

Manual de pasos para AdonisJs - typescript

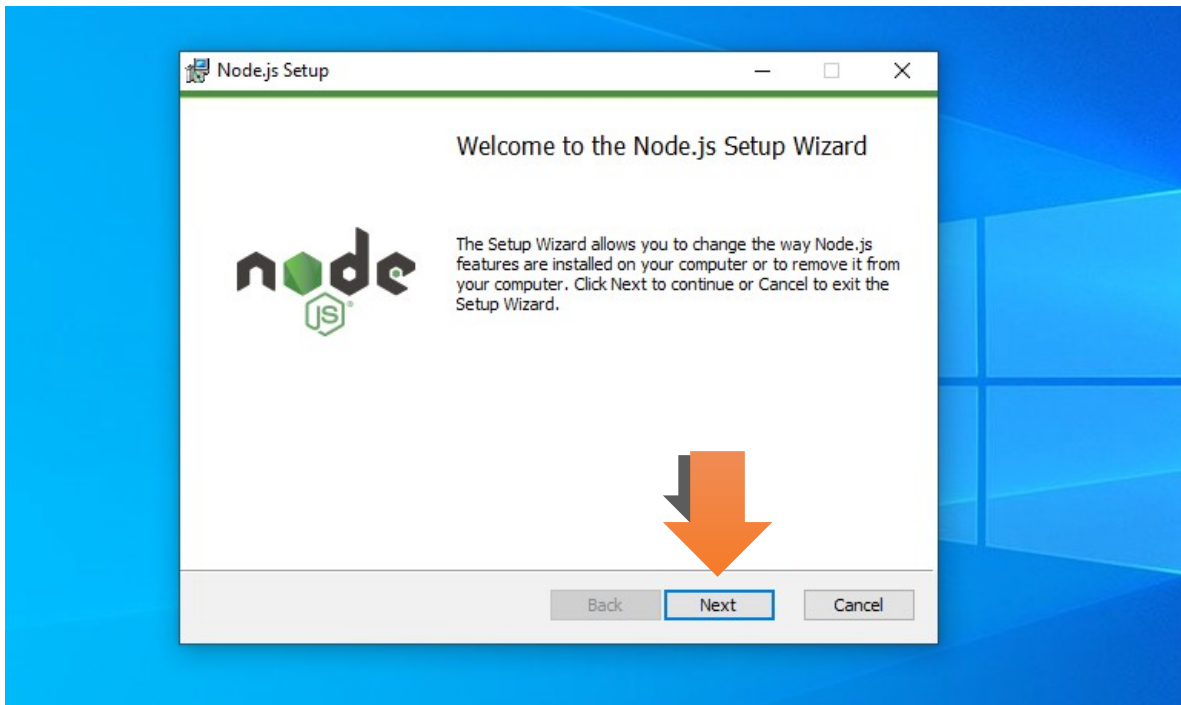
En este paso a paso vamos a crear una rest Api sencilla usando como framework de desarrollo AdonisJs y usando como motor de base de datos Mysql, si quieres usar otro motor de base de datos como Postgresql, Sql server, Oracle, es simplemente cambiando 3 parámetros dentro del archivo .env

Paso 1: INSTALAR NODEJS

Para instalar AdonisJs, lo primero que debemos instalar es Nodejs, para eso vamos a la pagina oficial de Nodejs y descargamos la última versión.

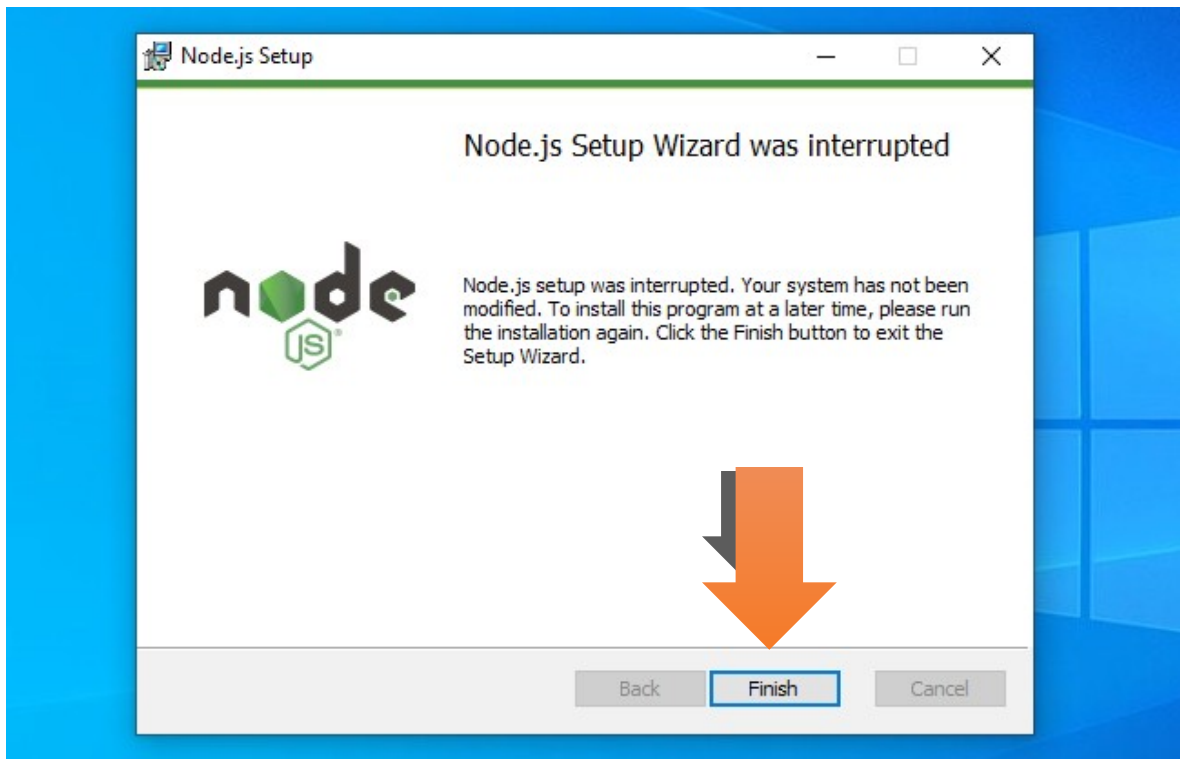
Sitio de descargar: <https://nodejs.org/en/>

Una vez terminado, procedemos a instalarlo.



Le damos en **NEXT** y a continuación **INSTALL**.

Una vez haya terminado la instalación le damos en **Finish**



Para ver si tenemos Nodejs instalado en nuestro equipo, vamos a la barra de inicio y escribimos **CMD** y abrimos una terminal de comandos.

```
Microsoft Windows [Versión 10.0.19042.804]
(c) 2020 Microsoft Corporation. Todos los derechos reservados.

C:\Users\EQUIPO>node -v
v14.16.0

C:\Users\EQUIPO>npm -v
6.14.11

C:\Users\EQUIPO>
```

Para ver si tenemos a Nodejs instalado, escribimos:

- node -v
- npm -v

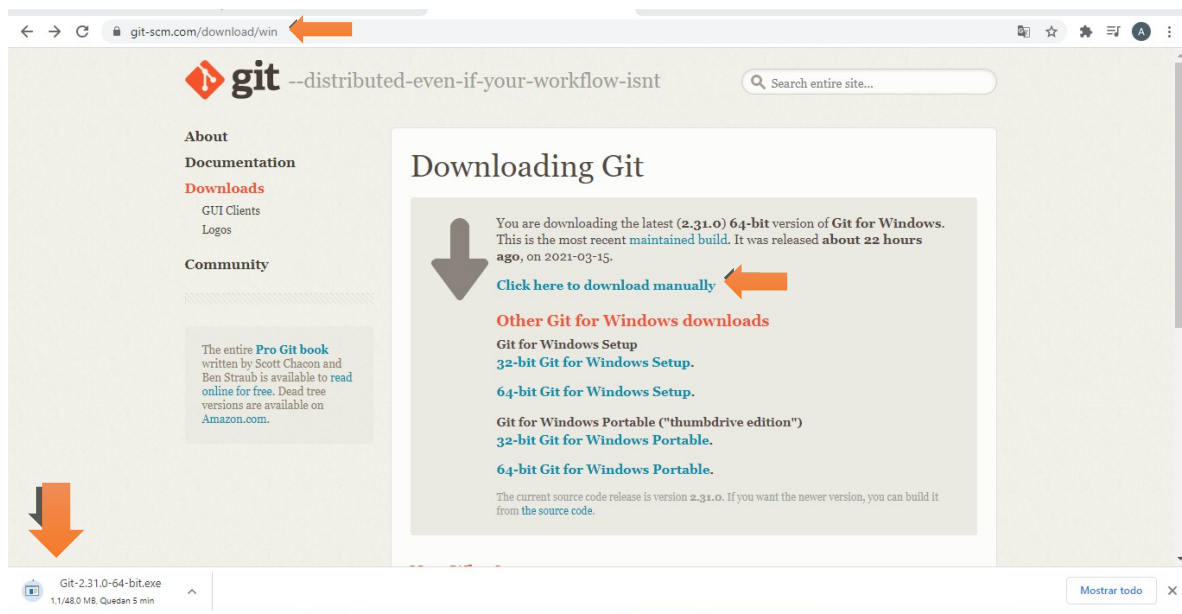
Como podemos apreciar tenemos ya Nodejs instalado en nuestro equipo, pero eso no nos garantiza que ya podemos crear un proyecto en AdonisJs, lo que tenemos que hacer es descargar el sistema de control de versiones Git para crear los

proyecto, ya que los proyectos de Adonisjs son descargados de un repositorio en GitHub.

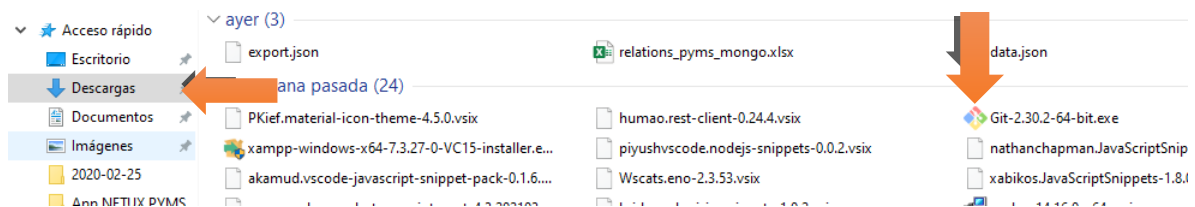
Paso 2: INSTALAR GIT

Para instalar Git nos dirigimos a la página de:

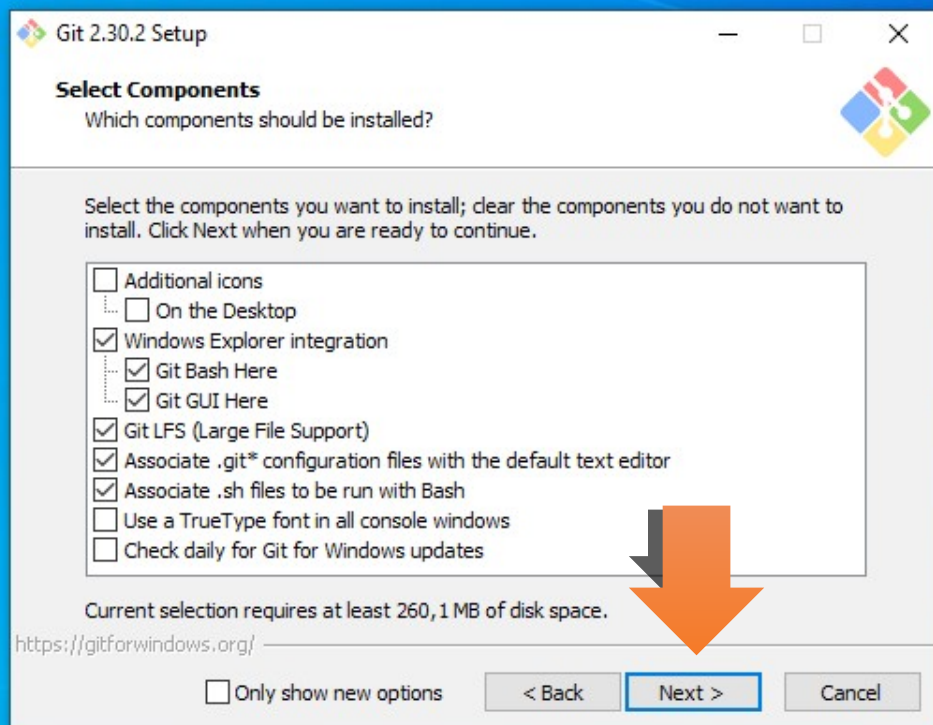
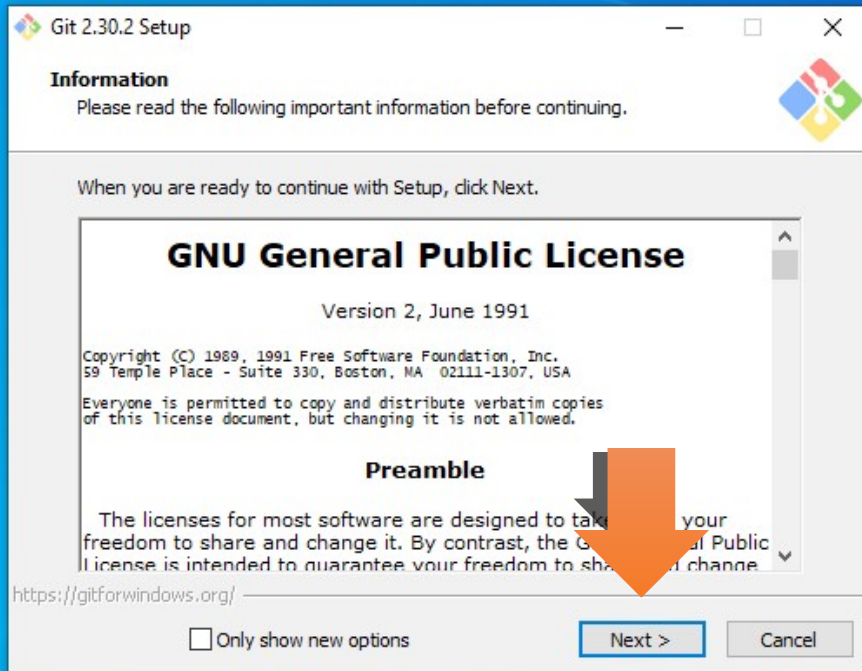
Sitio Web: <https://git-scm.com/download/win>

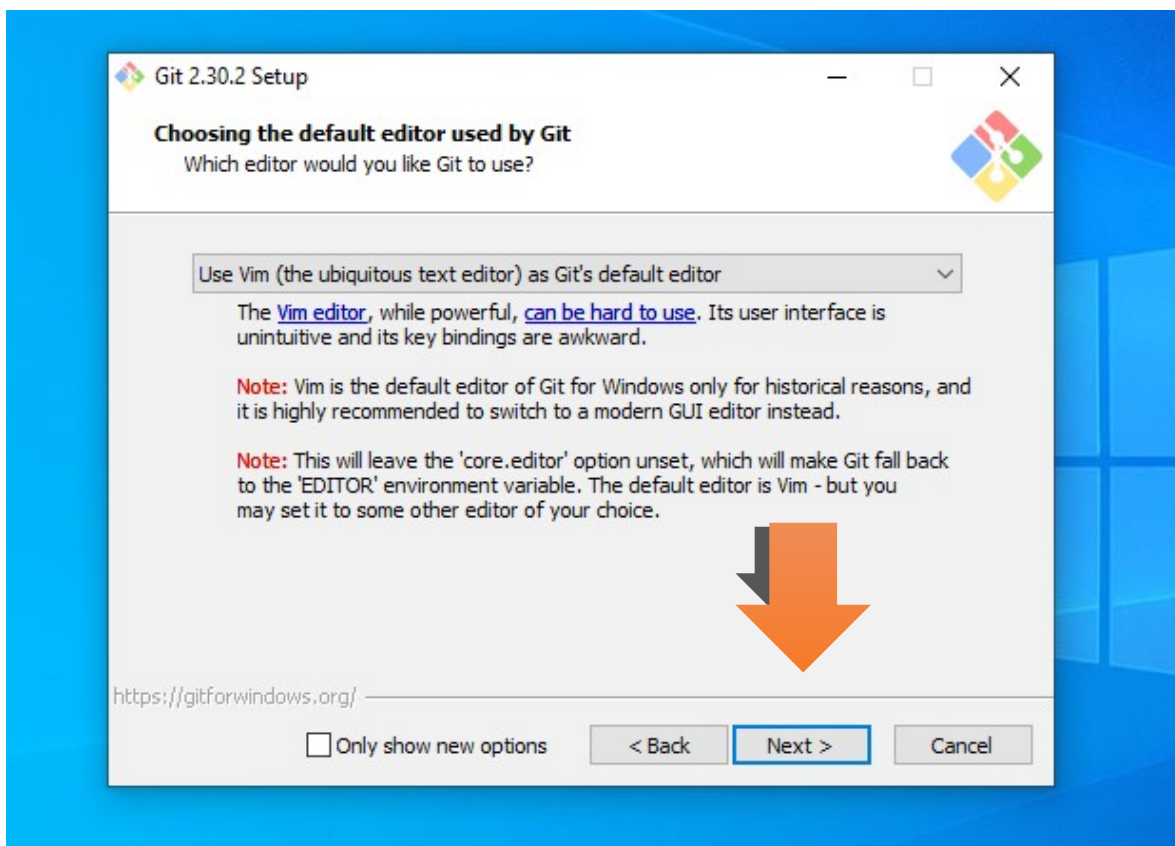
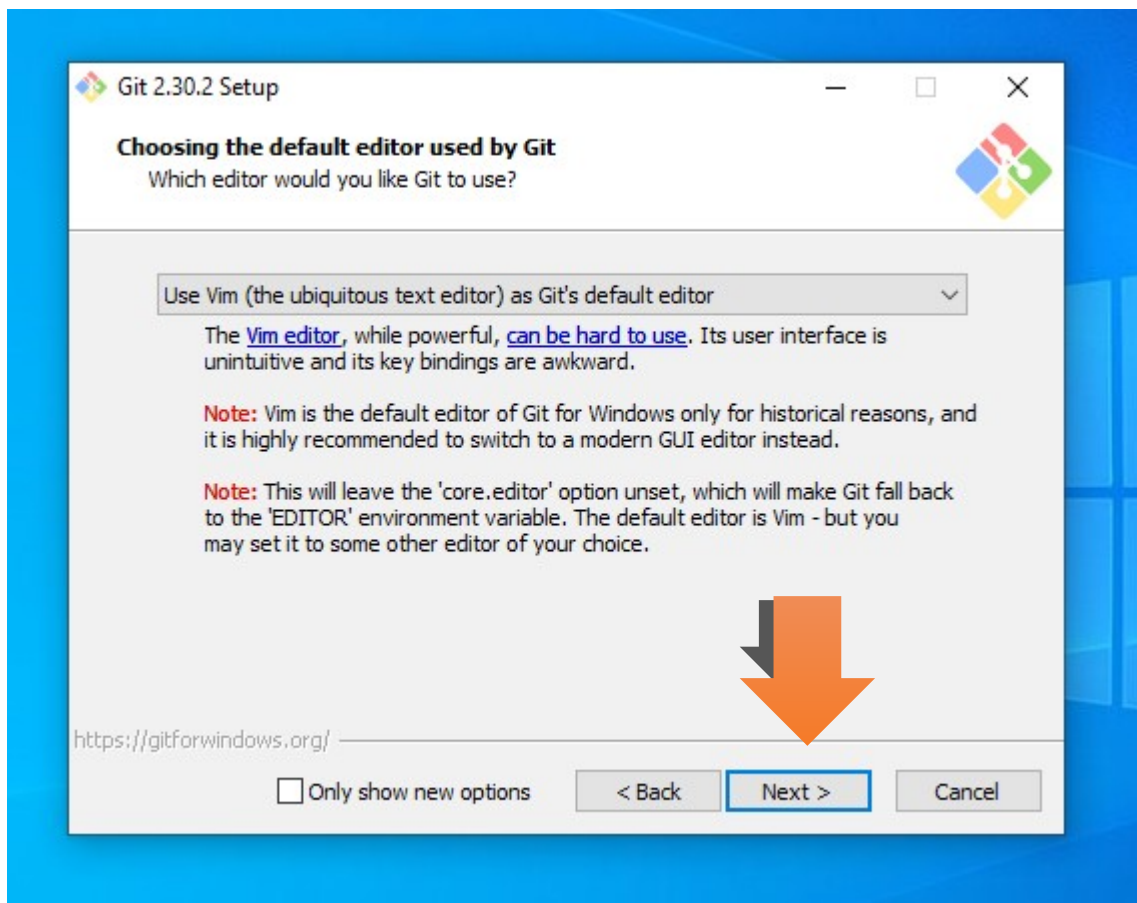


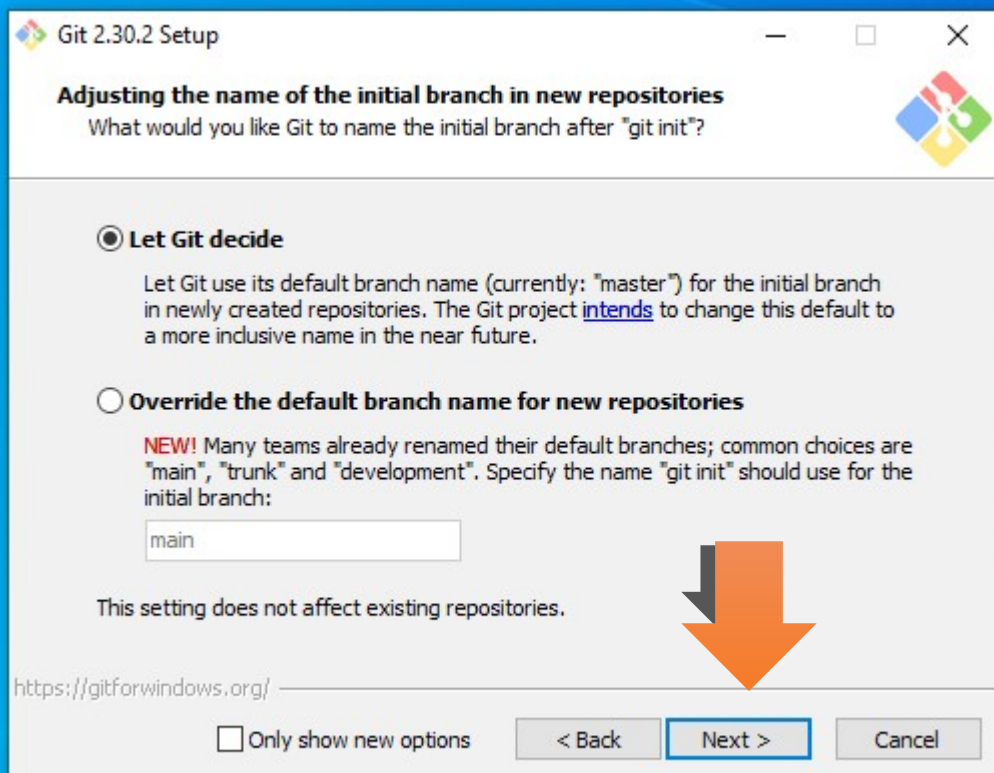
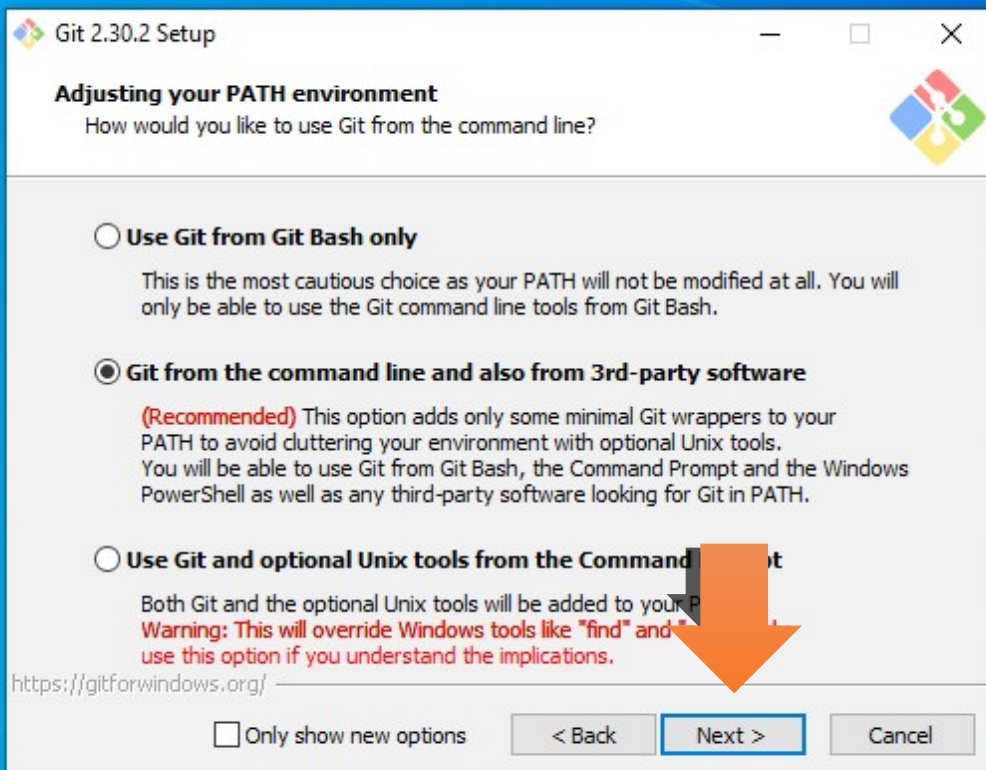
Abrimos el archivo que se encuentra de **Descargas** para poder hacer la instalación correspondiente de Git.



Una vez nos haya abierto la ventana de instalación, le damos **NEXT** a todas las ventanas que nos aparezcan, para no tener problemas más adelante.







Para ver que tengamos Git instalado en nuestro equipo, vamos a una consola de **CMD** y escribimos:

➤ git --version

```
Microsoft Windows [Versión 10.0.19042.804]
(c) 2020 Microsoft Corporation. Todos los derechos reservados.

C:\Users\EQUIPO>git --version
git version 2.30.2.windows.1

C:\Users\EQUIPO>
```

Paso 3: INSTALAR ADONISJS

Para instalar AdonisJs en nuestro basta con copiar el siguiente comando y pegarlo en la terminal de windows (**cmd**)

➤ npm i -g @adonisjs/cli

```
Microsoft Windows [Versión 10.0.19042.804]
(c) 2020 Microsoft Corporation. Todos los derechos reservados.

C:\Users\EQUIPO>adonis --version
4.0.12

C:\Users\EQUIPO>
```

Paso 4: CREAMOS UN PROYECTO ADONISJS CON TYPESCRIPT

Para crear un proyecto de AdonisJs con typescript, escribimos el siguiente comando:

➤ npm init adonis-ts-app <nombre del proyecto>

ignorando los signos de menor y mayor, creamos una carpeta donde alojemos el proyecto, en este caso esta en la carpeta escritorio (Desktop) y una carpeta creada en AdonisJs.

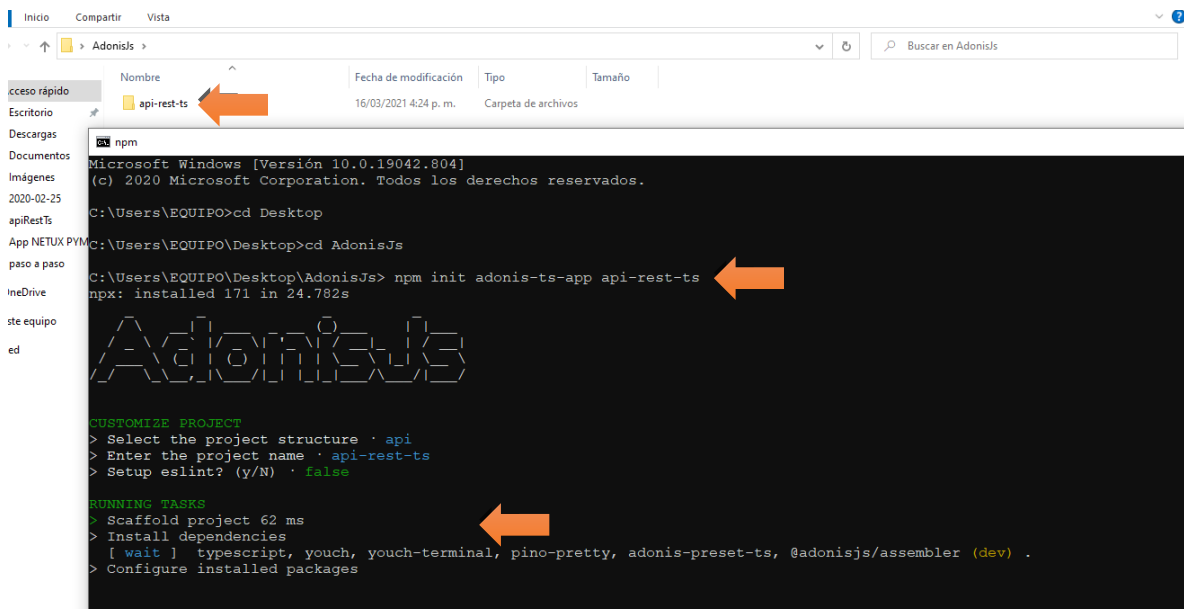
```
C:\Users\EQUIPO>cd Desktop
C:\Users\EQUIPO\Desktop>cd AdonisJs
C:\Users\EQUIPO\Desktop\AdonisJs> npm init adonis-ts-app api-rest-ts
```

Como vamos a crear una api-rest, escogemos la opción **API Server**.

```
AdonisJS

CUSTOMIZE PROJECT
> Select the project structure ... Press <ENTER> to select
> API Server
Web Application
```

Le damos **Enter** para el nombre, Para la opción de **Setup eslint** presionamos **N**, esperamos a que la descarga del proyecto se termine.



Paso 5: INSTALACIÓN DE ORM Y DEPENDENCIAS PARA LA BASE DE DATOS.


```
[ success ] Project created successfully

Run following commands to get started

> cd api-rest-ts
> node ace serve --watch

C:\Users\EQUIPO\Desktop\AdonisJs>
```

Como podemos observar, entramos a la carpeta usando los comando que nos imprime por la terminal.

➤ cd api-rest-ts

```
C:\Users\EQUIPO\Desktop\AdonisJs>cd api-rest-ts
```

Escribimos el siguiente comando para la instalación del ORM que vamos a usar en AdonisJS.

➤ npm i @adonisjs/lucid@alpha

```
C:\Users\EQUIPO\Desktop\AdonisJs\api-rest-ts> npm i @adonisjs/lucid@alpha
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@2.3.2 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@2.3.2: wanted {"
"os":"win32","arch":"x64"})

+ @adonisjs/lucid@10.0.0
added 49 packages from 63 contributors and audited 535 packages in 8.766s

40 packages are looking for funding
  run `npm fund` for details

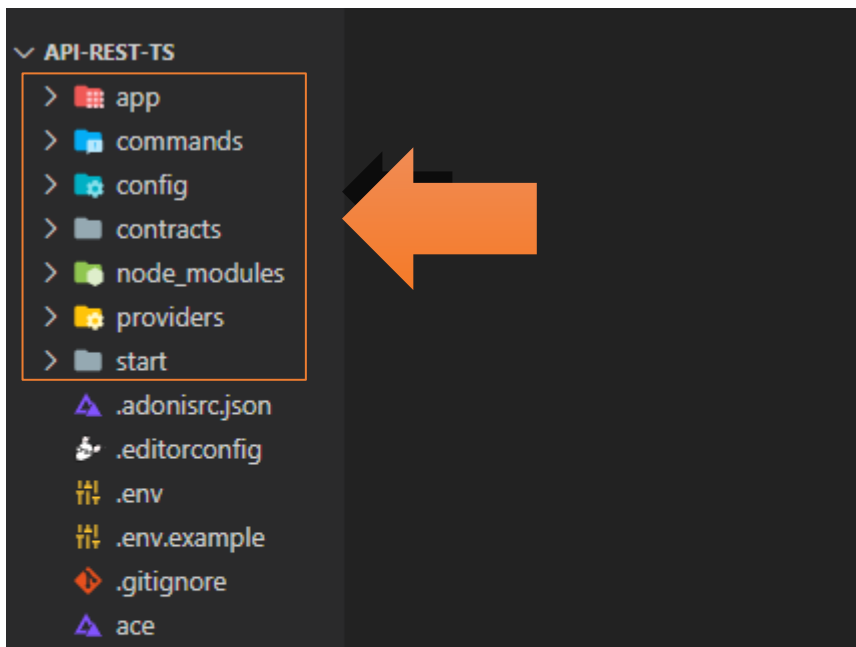
found 1 moderate severity vulnerability
  run `npm audit fix` to fix them, or `npm audit` for details

C:\Users\EQUIPO\Desktop\AdonisJs\api-rest-ts>
```

Instalación completada con éxito.

Paso 6: PROGRAMACIÓN DE API REST.

Para programar la api, vamos a usar **Vs Code**, pero también son libres de usar cualquier otro editor de código, ya sea Sublime Text 3, VIM, WebStorm, Atom.

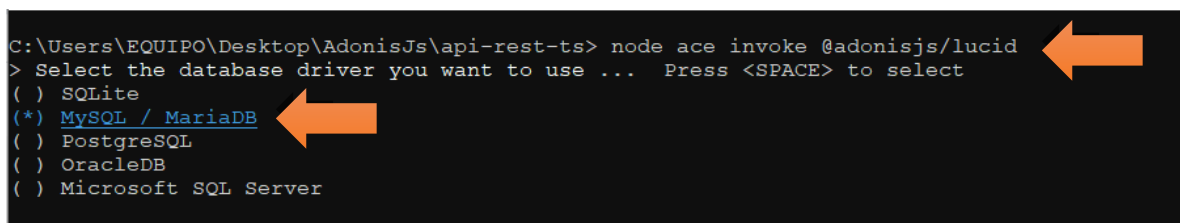


Como podemos observar, no tenemos la carpeta llamada **database** por lo que no podemos hacer migración a nuestra base de datos. Para eso tenemos que ingresar un comando para poder hacer migraciones, crear modelos etc.

➤ `node ace invoke @adonisjs/lucid`

Escribimos el comando y damos Enter, escogemos Mysql / MariaDB, pero si quieres usar otra base de datos también es válido, solo cambiarán los parámetros de configuración en el archivo .env

Para seleccionar la base de datos presionamos barra espaciadora y damos Enter.



A continuación, esperamos a que la instalación se termine de completar y damos Enter a la opción **In the terminal** para mostrar la información por la terminal.

```

C:\Users\EQUIPO\Desktop\AdonisJs\api-rest-ts> node ace invoke @adonisjs/lucid
> Select the database driver you want to use · mysql
CREATE: config/database.ts
UPDATE: .env,.env.example
[ wait ] Installing: luxon, mysql ..
CREATE: database\factories\index.ts
UPDATE: tsconfig.json { types += "@adonisjs/lucid" }
UPDATE: .adonisrc.json { commands += "@adonisjs/lucid/build/commands" }
UPDATE: .adonisrc.json { providers += "@adonisjs/lucid" }
[ info ] The package wants to display readable instructions for the setup
> Select where to display instructions ... Press <ENTER> to select
  In the browser
> In the terminal

```

```

## Variables for the MSSQL driver

MSSQL_SERVER: Env.schema.string({ format: 'host' }),
MSSQL_PORT: Env.schema.number(),
MSSQL_USER: Env.schema.string(),
MSSQL_PASSWORD: Env.schema.string.optional(),
MSSQL_DB_NAME: Env.schema.string(),

## Variables for the OracleDB driver

ORACLE_HOST: Env.schema.string({ format: 'host' }),
ORACLE_PORT: Env.schema.number(),
ORACLE_USER: Env.schema.string(),
ORACLE_PASSWORD: Env.schema.string.optional(),
ORACLE_DB_NAME: Env.schema.string(),
CREATE: ace-manifest.json file
C:\Users\EQUIPO\Desktop\AdonisJs\api-rest-ts>

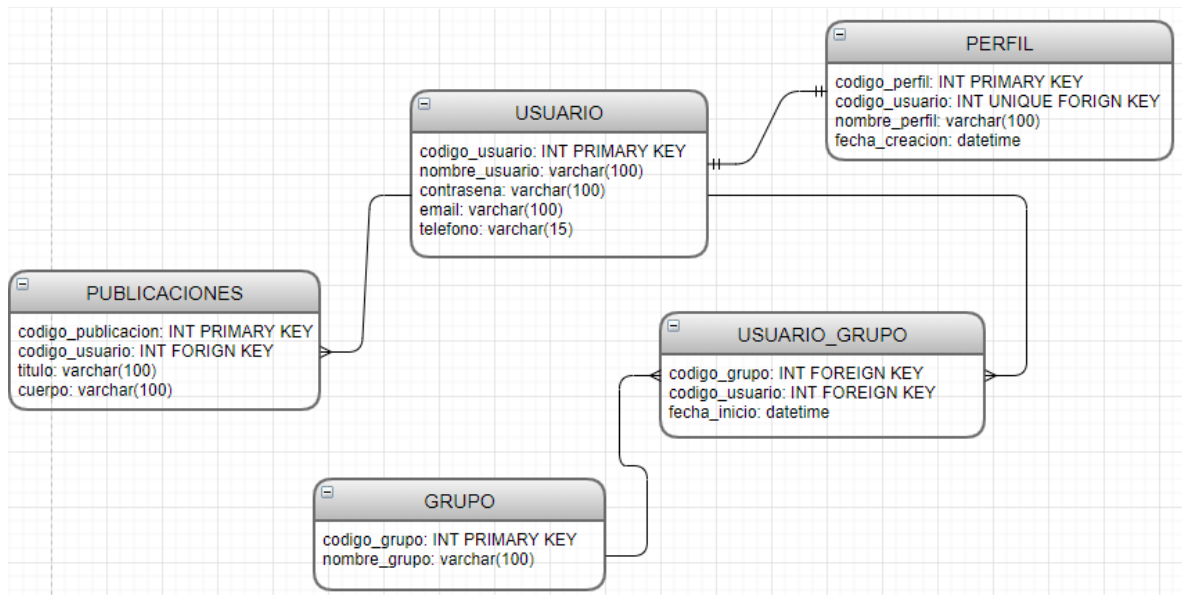
```

Ahora como podemos observar, si nos muestra la carpeta database



Paso 6: CREACIÓN DE MODELO RELACIONAL PARA MIGRAR A LA BASE DE DATOS.

Para este ejemplo vamos a usar el siguiente modelo entidad relación:

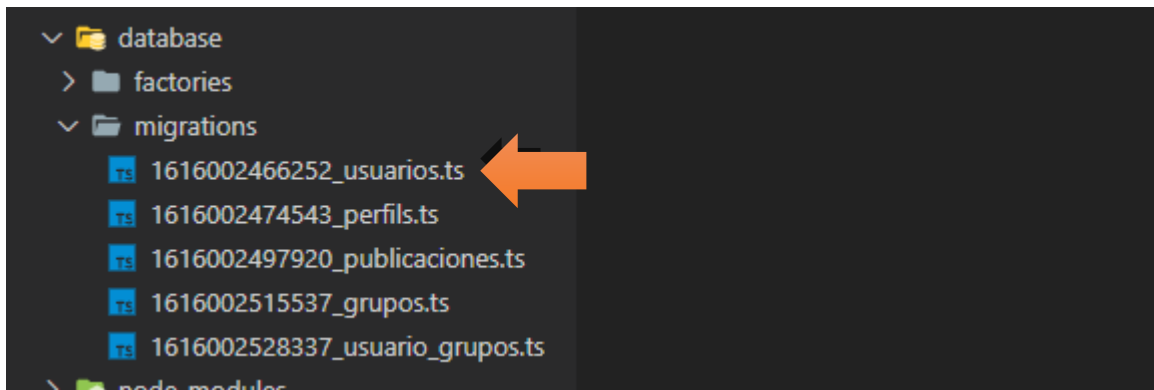


En este modelo, vemos que un usuario tiene 1 perfil (1:1), un usuario tiene muchas publicaciones (1:N) y que un usuario pertenece a muchos grupos y un grupo puede tener muchos usuarios (N:M)

Para crear cada una de estas tablas vamos a terminal y dentro del proyecto, escribimos:

- `node ace make:migration Usuario`
- `node ace make:migration Perfil`
- `node ace make:migration Publicaciones`
- `node ace make:migration Grupo`
- `node ace make:migration usuario_grupo`

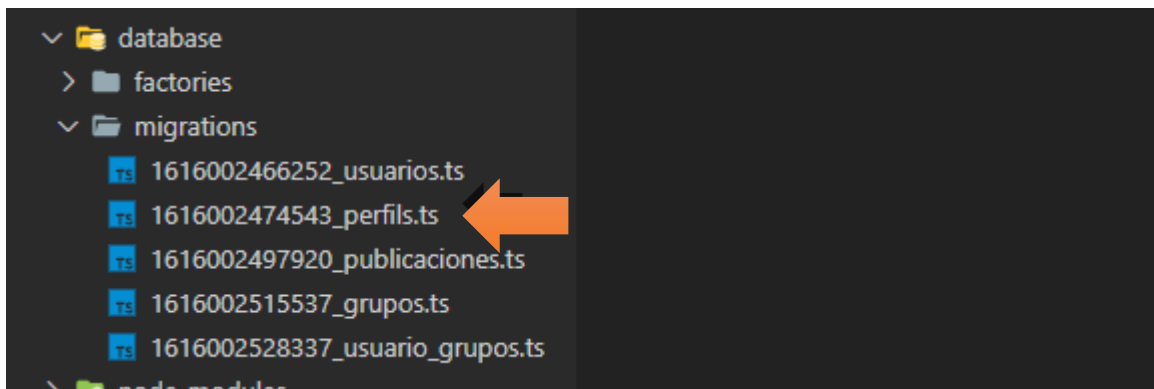
Una vez haya generado todas migraciones, nos dirigimos a la carpeta database y realizamos las relaciones e inserción de campos.



Abrimos la tabla usuarios.ts y lo modificamos de la siguiente forma:

```
export default class Usuarios extends BaseSchema {  
  protected tableName = 'usuarios'  
  
  public async up () {  
    this.schema.createTable(this.tableName, (table) => {  
      table.integer('codigo_usuario').primary().unsigned() /* codigo_usuario llave primaria */  
      table.string('nombre_usuario', 100).notNullable()  
      table.string('contrasena', 100).notNullable()  
      table.string('email', 100).notNullable()  
      table.string('telefono', 15).notNullable()  
      table.timestamps(false)  
    })  
  }  
}
```

Abrimos la tabla perfiles.ts

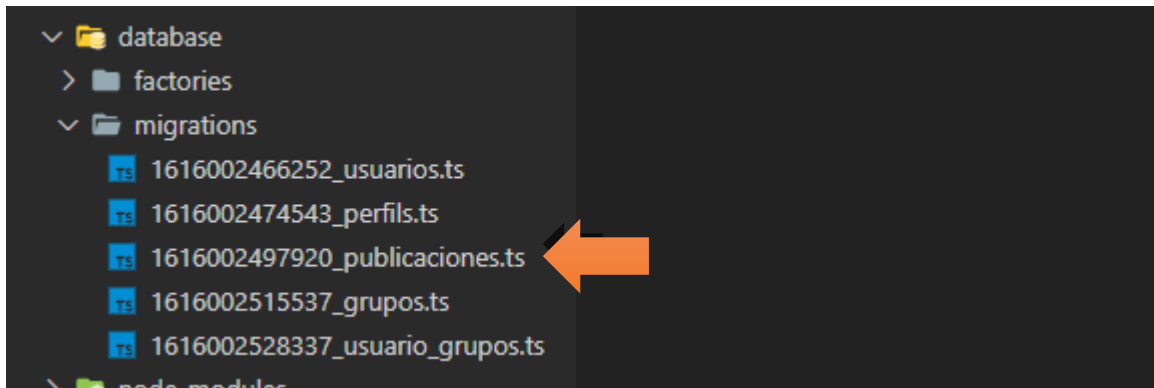


Hacemos la siguiente modificación de la tabla:


```
export default class Perfil extends BaseSchema {
  protected tableName = 'perfiles'

  public async up () {
    this.schema.createTable(this.tableName, (table) => {
      table.integer('codigo_perfil').primary().unsigned()
      table.string('nombre_perfil', 100).notNullable()
      table.date('fecha_creacion').notNullable()
      table.integer('codigo_usuario').unsigned().unique().index('codigo_usuario')
      table.foreign('codigo_usuario').references('usuarios.codigo_usuario').onDelete('cascade')
      table.timestamps(false)
    })
  }
}
```

Abrimos la tabla publicaciones.ts

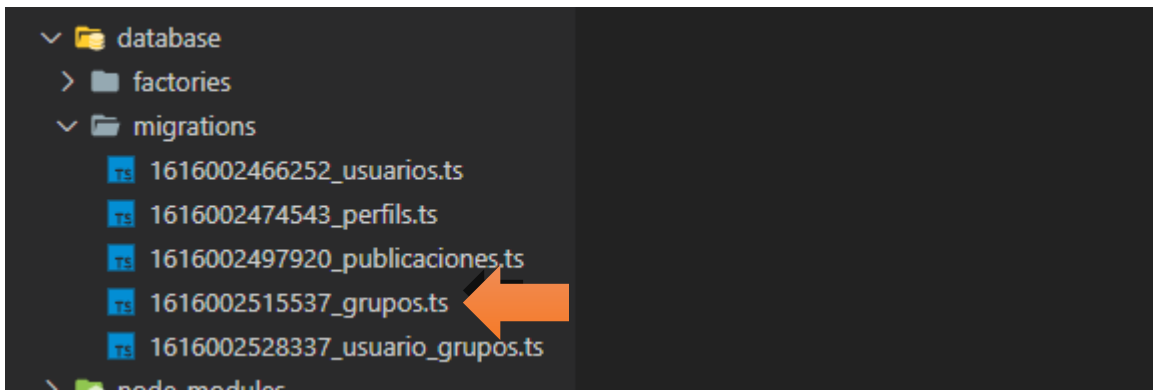


Hacemos la siguiente modificación de la tabla:

```
export default class Publicaciones extends BaseSchema {
  protected tableName = 'publicaciones'

  public async up () {
    this.schema.createTable(this.tableName, (table) => {
      table.integer('codigo_publicacion').primary().unsigned()
      table.string('titulo', 100).notNullable()
      table.string('cuerpo', 200).notNullable()
      table.integer('codigo_usuario').unsigned().index('codigo_usuario')
      table.foreign('codigo_usuario').references('usuarios.codigo_usuario').onDelete('cascade')
      table.timestamps(false)
    })
  }
}
```

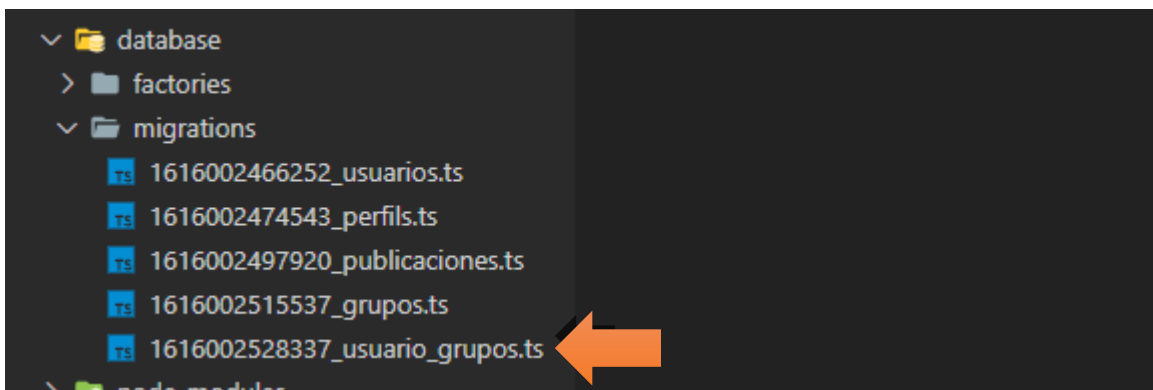
Abrimos la tabla grupos.ts



Hacemos la siguiente modificación de la tabla:

```
export default class Grupos extends BaseSchema {  
  protected tableName = 'grupos'  
  
  public async up () {  
    this.schema.createTable(this.tableName, (table) => {  
      table.integer('codigo_grupo').primary().unsigned()  
      table.string('nombre_grupo', 100).notNullable()  
      table.timestamps(false)  
    })  
  }  
}
```

Abrimos la tabla grupos.ts



Hacemos la siguiente modificación de la tabla:

```
export default class UsuarioGrupos extends BaseSchema {  
  protected tableName = 'usuario_grupos'  
  
  public async up () {  
    this.schema.createTable(this.tableName, (table) => {  
      table.integer('codigo_usuario').unsigned().index('codigo_usuario')  
      table.integer('codigo_grupo').unsigned().index('codigo_grupo')  
      table.date('fecha_inicio').notNullable()  
      table.foreign('codigo_usuario').references('usuarios.codigo_usuario').onDelete('cascade')  
      table.foreign('codigo_grupo').references('grupos.codigo_grupo').onDelete('cascade')  
      table.timestamps(false)  
    })  
  }  
}
```

Paso 7: MIGRAMOS NUESTRAS TABLAS AL MOTOR DE BASE DE DATOS.

Antes de migrar nuestras tablas a la base de datos, vamos a configurar nuestro archivo .env



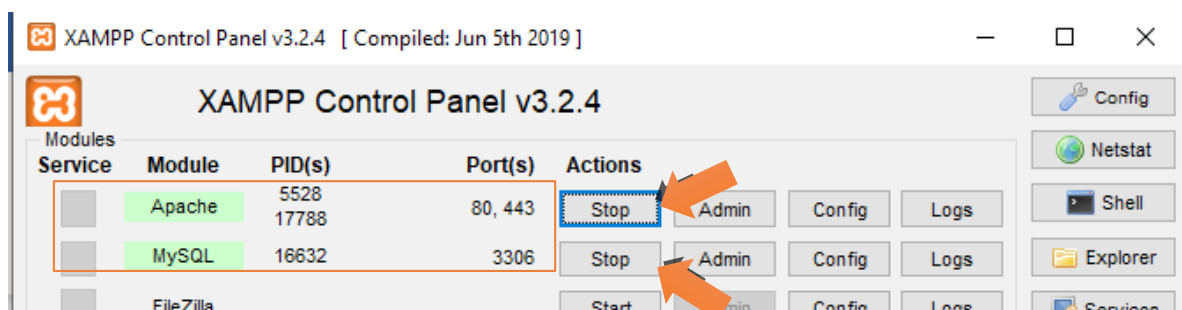
Debemos dejar configurado el archivo .env de la siguiente forma:

```
PORT=3333
HOST=127.0.0.1
NODE_ENV=development
APP_KEY=YNb0_BYhSNBr2-SIROu0cDeqg9shLSc6
DB_CONNECTION=mysql
MYSQL_HOST=localhost
MYSQL_PORT=3306
MYSQL_USER=root
MYSQL_PASSWORD=
MYSQL_DB_NAME=adonis
```

Ojo que la base de datos se debe llamar **adonis o nombrarla según la configuración**, Entonces activamos nuestro motor de base de datos local, para eso abrimos xampp y activamos Apache y Mysql, Si tienes otro motor de base de datos corriendo también es válido.

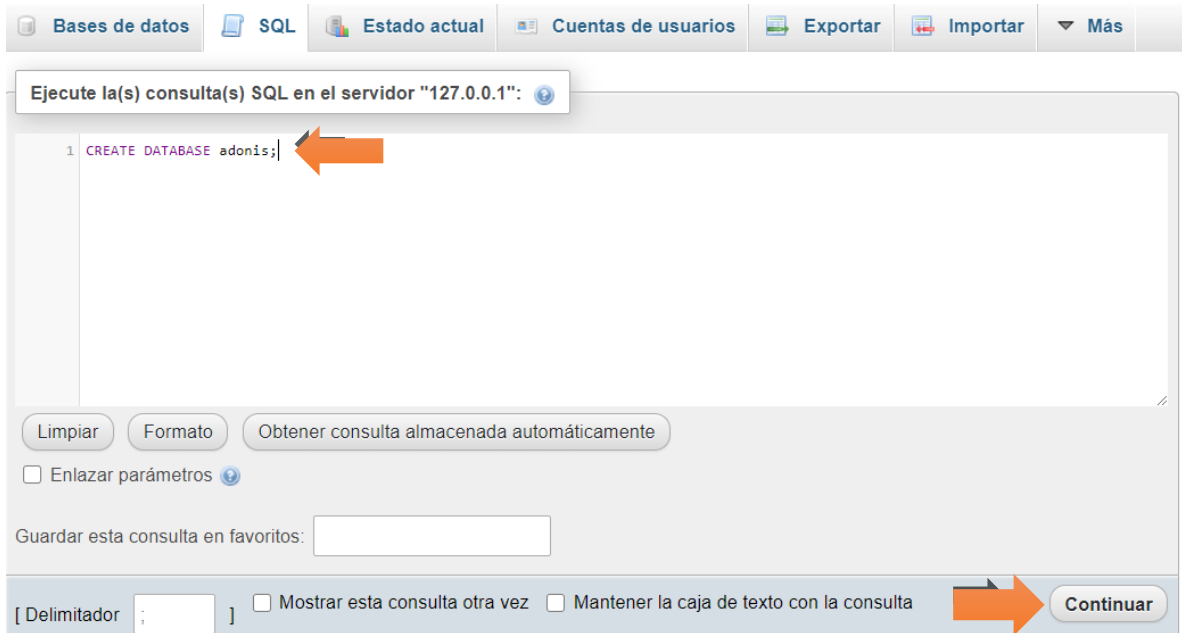
Si no tienen xampp instalado pueden descargarlo de la siguiente pagina:

- <https://www.apachefriends.org/es/download.html>



Ahora nos dirigimos a un navegador y escribimos:

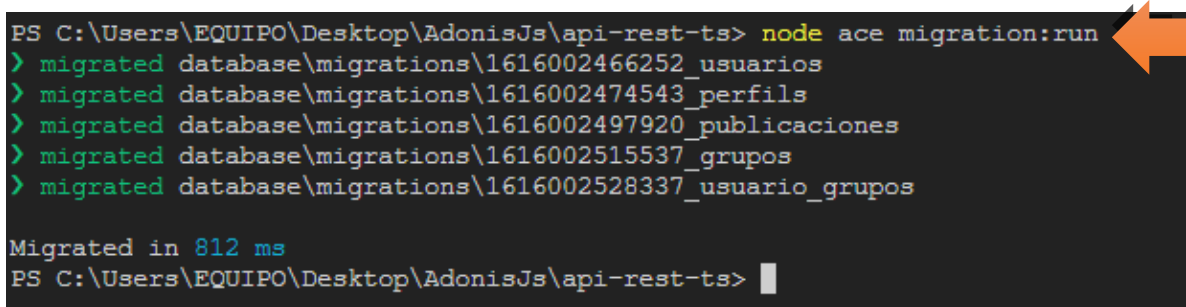
- <http://localhost/phpmyadmin/>



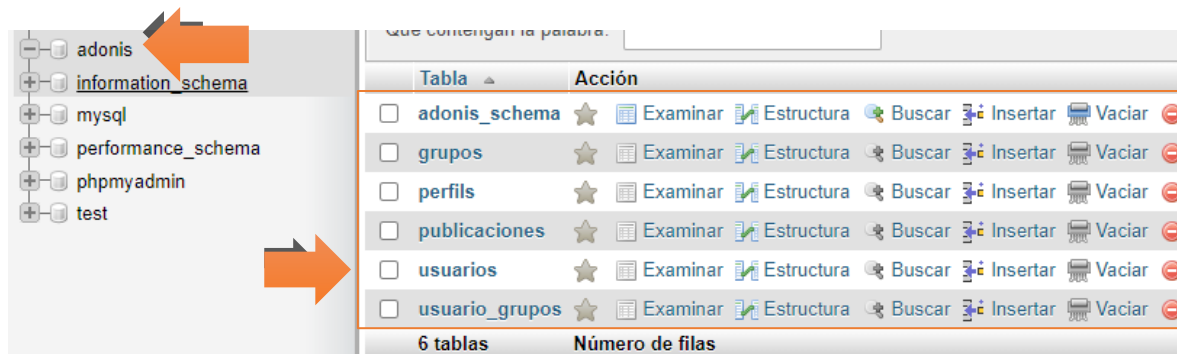
Una vez se haya creado la base de datos con éxito, procedemos a migrar nuestras tablas.

Para poder migrar nuestras tablas, escribimos el siguiente comando:

➤ `node ace migration:run`



Resultado:



Paso 8: CREACIÓN DE MODELOS.

Para la creación de nuestros modelos y poder usar el ORM Lucid de Adonis, vamos a escribir los siguientes comandos:

- node ace make:model Usuario
- node ace make:model Perfil
- node ace make:model Publicaciones
- node ace make:model Grupos
- node ace make:model usuario_grupos

```
PS C:\Users\EQUIPO\Desktop\AdonisJs\api-rest-ts> node ace make:model Usuario
CREATE: app\Models\Usuario.ts
PS C:\Users\EQUIPO\Desktop\AdonisJs\api-rest-ts> node ace make:model Perfil
CREATE: app\Models\Perfil.ts
PS C:\Users\EQUIPO\Desktop\AdonisJs\api-rest-ts> node ace make:model Publicaciones
CREATE: app\Models\Publicacione.ts
PS C:\Users\EQUIPO\Desktop\AdonisJs\api-rest-ts> node ace make:model Grupos
CREATE: app\Models\Grupo.ts
PS C:\Users\EQUIPO\Desktop\AdonisJs\api-rest-ts> 
```

Paso 9: CREACIÓN DE LOS CONTROLADORES.

Para la creación de los controladores escribimos los siguientes comandos:

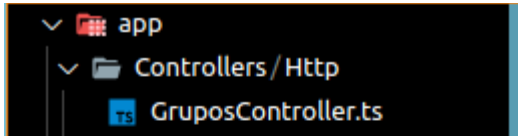
- node ace make:controller Usuario
- node ace make:controller Perfil
- node ace make:controller Publicaciones
- node ace make:controller Grupos
- node ace make:controller GrupoUsuario

A continuación escribimos la lógica de la API, OJO cada controlador contendrá un método para verificar que la llave

primaria no se repita.

Paso 10: CODIFICACIÓN:

Empezamos por escribir en **GRUPO**:



CONTROLADOR DE GRUPO:

```
import { HttpContextContract } from '@ioc:Adonis/Core/HttpContext'
import Grupo from 'App/Models/Grupo'

export default class GruposController {

  public async setRegistrarGrupo({request, response}: HttpContextContract) {
    try {
      const dataGrupo = request.only(['codigo_grupo', 'nombre_grupo'])
      const codigoGrupo = dataGrupo.codigo_grupo
      const codigoGrupoExistente = await this.getValidarGrupoExistente(codigoGrupo)
      if (codigoGrupoExistente === 0) {
        await Grupo.create(dataGrupo)
        response.status(200).json({"msg": "Grupo registrado con exito"})
      } else {
        response.status(400).json({"msg": "codigo del grupo ya se encuentra registrado!!"})
      }
    } catch (error) {
      console.log(error)
      response.status(500).json({"msg": "Error en el servidor !!"})
    }
  }

  private async getValidarGrupoExistente(codigo_grupo: Number): Promise<Number> {
    const total = await Grupo.query().where({"codigo_grupo":codigo_grupo}).count('*').from('grupos')
    return parseInt(total[0]["count(*)"])
  }
}
```

MODELO DE GRUPO:

```
import { DateTime } from 'luxon'
import { BaseModel, column } from '@ioc:Adonis/Lucid/Orm'

export default class Grupo extends BaseModel {

  @column({ isPrimary: true }) public codigo_grupo: number
  @column() public nombre_grupo: string

  @column.dateTime({ autoCreate: true })
  public createdAt: DateTime

  @column.dateTime({ autoCreate: true, autoUpdate: true })
  public updatedAt: DateTime
}
```

GRUPO USUARIO:

TS GrupoUsuariosController.ts

TS UsuarioGrupo.ts

CONTROLADOR GRUPO USUARIO:

```
import { HttpContextContract } from '@ioc:Adonis/Core/HttpContext'
import Grupo from 'App/Models/Grupo'
import Usuario from 'App/Models/Usuario'
import UsuarioGrupo from 'App/Models/UsuarioGrupo'

export default class GrupoUsuariosController {

  public async setRegistrarUsuarioGrupo({request, response}:HttpContextContract) {
    try {
      const dataGrupoUsuario = request.only(['codigo_usuario','codigo_grupo', 'fecha_inicio'])
      const codigoUsuario = dataGrupoUsuario.codigo_usuario
      const codigoGrupo = dataGrupoUsuario.codigo_grupo
      const datosExistentes: Number = await this.getValidarDatosGrupoYUsuario(codigoGrupo, codigoUsuario)
      switch (datosExistentes) {
        case 0:
          await UsuarioGrupo.create(dataGrupoUsuario)
          response.status(200).json({"msg": "Registro de usuario en grupo completo"})
          break;
        case 1:
          response.status(400).json({"msg": "El codigo del usuario no se encuentra registrado"})
          break;
        case 2:
          response.status(400).json({"msg": "El codigo del grupo no se encuentra registrado"})
          break;
      }
    } catch (error) {
      console.log(error)
      response.status(500).json({"msg": "Error en el servidor !!"})
    }
  }
}
```

MÉTODO DE VALIDACIÓN PARA GRUPO DE USUARIO:

```
private async getValidarDatosGrupoYUsuario(codigo_grupo: Number, codigo_usuario: Number): Promise<Number> {
  let total = await Grupo.query().where({"codigo_grupo":codigo_grupo}).count('*').from('grupos')
  let cantidadDatos = parseInt(total[0]["count(*)"])
  if (cantidadDatos !== 0) {
    total = await Usuario.query().where({"codigo_usuario":codigo_usuario}).count('*').from('usuarios')
    cantidadDatos = parseInt(total[0]["count(*)"])
    if (cantidadDatos !== 0) {
      return 0;
    } else {
      return 2; /* si el metodo retorna un 2, significa que el codigo de usuario no existe */
    }
  } else {
    return 1; /* si el metodo retorna un 1, significa que el codigo de grupo no existe */
  }
}
```

MODELO DE GRUPO DE USUARIO:

```
import { DateTime } from 'luxon'
import { BaseModel, column } from '@ioc:Adonis/Lucid/Orm'

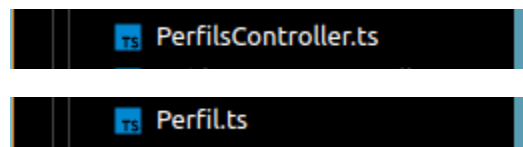
export default class UsuarioGrupo extends BaseModel {

  @column() public codigo_usuario: Number
  @column() public codigo_grupo: Number
  @column() public fecha_inicio: Date

  @column.dateTime({ autoCreate: true })
  public createdAt: DateTime

  @column.dateTime({ autoCreate: true, autoUpdate: true })
  public updatedAt: DateTime
}
```

PERFIL:



CONTROLADOR PERFIL:

```
import { HttpContextContract } from '@ioc:Adonis/Core/HttpContext'
import Perfil from 'App/Models/Perfil'

export default class PerfilsController {

  async setRegistrarPerfil({request, response}: HttpContextContract) {
    try {
      const dataPerfil = request.only([
        'codigo_perfil', 'codigo_usuario', 'nombre_perfil', 'fecha_creacion'
      ])
      const codigoPerfil = dataPerfil.codigo_perfil;
      const perfilExistente: Number = await this.getValidarPerfilExistente(codigoPerfil);
      if (perfilExistente === 0) {
        await Perfil.create(dataPerfil)
        response.status(200).json({"msg": "Registro de perfil completado"})
      } else {
        response.status(400).json({"msg": "Error, el codigo perfil ya se encuentra registrado"})
      }
    } catch (error) {
      response.status(500).json({"msg": "Error en el servidor !!"})
    }
  }

  private async getValidarPerfilExistente(codigo_perfil: Number): Promise<Number> {
    const total = await Perfil.query().where({"codigo_perfil": codigo_perfil}).count('*').from('perfils')
    return parseInt(total[0]["count(*)"])
  }
}
```

MODELO DEL PERFIL:

```
import { DateTime } from 'luxon'
import { BaseModel, column } from '@ioc:Adonis/Lucid/Orm'


export default class Perfil extends BaseModel {


  @column({ isPrimary: true }) public codigo_perfil: number
  @column() public codigo_usuario: number /* llave foranea */
  @column() public nombre_perfil: string
  @column() public fecha_creacion: Date

  @column.dateTime({ autoCreate: true })
  public createdAt: DateTime

  @column.dateTime({ autoCreate: true, autoUpdate: true })
  public updatedAt: DateTime
}
```

PUBLICACIONES:

 PublicacionesController.ts

 Publicacione.ts

CONTROLADOR PUBLICACIÓN:

```
import { HttpContextContract } from '@ioc:Adonis/Core/HttpContext'
import Publicaciones from 'App/Models/Publicacione'

export default class PublicacionesController {

  async setRegistroPublicacion({request, response}:HttpContextContract) {
    try {
      const dataPublicaciones = request.only([
        'codigo_publicacion', 'codigo_usuario', 'titulo', 'cuerpo'
      ])
      const codigoPublicacion = dataPublicaciones.codigo_publicacion;
      const codigoPublicacionExistente: Number = await this.getValidarPublicacionExistente(codigoPublicacion)
      if (codigoPublicacionExistente === 0) {
        await Publicaciones.create(dataPublicaciones)
        response.status(200).json({"msg": "Registro de publicacion completado con exito"})
      } else {
        response.status(400).json({"msg": "Error, el codigo publicacino ya se encuentra registrado"})
      }
    } catch (error) {
      response.status(500).json({"msg": "Error en el servidor !!"})
    }
  }

  private async getValidarPublicacionExistente(codigo_publicacion: Number): Promise<Number> {
    const total = await Publicaciones.query().where({"codigo_publicacion":codigo_publicacion}).count('*').from('publicaciones')
    return parseInt(total[0]["count(*)"])
  }
}
```

MODELO PUBLICACIÓN:


```
import { DateTime } from 'luxon'
import { BaseModel, column } from '@ioc:Adonis/Lucid/Orm'

export default class Publicacione extends BaseModel {

  @column({ isPrimary: true }) public codigo_publicacion: number
  @column() public codigo_usuario: number /* llave foranea */
  @column() public titulo: string
  @column() public cuerpo: string

  @column.dateTime({ autoCreate: true })
  public createdAt: DateTime

  @column.dateTime({ autoCreate: true, autoUpdate: true })
  public updatedAt: DateTime
}
```



USUARIOS:

Para el controlador de Usuarios vamos a definir ciertos métodos para listar la información, es decir, listar todos los usuarios, listar los usuarios que tengan un perfil, listar los usuarios que tengan publicaciones y que pertenezcan a uno o mas grupos.

Como primer paso vamos a definir el modelo de usuario:

1. Importamos los módulos necesarios.

```
import { DateTime } from 'luxon'
import { BaseModel, column, hasOne, HasOne, hasMany, HasMany, manyToMany, ManyToMany } from '@ioc:Adonis/Lucid/Orm'
```

2. Definimos los campos de usuario:

```
@column({ isPrimary: true }) public codigo_usuario: number
@column() public nombre_usuario: string
@column() public contrasena: string
@column() public email: string
@column() public telefono: string
```

3. Definimos la relación de usuario con Perfil (OneToOne):

```
/* relacion de 1:1 con Perfil */
@hasOne(() => Perfil, {
  |  localKey: 'codigo_usuario',
  |  foreignKey: 'codigo_usuario',
  | })
public perfil: HasOne<typeof Perfil>
```


4. Definimos la relación de usuario con publicación (OneToMany)


```
/* relacion de 1:n con Publicaciones */
@hasMany(() => Publicaciones, {
  |  localKey: 'codigo_usuario',
  |  foreignKey: 'codigo_usuario',
  |})
public publicaciones: HasMany<typeof Publicaciones>
```

Por ultimo definimos la relación de (ManyToMany)

```
/* relacion de n:m con Grupo */
@manyToMany(() => Grupo, {
  |  localKey: 'codigo_usuario', /* llave primaria de la tabla usuario */
  |  pivotForeignKey: 'codigo_usuario', /* llave foranea de la nueva tabla [ usuario_grupo ]*/
  |  relatedKey: 'codigo_grupo', /* llave primaria de la tabla grupo */
  |  pivotRelatedForeignKey: 'codigo_grupo', /* llave foranea de la nueva tabla [ usuario_grupo ]*/
  |  pivotTable: 'usuario_grupos', /* nombre de nueva tabla N:M [ ManyToMany ] */
  |})
public usuario_grupos: ManyToMany<typeof Grupo>
```

CONTROLADOR:

Para el controlador definimos una serie de métodos para listar cada uno de las relaciones.

 UsuariosController.ts

1. Importamos los módulos necesarios.

```
import { HttpContextContract } from '@ioc:Adonis/Core/HttpContext'
import Usuario from 'App/Models/Usuario'
```

2. Definimos el método para listar todos los usuarios.

```
public async getListarUsuarios(): Promise<Usuario[]> {
  |   const user = await Usuario.all()
  |   return user;
  |}
```

3. Definimos el método para listar los usuarios con su respectivo perfil.

```
public async getListarUsuariosYPerfil(): Promise<Usuario[]> {
  |   const user = await Usuario
  |   .query()
  |   .preload('perfil')
  |   return user;
  |}
```

4. Definimos el método para listar los usuarios con su respectivas publicaciones.

```
public async getListarUsuariosYPublicacion(): Promise<Usuario[]> {  
    const user = await Usuario  
        .query()  
        .preload('publicaciones')  
    return user;  
}
```

Por ultimo definimos el método para ver a que grupo pertenece dicho usuario.

```
public async getListarUsuariosGrupos(): Promise<Usuario[]> {  
    const user = await Usuario  
        .query()  
        .preload('usuario_grupos')  
    return user;  
}
```

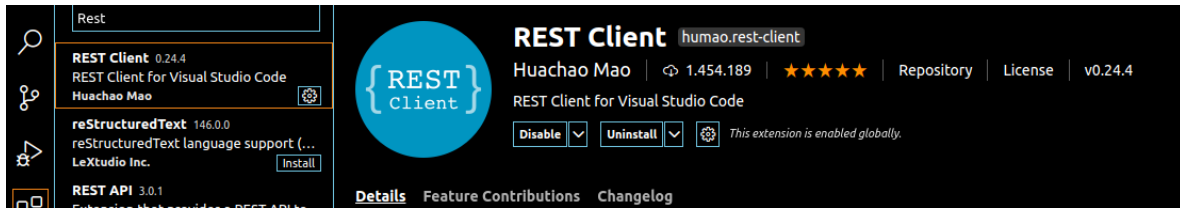
El siguiente método dentro del controlador de Usuarios es para la inserción de datos.

```
public async setRegistrarUsuarios({request, response}: HttpContextContract) {  
    const dataUsuario = request.only([  
        'codigo_usuario', 'nombre_usuario', 'contrasena',  
        'email', 'telefono', 'perfil'  
    ])  
    try {  
        const codigoUsuario = dataUsuario.codigo_usuario;  
        const usuarioExistente: Number = await this.getValidarUsuarioExistente(codigoUsuario)  
        if (usuarioExistente == 0) {  
            await Usuario.create(dataUsuario)  
            response.status(200).json({"msg": "Registro completado con exito"})  
        } else {  
            response.status(400).json({"msg": "Error, el codigo usuario ya se encuentra registrado"})  
        }  
    } catch (e) {  
        response.status(500).json({"msg": "Error en el servidor !!"})  
    }  
}
```

```
private async getValidarUsuarioExistente(codigo_usuario: Number): Promise<Number> {  
    const total = await Usuario.query().where({"codigo_usuario": codigo_usuario}).count('*').from('usuarios')  
    return parseInt(total[0]["count(*)"])  
}
```

Con esto completamos toda nuestra api, ahora lo que falta es llenar nuestra base de datos con datos de prueba.

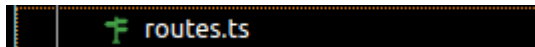
Para eso podemos usar postman o podemos usar una extensión que nos brinda VS Code llamada Rest Client.



Pueden instalar RET Client o pueden usar postman, Una vez hecho eso procedemos a definir las ruta, en este ejemplo solo se listara y se registrar la información.

RUTAS:

Nos dirigimos a la carpeta ***start/routes.ts***



Una vez dentro definimos las rutas:

```
Route.group(() => {  
  
  Route.get('/listar-usuarios', 'UsuariosController.getListarUsuarios')  
  Route.get('/listar-todo', 'UsuariosController.getListarUsuariosTodos')  
  Route.get('/listar-perfil', 'UsuariosController.getListarUsuariosYPerfil')  
  Route.get('/listar-publicaciones', 'UsuariosController.getListarUsuariosYPublicacion')  
  Route.get('/listar-usuarios-grupos', 'UsuariosController.getListarUsuariosGrupos')  
  
  Route.post('/registro-usuarios', 'UsuariosController.setRegistrarUsuarios')  
  Route.post('/registro-perfil', 'PerfilesController.setRegistrarPerfil')  
  Route.post('/registro-publicacion', 'PublicacionesController.setRegistroPublicacion')  
  Route.post('/registro-grupo', 'GruposController.setRegistrarGrupo')  
  Route.post('/registro-usuario-grupo', 'GrupoUsuariosController.setRegistrarUsuarioGrupo')  
  
}).prefix('/alcaldia')
```

Para poder acceder a los métodos que se definieron en los controladores, escribimos el nombre de la clase del controlador que vayamos a usar, por ejemplo, si queremos listar los usuarios seria:

<Nombre de la clase del controlador>.<Nombre del metodo>

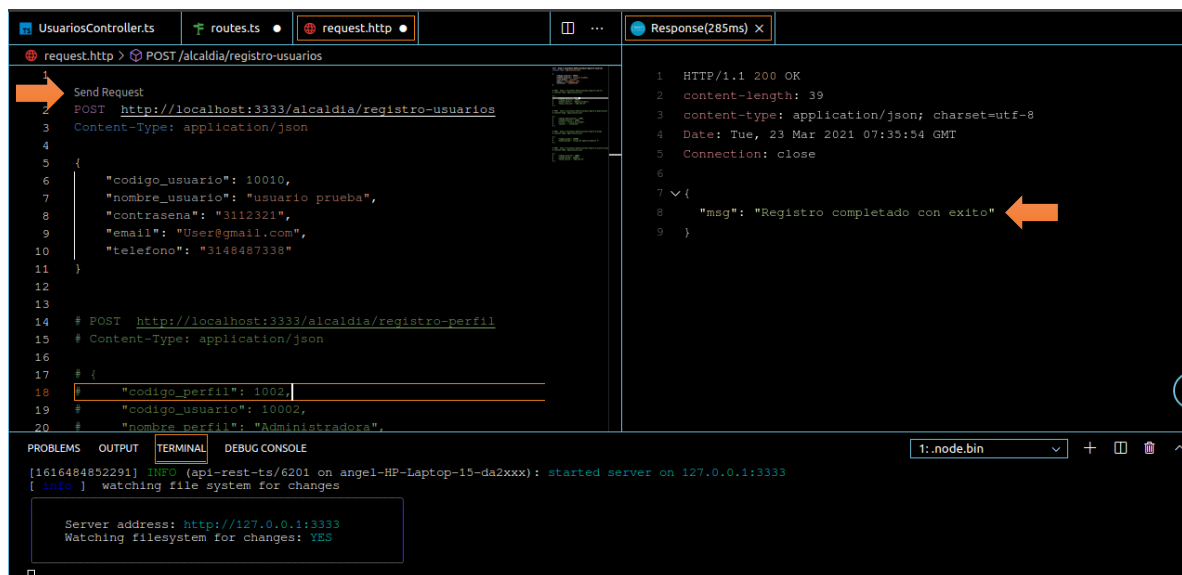
UsuariosController.getListarUsuarios

Ejemplo:

Para poder usar REST Client en Vs code, creamos un archivo llamado **request.http**



Ahora para poder hacer un registro escribimos nuestra ruta de la siguiente forma:



Verificamos que se haya registrado en nuestra base de datos:

Ahora vamos a un navegador, si tienes Google Chrome se debe instalar un extensión llamada JSON.

Escribimos en **localhost:3333/alcaldia/listar-usuarios**



