DOCUMENTACIÓN DE PROYECTO

Por: Irving Eduardo Valdez Salazar

Entrevista de trabajo

Fecha y hora de inicio: 25/abril/2024 6:08pm

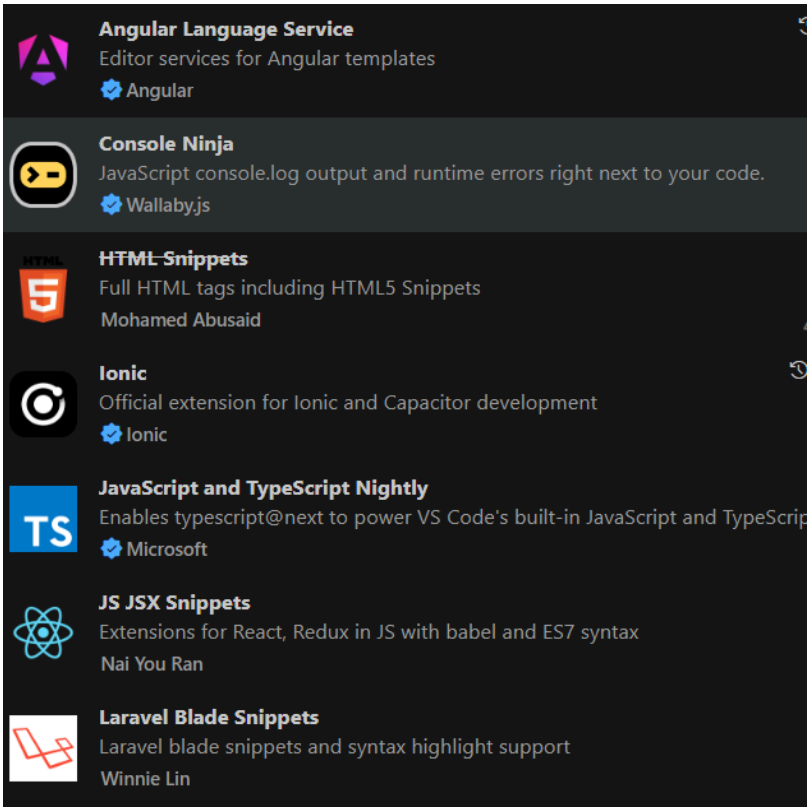
Fecha y hora de conclusión: 28/abril/2024 4:33am

ÍNDICE

Instrucciones de uso

Antes de empezar de lleno es bueno recalcar cómo es que se implementa cada una de estas dependencias y el preparar el entorno para que todo se ejecute correctamente
Instalación de ionic, para esto dejaré la página oficial del CLI en donde explica paso a paso de como instalarlo:
https://ionicframework.com/docs/intro/cli
Xampp:
https://www.apachefriends.org/es/download.html
Node.js
https://nodejs.org/en/download/current
Angular
https://angular.io/cli

Con todo esto podemos empezar migrando la base de datos (La cual estará junto con este proyecto)
Con esto ya puedes transportar todas las extensiones necesarias a VS code, las dejaré a continuación:



Me encantaría explicar cada una de estas pero por el momento no es tan importante, a rasgos generales te ayudarán a la prevención y detección rápida de errores.
Comandos para ejecución (Back):
Node app.js
Comandos para ejecución (Front)
ionic serve

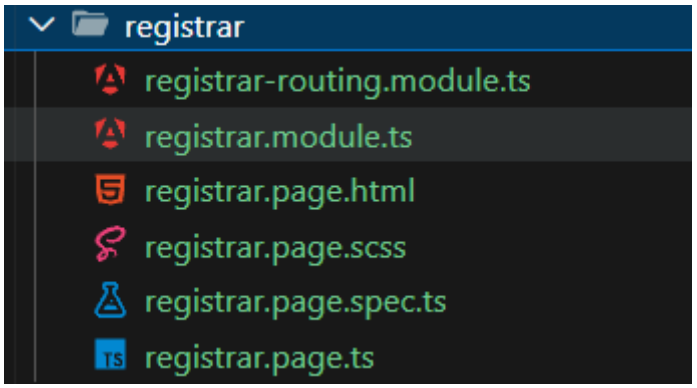
Con esto ya se puede testear la página

LogIn

Para iniciar sesión el usuario predefinido es Ramiro y la contraseña es “administrador134” lo dejé predefinido ya que quería experimentar con las migraciones y si todo está bien con las importaciones debería funcionar.
En caso de que no funcione, en la ruta quite el “/login” y ponga a continuación “/home”

¿Por qué Ionic?

Al momento de la confección de la página web, necesitaba una organización extraordinaria para terminarla en 3 días.
El proyecto se inició oficialmente a las 6 de la tarde del día 25 de abril del 2024 y se concluyó a las 4:33 am del 28 de abril del 2024 sin contar el tiempo de la documentación, esto quería decir que los frameworks que me ayudarían a hacer todo rápido como Laravel no me serviría de mucho, no se cumpliría con el objetivo, así que Ionic sería mi mejor compañero sería Ionic ya que contiene todas las herramientas necesarias y conicidad para poder desarrollar dicha página.
Tenemos en cada componente lo siguiente:



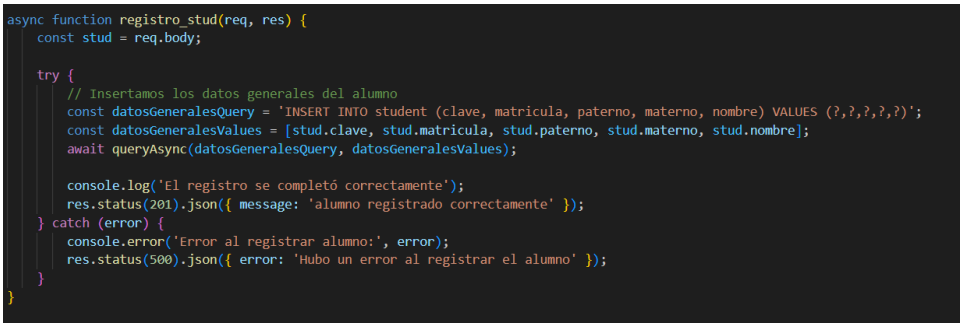
En primer lugar la carpeta que nos permite guardar todas las propiedades deseadas, esto incluye desde la lógica, los diseños y sin olvidar al HTML así que por excelencia antes de iniciar, ya sabía que Ionic sería el indicado, debido a su facilidad no era tan necesario empezar a hacer un proyecto ya que es fácil y sencillo crear uno.
Tenemos todos los elementos necesarios de una página web, así que todo es demasiado fácil y profesional.
¿Por qué no Laravel? Esto es sencillo de responder y es por que Laravel tiene demasiada facilidad de creación, tanto que podemos hacer un CRUD sin necesidad de nada, pero no se estaría cumpliendo la condición de usar una API, aunque se puede implementar, tampoco estaríamos agregando todas las funcionalidades y lógicas que Ionic nos permite

Angular

Ionic usa angular para varios métodos ¿Por qué no usar angular en vez de ionic? es porque ionic nos crea componentes indispensables de los que podemos aprovechar gran parte de su funcionalidad, mientras de angular funciona con componentes, ionic funciona con componentes y además con lógicas de componentes e implementaciones de inyección de código por el que podemos usar una página en un modal, también podemos hacer componentes puros para usarlos como placeholder, así podemos darle diseños y animaciones más profesionales a nuestros proyectos

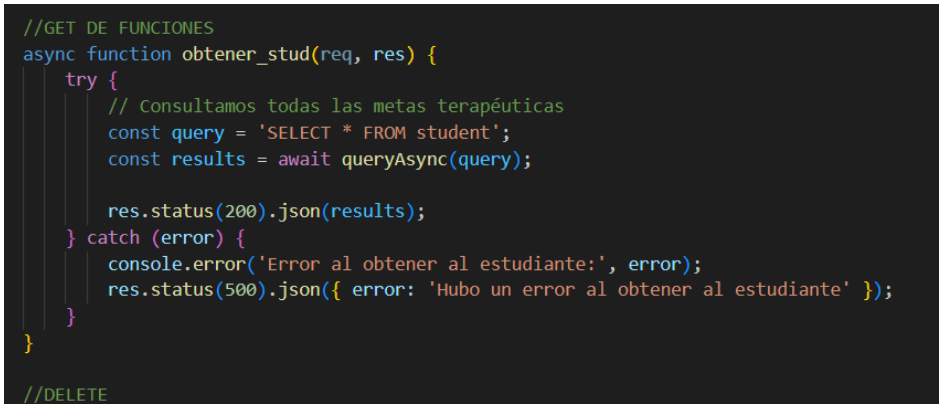
API

Para esta API tenemos los métodos más comunes los cuales son:
GET



```
async function registro_stud(req, res) {
    const stud = req.body;

    try {
        // Insertamos los datos generales del alumno
        const datosGeneralesQuery = 'INSERT INTO student (clave, matricula, paterno, materno, nombre) VALUES (?,?,?,?,?)';
        const datosGeneralesValues = [stud.clave, stud.matricula, stud.paterno, stud.materno, stud.nombre];
        await queryAsync(datosGeneralesQuery, datosGeneralesValues);

        console.log('El registro se completó correctamente');
        res.status(201).json({ message: 'alumno registrado correctamente' });
    } catch (error) {
        console.error('Error al registrar alumno:', error);
        res.status(500).json({ error: 'Hubo un error al registrar el alumno' });
    }
}
```

INSERT:



```
//GET DE FUNCIONES
async function obtener_stud(req, res) {
    try {
        // Consultamos todas las metas terapéuticas
        const query = 'SELECT * FROM student';
        const results = await queryAsync(query);

        res.status(200).json(results);
    } catch (error) {
        console.error('Error al obtener al estudiante:', error);
        res.status(500).json({ error: 'Hubo un error al obtener al estudiante' });
    }
}

//DELETE
```

DELETE

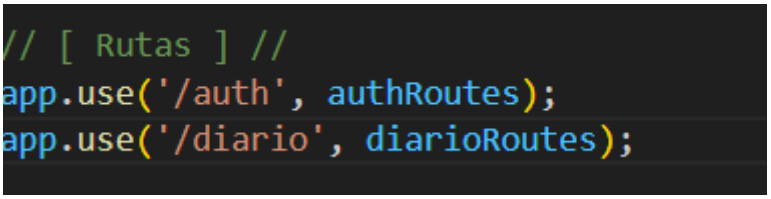
```
//DELETE

async function deleteUser(req, res) {
    const { id } = req.params;

    try {
      // Verificar si el usuario existe en la base de datos antes de borrarlo
      const userQuery = await queryAsync('SELECT * FROM student WHERE id = ?', [id]);
      const user = userQuery[0];

      if (!user) {
        return res.status(404).json({ error: 'Usuario no encontrado' });
      }

      // Borrar el usuario de la base de datos
      await queryAsync('DELETE FROM student WHERE id = ?', [id]);

      res.status(200).json({ message: 'Usuario eliminado exitosamente' });
    } catch (error) {
      console.error('Error al eliminar el usuario:', error);
      res.status(500).json({ error: 'Error en el servidor' });
    }
  }
```

PUT

```
//Editar
async function updateUser(req, res) {
  const { id } = req.params;
  const updatedUser = req.body;

  try {
    // Verificar si el usuario existe en la base de datos
    const userQuery = await queryAsync('SELECT * FROM student WHERE id = ?', [id]);
    const user = userQuery[0];
    if (!user) {
      return res.status(404).json({ error: 'Usuario no encontrado' });
    }

    // Actualizar los datos del usuario en la base de datos
    const updateQuery = 'UPDATE student SET clave = ?, matricula = ?, paterno = ?, materno = ?, nombre = ? WHERE id = ?';
    const values = [updatedUser.clave, updatedUser.matricula, updatedUser.paterno, updatedUser.materno, updatedUser.nombre, id];
    await queryAsync(updateQuery, values);

    res.status(200).json({ message: 'Usuario actualizado exitosamente' });
  } catch (error) {
    console.error('Error al actualizar el usuario:', error);
    res.status(500).json({ error: 'Error en el servidor' });
  }
}
```

Consumo de API

Para esto nos conectaremos con 2 cosas, rutas por parte del back, el cual se puede apreciar en la siguiente imagen:



```
// [ Rutas ] //
app.use('/auth', authRoutes);
app.use('/diario', diarioRoutes);
```

Por parte del back se definen las rutas
Y por parte del front:

```
@Injectable({
  providedIn: 'root',
})
export class LoginService {
  private ApiURL = environment.ApiURL;

  constructor(private http: HttpClient) { }

  iniciarSesion(usuario: Usuario): Observable<any> {
    const headers = new HttpHeaders({ 'Content-Type': 'application/json' });
    const inicioSesionUrl = `${this.ApiURL}/auth/login`;

    return this.http.post<any>(inicioSesionUrl, usuario, { headers });
  }

  createUser(userCreate: UserCreate): Observable<any> {
    const headers = new HttpHeaders({ 'Content-Type': 'application/json' });
    const createUserUrl = `${this.ApiURL}/diario/registro_stud`;
    return this.http.post(createUserUrl, userCreate, { headers });
  }

  deleteUser(id: string) {
    const url = `/diario/deleteUser/${id}`; // Cambia '/api/deleteUser' por la ruta correcta en tu backend
    return this.http.delete(url);
  }

  updateUser(updateUs: Updateus) {
  const url = `/updateUser/${updateUs.id}`;
  return this.http.put(url, updateUs);
}
```

Aquí apreciamos que cada método tiene su propia sentencia de la cual extraemos las rutas necesarias y hacemos referencias a las funciones respectivas de cada método, así aseguramos que todo funcione correctamente