast.info( content: 'Sesión cerrada correctamente');
127
128      setTimeout( handler: () :void => {
129        window.location.href = '/startPage';
130      }, timeout: 1500)
131    };
132
133
134    const isVerified = sessionStorage.getItem( key: 'verified') === 'true'
135
136
```

4.2.8.Verificación de autenticación 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.isAuthenticated || !isVerified) && (
              <IonItem routerLink="/login" routerDirection="none">
                <IonIcon slot="start" icon={loginOutline}/>
                <IonLabel>Iniciar sesión</IonLabel>
              </IonItem>
            )}
            {routes.map(route => ((login.isAuthenticated && 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>
            )}
          </IonList>
          {!(login.isAuthenticated && isVerified) && (
            <IonItem button={true} onClick={() => handleLogout()}>

```

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

```

167         <IonIcon slot="start" icon={logout} />
168         <IonLabel>Cerrar sesión</IonLabel>
169       </IonItem>
170     </IonList>
171   </IonContent>
172 </IonMenu>
173 <IonPage id="main-content">
174   <IonHeader>
175     <IonToolbar style={{marginTop: '40px'}}>
176       <IonButtons slot="end">
177         <IonMenuButton />
178       </IonButtons>
179       
180     </IonToolbar>
181   </IonHeader>
182   <IonContent>
183     <IonTabs>
184       <IonRouterOutlet>

```

4.2.10. Rutas

```

194     <Route exact path="/startPage" component={StartPage} />
195     <Route exact path="/courses" component={CoursePage} />
196     <Route exact path="/profile" component={ProfilePage} />
197     <Route exact path="/tabs" component={TabsPage} />
198     <Route exact path="/Login" component={Login} />
199     <Route exact path="/verify" component={VerifyUser} />
200     <Route path="/student/periods" render={() => <PeriodsListStudent/>} exact={true}/>
201     <Route path="/schedule" render={() => <ScheduleView/>} exact={true}/>
202     <Route path="/teacher/periods" render={() => <PeriodsListTeacher/>} exact={true}/>
203     <Route path="/create/periods" render={() => <CreatePeriodForm/>} exact={true}/>
204     <Route path="/create/students" exact>
205       <NewStudentForm />
206     </Route>
207     <Route path="/create/teachers" exact>
208       <NewTeacherForm />
209     </Route>
210     <Route path="/create/subjects" exact>
211       <NewSubjectForm />
212     </Route>
213     <Route path="/create/classRooms" exact>
214       <NewClassRoomForm />
215     </Route>
216     <Redirect to="/startPage"/>
217   </IonRouterOutlet>
218   <IonTabBar slot="bottom">
219     <IonTabButton tab='startPage' href='/startPage'>
220

```

4.2.11. Rutas y fin de estructura

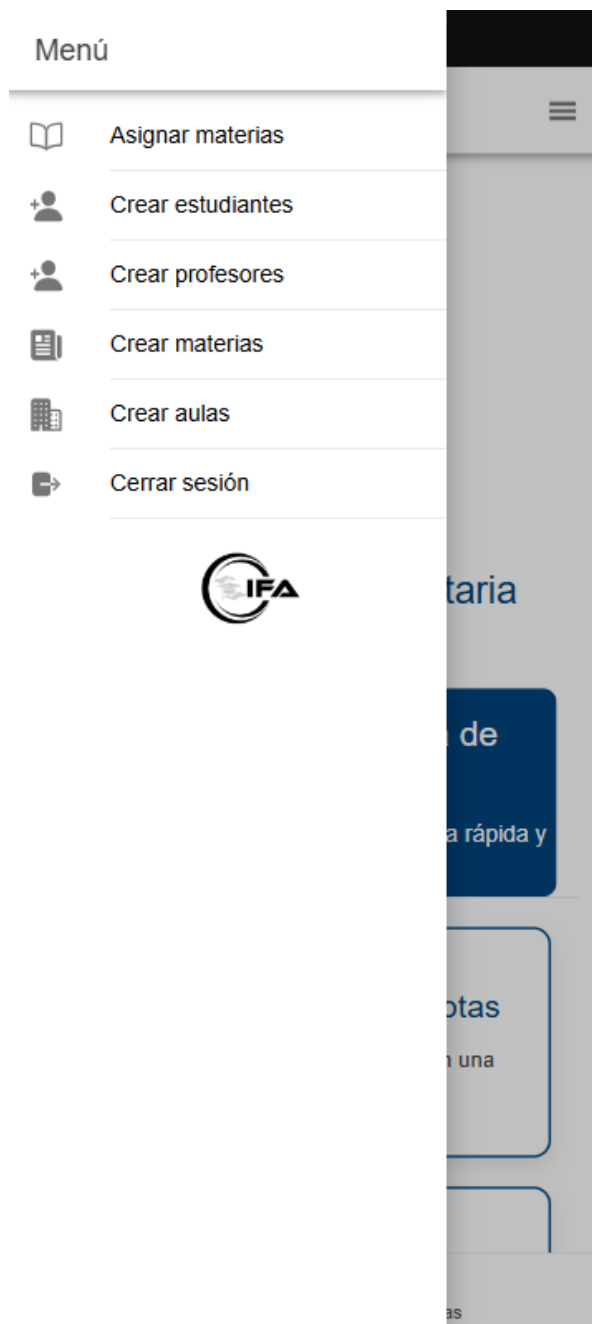
```

220     <IonTabButton tab='startPage' href='/startPage'>
221       <IonIcon icon={homeOutline}/>
222       <IonLabel>Inicio</IonLabel>
223     </IonTabButton>
224
225     <IonTabButton tab='courses' href='/courses'>
226       <IonIcon icon={bookOutline}/>
227       <IonLabel>Carreras</IonLabel>
228     </IonTabButton>
229
230     </IonTabBar>
231   </IonTabs>
232 </IonContent>
233 </IonPage>
234 </IonReactRouter>
235 <ToastContainer/>
236 </IonApp>
237 );
238 }

```

5. Administrador:

5.1. Menu:



5.2. Registro de grupos:

Formulario para la creación de Periodo

Nombre del periodo/grupo

Fecha inicial

Fecha final

Seleccione el curso para filtrar materias y estudiantes.

Selección de curso

Asignar las materias con su docente y estudiantes.

+ AÑADIR MATERIA

No se han seleccionado materias

CREAR PERIODO/GRUPO

Inicio Carreras

CANCELAR Asignar profesor a materia CONFIRMAR

Seleccione la materia

Selección de materia

Seleccione el docente

Selección de docente

Seleccione los estudiantes

Selección de estudiantes

Selección de horarios

+ LUNES

+ MARTES

+ MIÉRCOLES

+ JUEVES

+ VIERNES

+ SÁBADO

+ DOMINGO

5.3. Registro de aulas:

Formulario para la creación de aulas

Nombre de salón

Salón

Piso del salón

1

CREAR AULA

Inicio Carreras

5.4. Registro de materias:

IFA

Selección de cursos

Ingeniería de sistemas

contaduría

Formulario para la creación de materias

Nombre materia

Constitución política

Número de créditos

2

Código de la materia

1543

SELECCIONAR CURSOS

CREAR MATERIA

Inicio Carreras

GUARDAR SELECCIÓN

5.5. Registro de estudiantes:

IFA

Formulario para la creación de estudiantes

Nombre completo

Nombre completo

Número de documento

N° Documento

Correo electrónico

alguien@alguien.com

Número de celular

3105689654

Asignar curso estudiante ▾

CREAR ESTUDIANTE

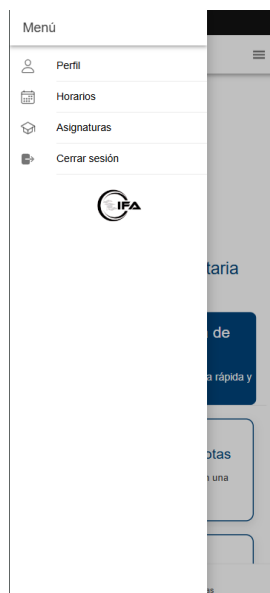
Inicio Carreras

5.6. Registro de docentes:

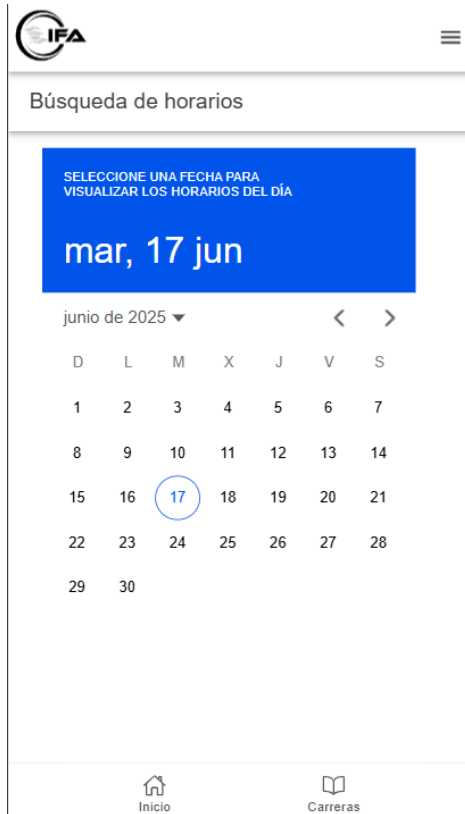
The screenshot shows a mobile application interface for creating a teacher profile. At the top left is the IFA logo, and at the top right is a hamburger menu icon. The title bar is dark blue with the text 'Formulario para la creación de profesores'. Below the title bar are several input fields: 'Usuario' (empty), 'Contraseña' (empty), 'Correo electronico' (containing 'alguien@alguien.com'), 'Nombres' (empty), 'Apellidos' (empty), and 'Número de celular' (containing '3105689654'). A blue button labeled 'CREAR PROFESOR' is positioned below the last input field. At the bottom of the screen is a navigation bar with two icons: 'Inicio' (home) and 'Carreras' (courses).

6. Docente:

6.1. Menu:



6.2. Consultar agenda diaria:



IFA

Búsqueda de horarios

SELECCIONE UNA FECHA PARA VISUALIZAR LOS HORARIOS DEL DÍA

mar, 17 jun

junio de 2025

D	L	M	X	J	V	S
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

Inicio Carreras

6.3. Consultar asignaturas:

The screenshot displays a mobile application interface for consulting subjects. On the left, a sidebar menu shows a list of semesters: 'Semestre 1 - Ing. de sistemas', 'Semestre 4 - Ing. de sistemas', 'Semestre 5 - Admin. de empresas', 'Semestre 4 - Medicina', and 'Semestre 2 - Ing. mecatronica'. The 'MATEMATICAS BÁSICAS' subject is selected, indicated by an orange arrow. The main content area shows details for 'Matematicas básicas', including a 'CERRAR' button, validity dates 'Del 2025-05-01 al 2025-05-31', and a schedule table. The schedule table has columns for 'Día', 'Inicio', 'Fin', and 'Aula', with rows for 'Lunes' (07:00-08:00, A-101) and 'Martes' (09:00-10:00, B-202). Below the schedule is a 'Lista de estudiantes' section with two names: 'PACHO CALAS' and 'CRISTIANO PEREZ'. The 'CRISTIANO PEREZ' name is selected, indicated by an orange arrow. On the right, a 'Calificaciones' section shows four notes, each with a score of 5, and a 'Definitiva' status with a score of 5. The interface also includes a top navigation bar with 'CANCELAR', 'Cristiano Perez', and 'CONFIRMAR' buttons, and a bottom navigation bar with 'Inicio' and 'Carreras' icons.

Día	Inicio	Fin	Aula
Lunes	07:00	08:00	A-101
Martes	09:00	10:00	B-202

Nota	Calificación
Nota 1	5
Nota 2	5
Nota 3	5
Nota 4	5

Definitiva 5

7. Estudiante:

7.1. Menu:



7.2. Calificaciones:

The screenshot displays the IFA interface for course 'Algoritmos 1'. On the left, a sidebar shows the course selection path: 'MATEMATICAS BÁSICAS' > 'ALGORITMOS 1'. The course details on the right include the teacher 'Pedra Torres', the validity period 'Del 2025-05-01 al 2025-05-31', and the average grade 'Nota promedio: 2.5'. Below this, the 'Horario' (Schedule) is shown in a table format:

Día	Inicio	Fin	Aula
Lunes	07:00	08:00	A-101
Domingo	06:00	07:00	E-505
Sabado	09:00	10:00	B-202

At the bottom, the 'Calificaciones' (Grades) section shows a list of grades for 'Nota 1' through 'Nota 4', all currently at 0, and a 'Definitiva' (Final) grade also at 0.

7.3. Consultar agenda diaria:

The screenshot shows the 'Búsqueda de horarios' (Schedule Search) interface. It features a calendar for 'junio de 2025' with the date 'mar, 17 jun' selected. A message above the calendar reads: 'SELECCIONE UNA FECHA PARA VISUALIZAR LOS HORARIOS DEL DÍA'. Below the calendar, a message states: 'No hay materias en la fecha seleccionada!'. The interface includes navigation arrows and a bottom menu with 'Inicio' and 'Carreras' options.

8. Documentacion backend:

8.1. Rutas para obtención y creación en el backend:

```

SpringSecurityConfig.java x
1  package com.santorres.academica.configuration;
2
3  > import ...
24
25  @Configuration  sanTorres770 +2
26  public class SpringSecurityConfig {
27
28      private final AuthenticationConfiguration authenticationConfiguration; 4 usages
29
30      public SpringSecurityConfig(AuthenticationConfiguration authenticationConfiguration) {  sanTorres770
31          this.authenticationConfiguration = authenticationConfiguration;
32      }
33
34      @Bean  sanTorres770
35      PasswordEncoder passwordEncoder() { return new BCryptPasswordEncoder(); }
38
39      @Bean  sanTorres770
40      AuthenticationManager authenticationManager() throws Exception {
41          return authenticationConfiguration.getAuthenticationManager();
42      }
43
44      @Bean  sanTorres770 +2
45      SecurityFilterChain filterChain(HttpSecurity http) throws Exception{
46          return http.authorizeHttpRequests( AuthorizationManagerRequestMat... authRules -> authRules
47              .requestMatchers(HttpMethod.GET, @"/api/users/{id}").permitAll()
48              .requestMatchers(HttpMethod.GET, @"/api/users").permitAll()
49              .requestMatchers(HttpMethod.POST, @"/api/users/verify").permitAll()
50              .requestMatchers(HttpMethod.GET, @"/api/students").permitAll()
51              .requestMatchers(HttpMethod.GET, @"/api/students/{id}").permitAll()
52              .requestMatchers(HttpMethod.POST, @"/api/students").permitAll()
53              .requestMatchers(HttpMethod.POST, @"/api/teachers").permitAll()
54              .requestMatchers(HttpMethod.GET, @"/api/teachers").permitAll()

```

8.2. Rutas para obtención y creación en el backend:

```

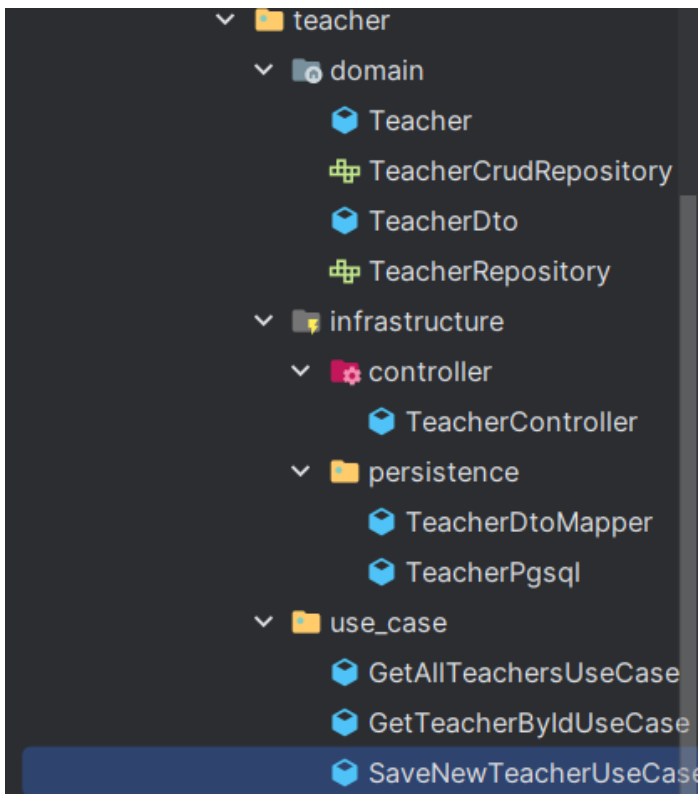
54      .requestMatchers(HttpMethod.GET, @"/api/teachers").permitAll()
55      .requestMatchers(HttpMethod.GET, @"/api/teachers/{id}").permitAll()
56      .requestMatchers(HttpMethod.POST, @"/api/subjects").permitAll()
57      .requestMatchers(HttpMethod.GET, @"/api/courses").permitAll()
58      .requestMatchers(HttpMethod.GET, @"/api/periods").permitAll()
59      .requestMatchers(HttpMethod.POST, @"/api/periods").permitAll()
60      .requestMatchers(HttpMethod.GET, @"/api/periods/student/{studentId}").permitAll()
61      .requestMatchers(HttpMethod.GET, @"/api/periods/teacher/{studentId}").permitAll()
62      .requestMatchers(HttpMethod.GET, @"/api/periods/student/last/{studentId}").permitAll()
63      .requestMatchers(HttpMethod.GET, @"/api/periods/teacher/last/{studentId}").permitAll()
64      .requestMatchers(HttpMethod.GET, @"/api/classrooms").permitAll()
65      .requestMatchers(HttpMethod.POST, @"/api/classrooms").permitAll()
66      .requestMatchers(HttpMethod.PUT, @"/api/notes").permitAll()
67      .requestMatchers(HttpMethod.GET, @"/api/schedules/student/{studentId}").permitAll()
68      .requestMatchers(HttpMethod.GET, @"/api/schedules/teacher/{teacherId}").permitAll()
69      .requestMatchers(HttpMethod.POST, @"/confirmation/user/email").permitAll()
70      .anyRequest().authenticated())
71      .addFilter(new JwtAuthenticationFilter(authenticationConfiguration.getAuthenticationManager()))
72      .addFilter(new JwtValidationFilter(authenticationConfiguration.getAuthenticationManager()))
73      .csrf(AbstractHttpConfigurer::disable)
74      .sessionManagement( SessionManagementConfigurer<HttpSecurity> management -> management.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
75      .cors( CorsConfigurer<HttpSecurity> cors -> cors.configurationSource(corsConfigurationSource()))
76      .build();
77  }
78
79  @Bean  sanTorres770
80  CorsConfigurationSource corsConfigurationSource() {
81
82      CorsConfiguration config = new CorsConfiguration();

```

8.3. Rutas para obtención y creación en el backend

```
79 := @Bean sanTorres770
80 CorsConfigurationSource corsConfigurationSource() {
81
82     CorsConfiguration config = new CorsConfiguration();
83     config.setAllowedOrigins(Arrays.asList("http://localhost:8100"));
84     config.setAllowedOriginPatterns(Arrays.asList("*"));
85     config.setAllowedMethods(Arrays.asList("GET", "POST", "PUT", "DELETE"));
86     config.setAllowedHeaders(Arrays.asList("Authorization", "Content-Type", "X-Requested-With"));
87     config.setAllowCredentials(true);
88
89     UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
90     source.registerCorsConfiguration( pattern: "/*", config);
91     return source;
92 }
93
94 := @Bean sanTorres770
95 FilterRegistrationBean<CorsFilter> corsFilter() {
96     FilterRegistrationBean<CorsFilter> bean = new FilterRegistrationBean<(<
97         new CorsFilter(corsConfigurationSource()));
98     bean.setOrder(Ordered.HIGHEST_PRECEDENCE);
99     return bean;
100 }
101
102 }
103
```

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 GetAllTeachersUseCase 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 serie 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. Class Teacher.java:

```

Teacher.java x
1 package com.santorres.academica.teacher.domain;
2
3 > import ...
4
5
6
7
8
9
10 @Entity
11 @Table
12 @Getter
13 @Setter
14 @NoArgsConstructor
15 @AllArgsConstructor
16
17
18 public class Teacher {
19     @Id
20     @GeneratedValue(strategy = GenerationType.UUID)
21     private String id;
22     private String name;
23     private String email;
24     private String phone;
25
26     @Column(unique = true, nullable = true)
27     private String documentNumber;
28
29     @OneToOne(cascade = CascadeType.ALL)
30     @MapsId
31     @JoinColumn(name = "user_id")
32     private User user;
33

```

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.

```

TeacherCrudRepository.java x
1 package com.santorres.academica.teacher.domain;
2
3 import org.springframework.data.repository.CrudRepository;
4
5 public interface TeacherCrudRepository extends CrudRepository<Teacher, String> {
6     }
7

```

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 ×
1  package com.santorres.academica.teacher.domain;
2
3  > import ...
4
5
6
7
8
9  @Getter 23 usages 👤 Andrea Granada +1
10 @Setter
11 @NoArgsConstructor
12 @AllArgsConstructor
13
14 public class TeacherDto {
15     private String id;
16     private String name;
17     private String email;
18     private String phone;
19     private String documentNumber;
20     private UserDto user;
21 }
```

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.

```
TeacherDtoMapper.java x
1 package com.santorres.academica.teacher.infrastructure.persistence;
2
3 > import ...
10
11 public class TeacherDtoMapper { 6 usages Andrea Granada +1
12
13     private Teacher teacher; 8 usages
14
15     private TeacherDtoMapper () {}; 1 usage Andrea Granada
16
17 @ public static TeacherDtoMapper builder() {return new TeacherDtoMapper ();} Andrea Granada
18
19     public TeacherDtoMapper setTeacher(Teacher teacher) { Andrea Granada
20         this.teacher = teacher;
21         return this;
22     }
23
24     public TeacherDto build(){ Andrea Granada +1
25         if (teacher == null) {
26             throw new RuntimeException("Teacher is null");
27         }
28
29         return new TeacherDto(
30             this.teacher.getId(),
31             this.teacher.getName(),|
32             this.teacher.getEmail(),
33             this.teacher.getPhone(),
34             this.teacher.getDocumentNumber(),
35             UserDtoMapper.builder().setUser(this.teacher.getUser()).build()
36
37         );
38     }
```

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.

```
TeacherRepository.java x
1 package com.santorres.academica.teacher.domain;
2
3 import java.util.List;
4 import java.util.Optional;
5
6 public interface TeacherRepository { 11 usages 1 implementation sanTorres770
7     Optional<TeacherDto> saveNew(Teacher teacher); 1 implementation sanTorres770
8
9     List<TeacherDto> getAllTeachers(); 1 usage 1 implementation sanTorres770
10
11     TeacherDto getTeacherById(String id); 1 usage 1 implementation sanTorres770
12 }
13
```

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

```

TeacherPgsql.java x
1  package com.santorres.academica.teacher.infrastructure.persistence;
2
3  > import
12
13  @Repository  sanTorres770
14  public class TeacherPgsql implements TeacherRepository {
15
16      private final TeacherCrudRepository teacherCrudRepository; 4 usages
17
18      public TeacherPgsql(TeacherCrudRepository teacherCrudRepository) {  sanTorres770
19          this.teacherCrudRepository = teacherCrudRepository;
20      }
21
22      @Override  sanTorres770
23      public Optional<TeacherDto> saveNew(Teacher teacher) {
24          return Optional.of(toDto(teacherCrudRepository.save(teacher)));
25      }
26
27      @Override 1 usage  sanTorres770
28      public List<TeacherDto> getAllTeachers() {
29
30          List<Teacher> teachers = (List<Teacher>) teacherCrudRepository.findAll();
31
32          List<TeacherDto> teacherDtos = new ArrayList<>();
33
34          for (Teacher teacher : teachers) {
35
36              teacherDtos.add(toDto(teacher));
37          }
38
39          return teacherDtos;

```

```

TeacherPgsql.java x
14  public class TeacherPgsql implements TeacherRepository {
15
16      }
17
18
19
20
21
22
23
24
25
26
27      @Override 1 usage  sanTorres770
28      public List<TeacherDto> getAllTeachers() {
29
30          List<Teacher> teachers = (List<Teacher>) teacherCrudRepository.findAll();
31
32          List<TeacherDto> teacherDtos = new ArrayList<>();
33
34          for (Teacher teacher : teachers) {
35
36              teacherDtos.add(toDto(teacher));
37          }
38
39          return teacherDtos;
40      }
41
42      @Override 1 usage  sanTorres770
43      public TeacherDto getTeacherById(String id) { return toDto(teacherCrudRepository.findById(id).orElse( other: null)); }
44
45
46      private TeacherDto toDto(Teacher teacher) { 3 usages  sanTorres770
47
48          return TeacherDtoMapper.builder().setTeacher(teacher).build();
49      }
50  }
51
52

```

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”

```

TeacherController.java x
1 package com.santorres.academica.teacher.infrastructure.controller;
2
3 import ..
4
5
6
7
8
9
10
11
12
13
14
15
16
17 @RestController
18 @RequestMapping("/api/teachers")
19 public class TeacherController {
20
21     private final SaveNewTeacherUseCase saveNewTeacherUseCase;
22     private final GetAllTeachersUseCase getAllTeachersUseCase;
23     private final GetTeacherByIdUseCase getTeacherByIdUseCase;
24
25     public TeacherController(SaveNewTeacherUseCase saveNewTeacherUseCase, GetAllTeachersUseCase getAllTeachersUseCase, GetTeacherByIdUseCase getTeacherByIdUseCase) {
26         this.saveNewTeacherUseCase = saveNewTeacherUseCase;
27         this.getAllTeachersUseCase = getAllTeachersUseCase;
28         this.getTeacherByIdUseCase = getTeacherByIdUseCase;
29     }
30
31     @PostMapping
32     public ResponseEntity<?> addTeacher(@Valid @RequestBody Teacher teacher, BindingResult result) {
33
34         if (result.hasErrors()) {
35             return FormValidationUtil.validation(result);
36         }
37
38         return ResponseEntity.status(HttpStatus.CREATED).body(saveNewTeacherUseCase.saveNewTeacher(teacher));
39     }
40
41
42     @GetMapping
43     public List<TeacherDto> getAllTeachers() { return getAllTeachersUseCase.getAllTeachers(); }
44 }

```

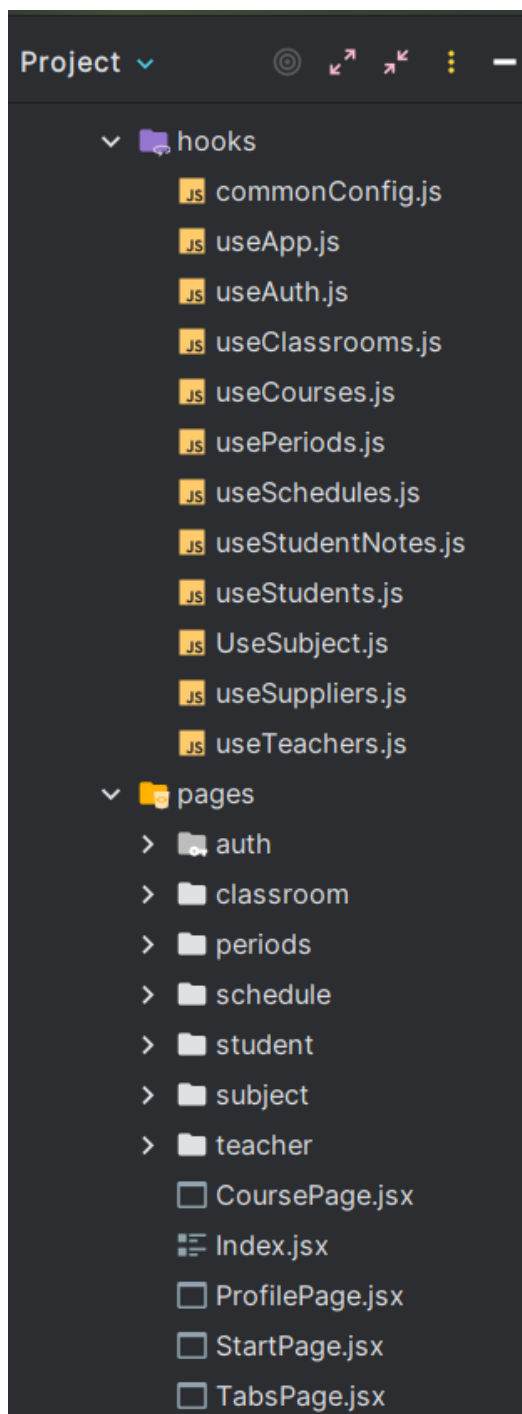
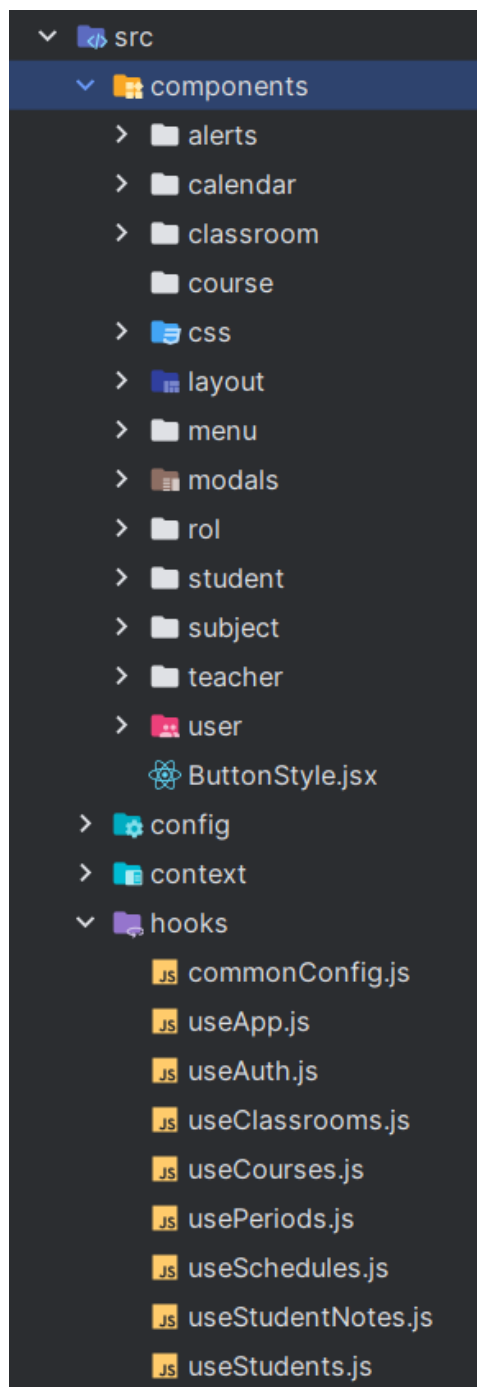
```

TeacherController.java x
19 public class TeacherController {
20
21     private final SaveNewTeacherUseCase saveNewTeacherUseCase;
22     private final GetAllTeachersUseCase getAllTeachersUseCase;
23     private final GetTeacherByIdUseCase getTeacherByIdUseCase;
24
25     public TeacherController(SaveNewTeacherUseCase saveNewTeacherUseCase, GetAllTeachersUseCase getAllTeachersUseCase, GetTeacherByIdUseCase getTeacherByIdUseCase) {
26         this.saveNewTeacherUseCase = saveNewTeacherUseCase;
27         this.getAllTeachersUseCase = getAllTeachersUseCase;
28         this.getTeacherByIdUseCase = getTeacherByIdUseCase;
29     }
30
31     @PostMapping
32     public ResponseEntity<?> addTeacher(@Valid @RequestBody Teacher teacher, BindingResult result) {
33
34         if (result.hasErrors()) {
35             return FormValidationUtil.validation(result);
36         }
37
38         return ResponseEntity.status(HttpStatus.CREATED).body(saveNewTeacherUseCase.saveNewTeacher(teacher));
39     }
40
41
42     @GetMapping
43     public List<TeacherDto> getAllTeachers() { return getAllTeachersUseCase.getAllTeachers(); }
44
45
46     @GetMapping("/{id}")
47     public ResponseEntity<?> getTeacherById(@PathVariable String id){
48         return ResponseEntity.status(HttpStatus.OK).body(getTeacherByIdUseCase.getTeacherById(id));
49     }
50 }
51 }
52

```

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 reusable 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 locales, 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 también vemos el inicio 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 NewStudentForm.

Seguido a esas 4 imágenes que describimos anteriormente, las 2 siguientes a éstas 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 <code>handleSubmit()</code>	Lo ejecuta al hacer <code>onSubmit={...}</code>
Maneja <code>courseRef</code>	Define el useRef y lo pasa como prop	Actualiza su valor al seleccionar curso
Función <code>saveNewStudent</code>	La ejecuta desde <code>useApp()</code>	No tiene lógica de backend
Campos visuales (InputForm)	No los renderiza	Sí, los genera con <code>inputs.map(...)</code>
Select de cursos	Pasa <code>courseRef</code> y obtiene cursos desde contexto	Lo muestra con IonSelect

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

```
NewStudentForm.jsx ×
1  import StudentForm from "../../components/student/StudentForm.jsx";
2  import useApp from "../../hooks/useApp.js";
3  import { useRef, useState } from "react";
4  import { toast } from "react-toastify";
5  import LoadingAlert from "../../components/alerts/LoadingAlert.jsx";
6
7  export default function NewStudentForm() { Show usages CamiRinconR +1
8      const nameRef : RefObject<null> = useRef( initialValue: null);
9      const emailRef : RefObject<null> = useRef( initialValue: null);
10     const phoneRef : RefObject<null> = useRef( initialValue: null);
11     const courseRef : RefObject<string> = useRef( initialValue: '');
12     const documentRef : RefObject<null> = useRef( initialValue: null);
13
14     const {validationErrors,
15         setValidationErrors,
16         saveNewStudent} = useApp()
17
18     const [errores, setErrores] = useState( initialState: []);
19     const [isLoading : boolean, setIsLoading] = useState( initialState: false);
20
21
22     const handleRestartValue = () : void => { Show usages CamiRinconR +1
23         nameRef.current.value = '';
24         phoneRef.current.value = '';
25         emailRef.current.value = '';
26         courseRef.current = '';
27         documentRef.current.value = '';
28     };
29
30     const handleSubmit = (e) : void => { Show usages CamiRinconR +1
31         e.preventDefault();
32         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 x
7   export default function NewStudentForm() { Show usages CamiRinconR +1
81   <StudentForm
92   },
93   {
94     id: '2',
95     inputId: 'documentNumber',
96     type: 'text',
97     reference: documentRef,
98     placeholder: 'N° Documento',
99     labelValue: 'Número de documento',
100  },
101  {
102    id: '3',
103    inputId: 'email',
104    type: 'email',
105    reference: emailRef,
106    placeholder: 'alguien@alguien.com',
107    labelValue: 'Correo electrónico',
108  },
109  {
110    id: '4',
111    inputId: 'phone',
112    type: 'tel',
113    reference: phoneRef,
114    placeholder: '3105689654',
115    labelValue: 'Número de celular',
116  },
117  ]}
118  nameButton={'Crear estudiante'}
119  />
120
121  );
```

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

```
StudentForm.jsx x
1  import InputForm from "../layout/InputForm.jsx";
2  import ButtonStyle from "../ButtonStyle.jsx";
3  import {
4    IonContent,
5    IonItem,
6    IonList,
7    IonPage,
8    IonSelect,
9    IonSelectOption,
10 } from "@ionic/react";
11 import "../css/generalStyle.css";
12 import useApp from "../../hooks/useApp.js";
13 import { useEffect } from "react";
14
15 function StudentForm({ inputs, nameButton, onSubmit, courseRef }) { Show usages CamiRinconR
16   const { allCourses, getAllCourses } = useApp();
17
18   useEffect( effect: () :void => {
19     getAllCourses();
20   }, deps: []);
21
22   return (
23     <IonPage>
24       <IonContent>
25         <div
26           className={"div-form"}
27           style={{
28             display: "flex",
29             justifyContent: "center",
30             alignItems: "center",
31             marginLeft: "20px",
32             marginRight: "20px",
```


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

```

1  > import ...
2
3
4  export const useStudents = () => { Show usages CamiRinconR +1
5
6  const [studentById, setStudentById] = useState( initialState: {} )
7
8  const config = () : {headers: {...}} => { Show usages CamiRinconR
9
10     return {
11       headers: {
12         Authorization: `Bearer ${sessionStorage.getItem( key: 'token' )}`
13       }
14     }
15
16     const saveNewStudent = async ( newStudent ) : Promise<AxiosResponse<...>> => { Show usages CamiRinconR
17
18       return await customAxios.post( url: "/api/students", newStudent, config() );
19     };
20
21     const getStudentById = async () : Promise<void> => { Show usages sanTorres770
22
23     try {
24       const { data } = await customAxios( ` /api/students/${sessionStorage.getItem( key: "queryId" )}`, config() );
25       setStudentById( data );
26     } catch ( error ) {
27
28     }
29
30     };
31
32     return {
33       saveNewStudent,
34       studentById,

```

El useStudent es el hook que encapsula la lógica relacionada con el 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 específico. 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.

```
SpringSecurityConfig.java x
26 public class SpringSecurityConfig {
42     }
43
44 @Bean
45 SecurityFilterChain filterChain(HttpSecurity http) throws Exception{
46     return http.authorizeHttpRequests( AuthorizationManagerRequestMat... authRules -> authRules
47         .requestMatchers(HttpMethod.GET, "/api/users/{id}").permitAll()
48         .requestMatchers(HttpMethod.GET, "/api/users").permitAll()
49         .requestMatchers(HttpMethod.POST, "/api/users/verify").permitAll()
50         .requestMatchers(HttpMethod.GET, "/api/students").permitAll()
51         .requestMatchers(HttpMethod.GET, "/api/students/{id}").permitAll()
52         .requestMatchers(HttpMethod.POST, "/api/students").permitAll()
53         .requestMatchers(HttpMethod.POST, "/api/teachers").permitAll()
54         .requestMatchers(HttpMethod.GET, "/api/teachers").permitAll()
55         .requestMatchers(HttpMethod.GET, "/api/teachers/{id}").permitAll()
56         .requestMatchers(HttpMethod.POST, "/api/subjects").permitAll()
57         .requestMatchers(HttpMethod.GET, "/api/courses").permitAll()
58         .requestMatchers(HttpMethod.GET, "/api/periods").permitAll()
59         .requestMatchers(HttpMethod.POST, "/api/periods").permitAll()
60         .requestMatchers(HttpMethod.GET, "/api/periods/student/{studentId}").permitAll()
```

De esta manera se manejan todos los componentes.

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.
	IntelliJ Idea / 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/>