VALLE GRANDE

DESARROLLADOR CON

Java | Spring Boot | Python

ESTÁNDARES A CONSIDERAR

CONTENIDO

CREACIÓN DE UN PROYECTO JAVA

Los proyectos se deben crear utilizando Maven para lo cual deben considerar los siguientes datos:

Group Id:	pe.edu.vallegrande
Artifact Id:	<Nombre del proyecto>

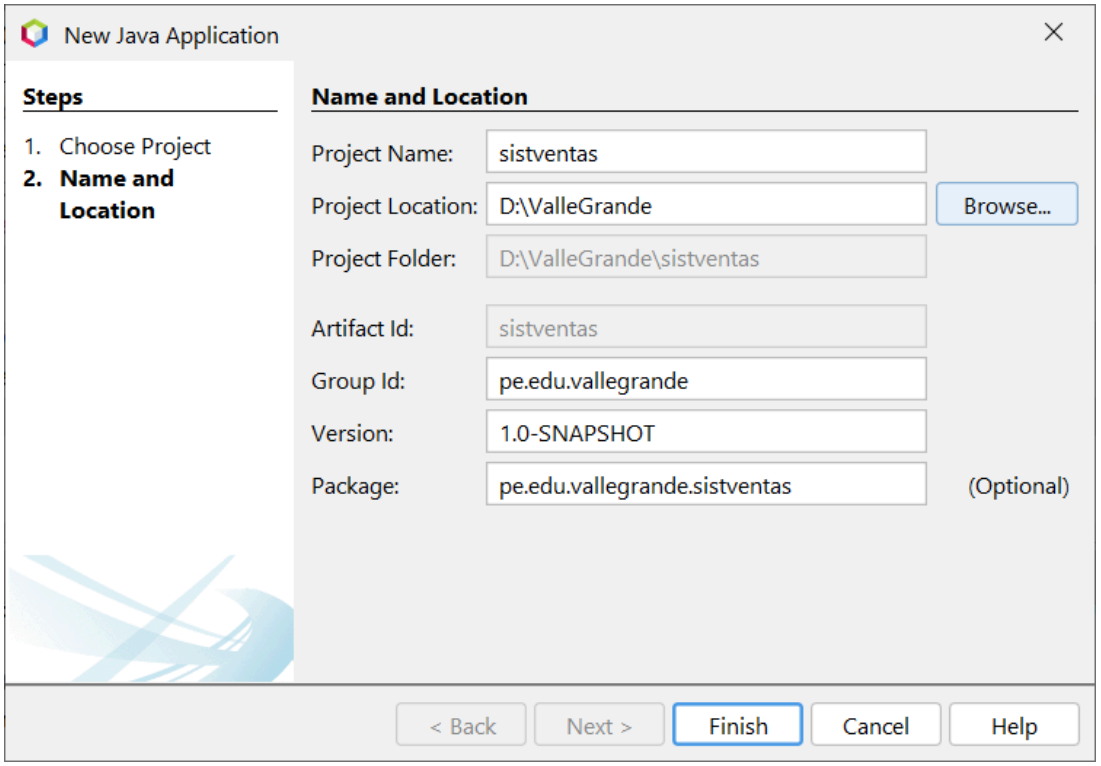
Bajo este contexto el paquete base sería:

pe.edu.vallegrande.<Nombre del proyecto>

Por ejemplo, si vamos a desarrollar un sistema de ventas (SistVentas) tenemos:

Group Id:	pe.edu.vallegrande
Artifact Id:	sistventas
Paquete base:	pe.edu.vallegrande.sistventas

A continuación, se tiene un ejemplo con NetBeans:



VALLE GRANDE

PAQUETERÍA EN JAVA

Propuesta (Java SE | Java EE)

Las soluciones deben plantearse bajo el enfoque de servicios, por lo que capas lógicas de una aplicación java deben considerar como mínimo las siguientes capas:

view	Para las interfaces de usuario, específicamente los formularios Swing para aplicaciones de escritorio.
service	Para la lógica de la solución. Podría contener 2 paquetes uno para la especificación y otro para la implementación.
controller	Para las clases controladoras, en caso web serían los servlets.
dto	Para los objetos de transferencia de datos.
model	Para las clases de dominio en caso se utilicen frameworks ORM.
dao	Para las clases correspondientes a la lógica de persistencia.
util	Para clases utilitarias.
exception	Para excepciones personalizadas.
db	Se coloca la clase **AccessDB.java** de la conexión a la base de datos.
prueba	Van las clases de los casos de prueba.

Por lo tanto, finalmente los paquetes deben quedar de la siguiente manera:

view	pe.edu.vallegrande.<Nombre del proyecto>.view Las clases deben tener como sufijo view, por ejemplo, VentasView Para el CRUD de cliente serian 2 formularios: ClientCrudView y ClientEditView
service	pe.edu.vallegrande.<Nombre del proyecto>.service Para el CRUD: ClientCrudService
controller	pe.edu.vallegrande.<Nombre del proyecto>.controller Para el CRUD: ClientCrudController
dto	pe.edu.vallegrande.<Nombre del proyecto>.dto Para el CRUD: ClientDto
model	pe.edu.vallegrande.<Nombre del proyecto>.model

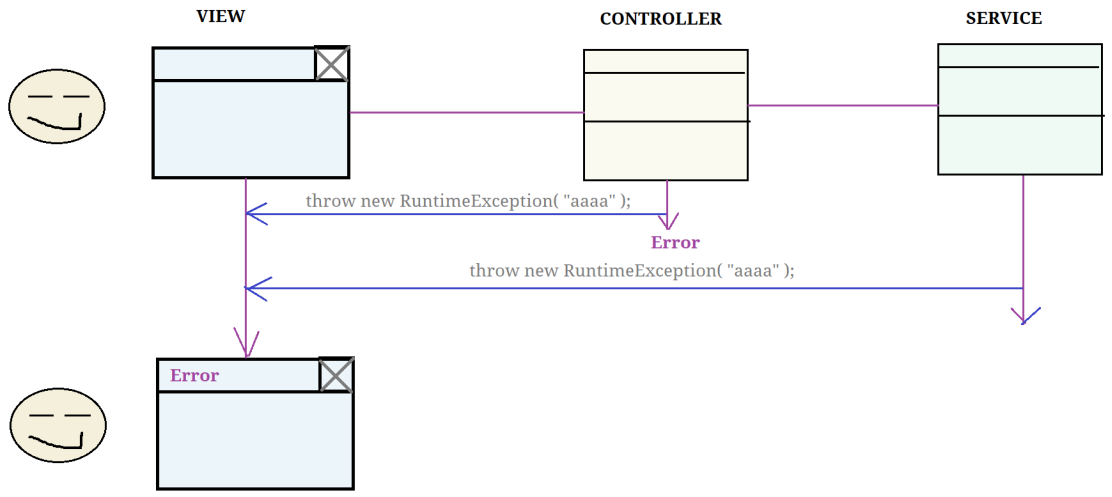
dao	pe.edu.vallegrande.<Nombre del proyecto>.dao
util	pe.edu.vallegrande.<Nombre del proyecto>.util
exception	pe.edu.vallegrande.<Nombre del proyecto>.excepciones

En caso de ser necesario un nuevo paquete, por ejemplo, un paquete para clases particulares del proyecto como la conexión con la base de datos (db): o casos de prueba (prueba) o pruebas unitarias (test), entre otras.

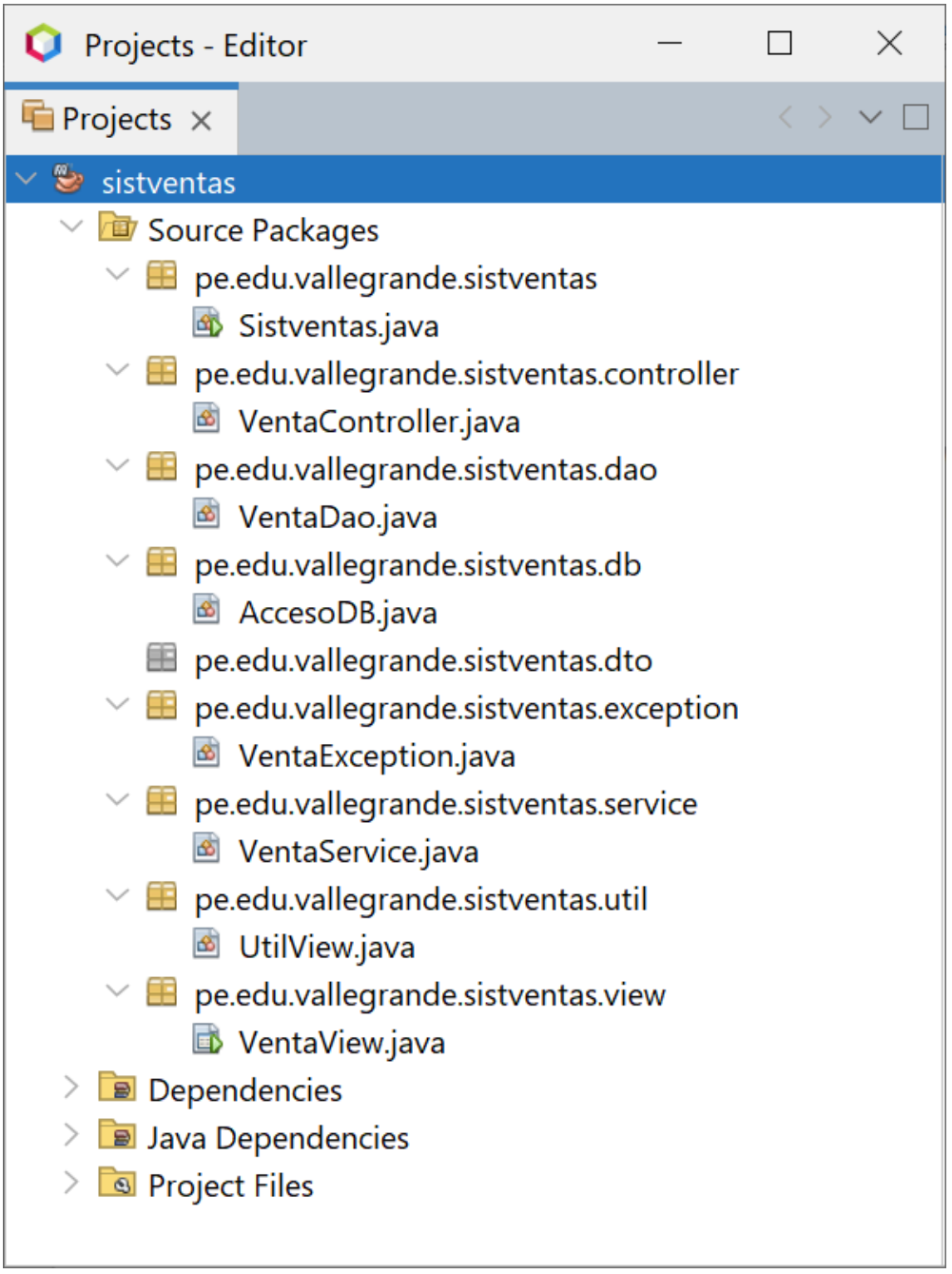
pe.edu.vallegrande.<Nombre del proyecto>.db

De esa manera se pueden incluir nuevos paquetes.

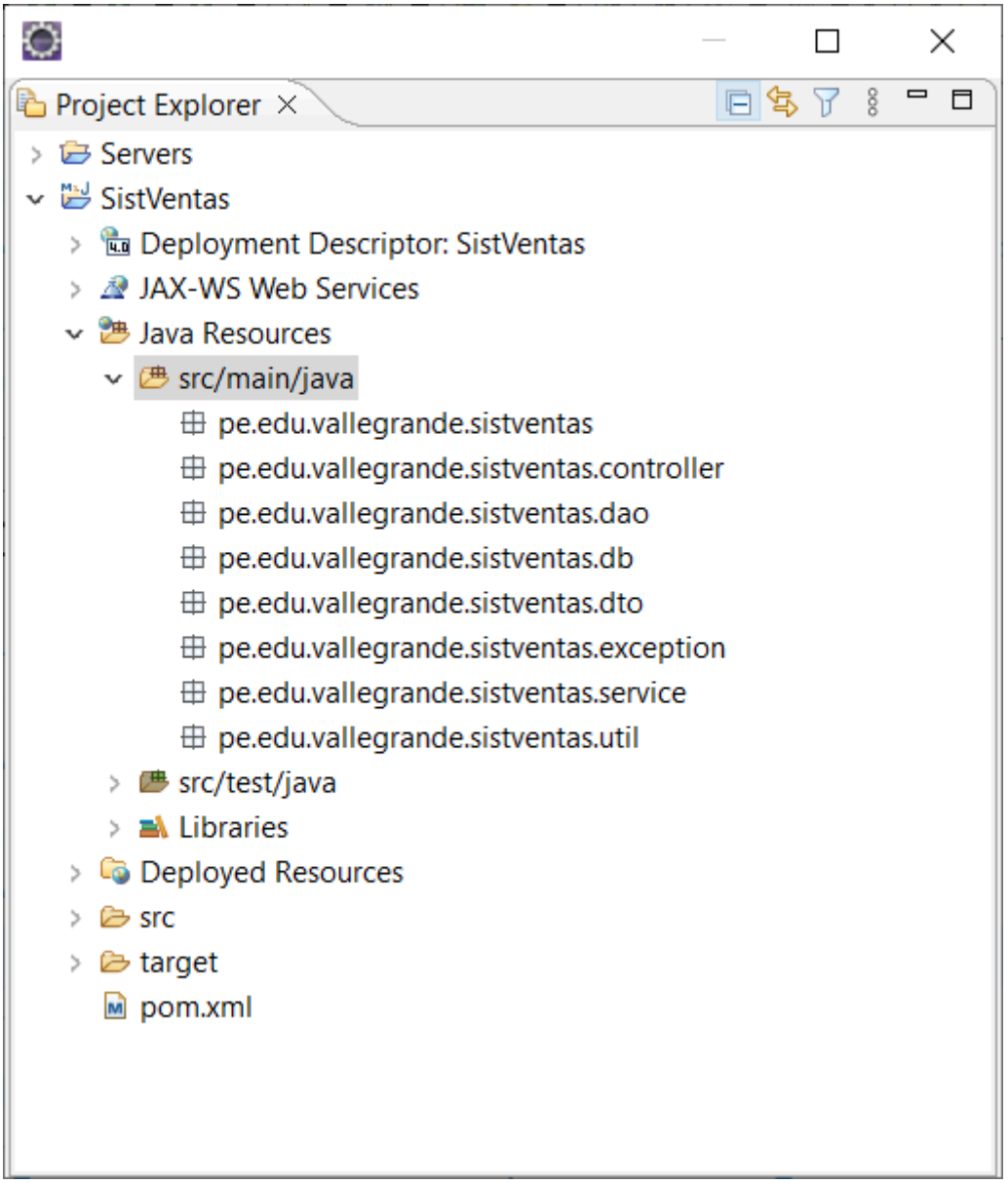
ARQUITECTURA EN CAPAS



Ejemplo 1


Projects - Editor
Projects
sistventas
Source Packages
pe.edu.vallegrande.sistventas
Sistventas.java
pe.edu.vallegrande.sistventas.controller
VentaController.java
pe.edu.vallegrande.sistventas.dao
VentaDao.java
pe.edu.vallegrande.sistventas.db
AccesoDB.java
pe.edu.vallegrande.sistventas.dto
pe.edu.vallegrande.sistventas.exception
VentaException.java
pe.edu.vallegrande.sistventas.service
VentaService.java
pe.edu.vallegrande.sistventas.util
UtilView.java
pe.edu.vallegrande.sistventas.view
VentaView.java
Dependencies
Java Dependencies
Project Files

Ejemplo 2


Project Explorer
Servers
SistVentas
Deployment Descriptor: SistVentas
JAX-WS Web Services
Java Resources
src/main/java
pe.edu.vallegrande.sistventas
pe.edu.vallegrande.sistventas.controller
pe.edu.vallegrande.sistventas.dao
pe.edu.vallegrande.sistventas.db
pe.edu.vallegrande.sistventas.dto
pe.edu.vallegrande.sistventas.exception
pe.edu.vallegrande.sistventas.service
pe.edu.vallegrande.sistventas.util
src/test/java
Libraries
Deployed Resources
src
target
pom.xml

PAQUETERÍA PARA SPRING BOOT

Propuesta

Para una aplicación web o de servicios REST (API REST) con Spring Boot se debe considerar los siguientes paquetes según lo plantea el framework:

controller	Para aplicaciones web con Spring Boot.
rest	Para los servicios REST o End Point.
dto	Para los objetos de transferencia de datos.
model	Para las clases de dominio en caso se utilicen frameworks ORM.
service	Para la lógica de la solución. Podría contener 2 paquetes uno para la especificación y otro para la implementación.
repository	Para las interfaces de acceso a las fuentes de datos.
util	Para clases utilitarias.
exception	Para excepciones personalizadas.

Por lo tanto, finalmente los paquetes deben quedar de la siguiente manera:

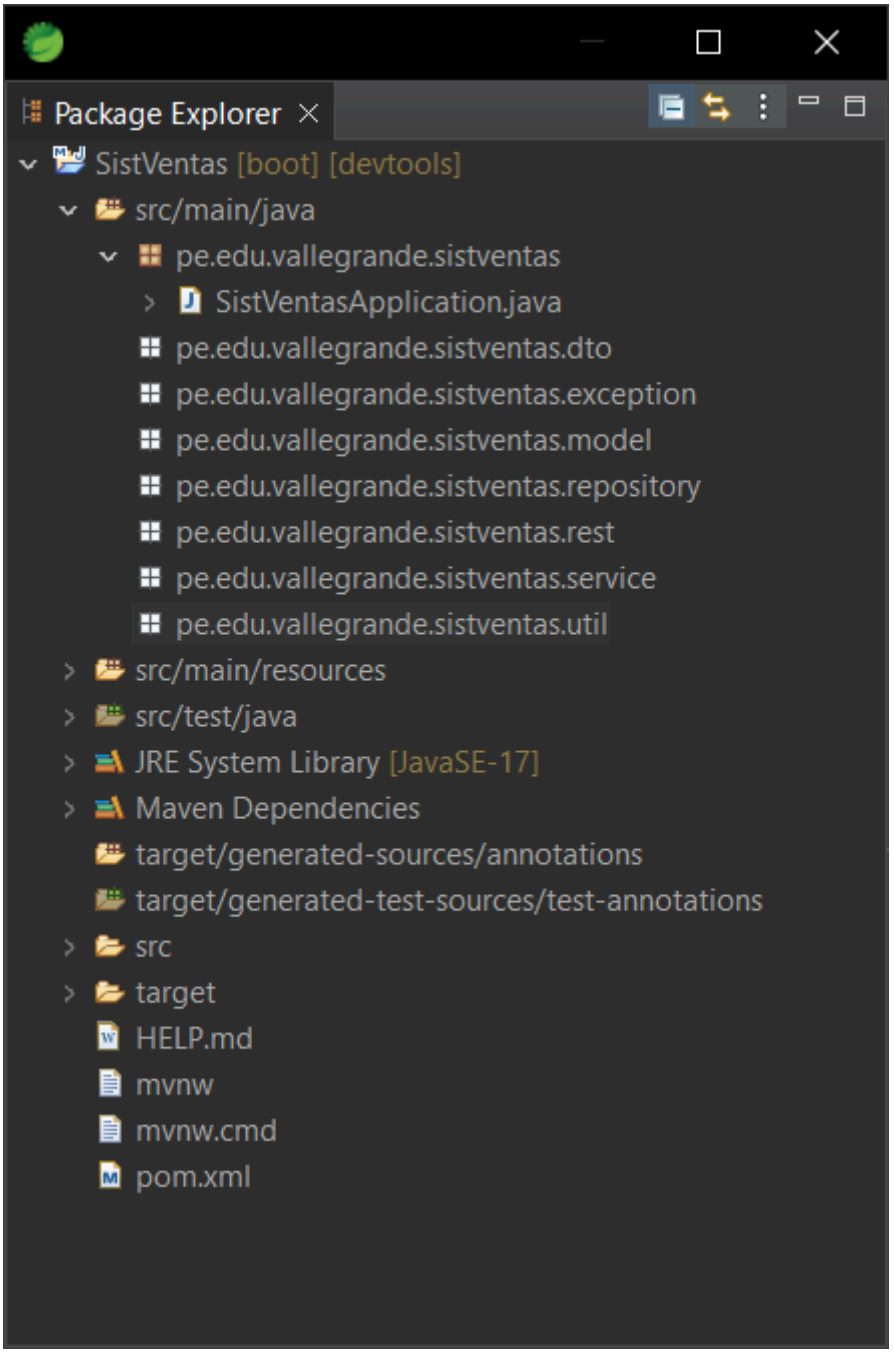
controller	pe.edu.vallegrande.<Nombre del proyecto>.controller
rest	pe.edu.vallegrande.<Nombre del proyecto>.rest
dto	pe.edu.vallegrande.<Nombre del proyecto>.dto
model	pe.edu.vallegrande.<Nombre del proyecto>.model
service	pe.edu.vallegrande.<Nombre del proyecto>.service
repository	pe.edu.vallegrande.<Nombre del proyecto>.repository
util	pe.edu.vallegrande.<Nombre del proyecto>.util
exception	pe.edu.vallegrande.<Nombre del proyecto>.exception

En caso de ser necesario un nuevo paquete, por ejemplo, un paquete para clases particulares del proyecto como clases de configuración (configuration):

pe.edu.vallegrande.<Nombre del proyecto>.configuration

De esa manera se pueden incluir nuevos paquetes.

Ejemplo 3



backend: vg-ms-[NAME-MS]

frontend: vg-web-[NAME-MS]

prs: JAVA 17

nombre de repositorios y tablas : ingles

mongo: https://cloud.mongodb.com/

Comandos GitPod Java 17:
sudo apt update

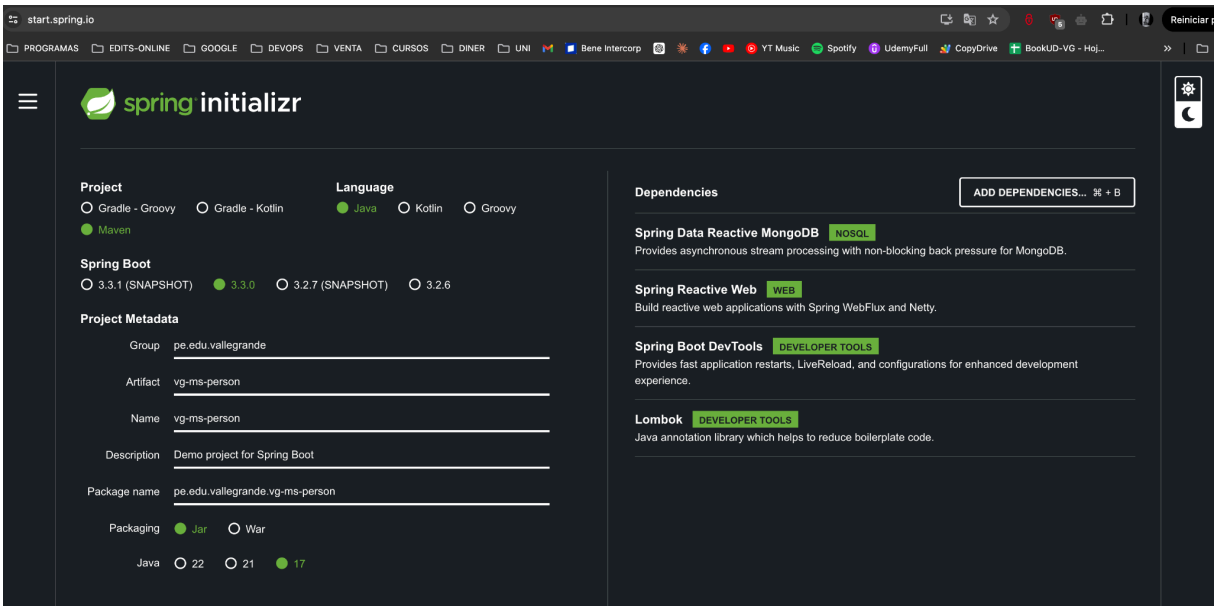
sudo apt install openjdk-17-jdk -y

export JAVA_HOME=/usr/lib/jvm/java-17-openjdk-amd64

export PATH=$JAVA_HOME/bin:$PATH

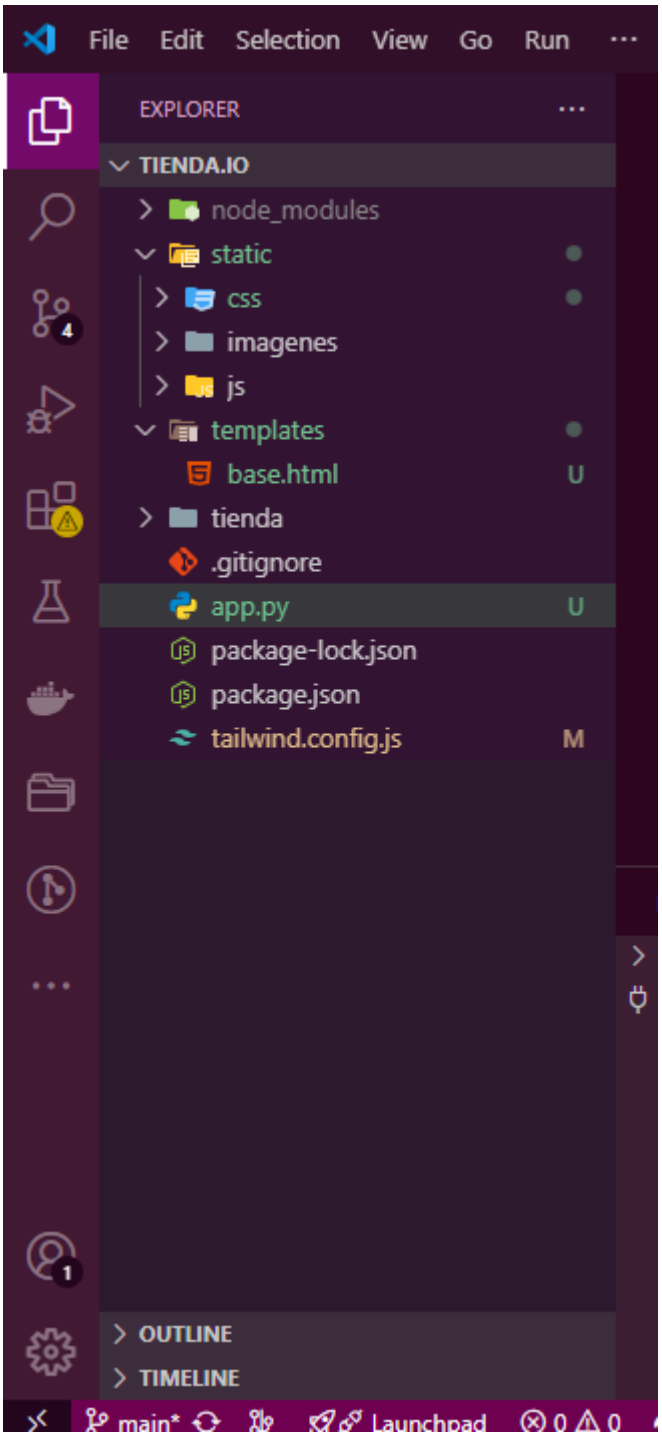
java -version

mvn -version



VALLE GRANDE

Orden de la carpetas para PYTHON (2° Semestre)



VALLE GRANDE

Orden de la carpetas para PYTHON (4° Semestre)

AULA A

Estructura de carpeta Backend (Python + Flask).

```
AS241S4_PII_T01-BE/
├── app/                        # Carpeta principal de la aplicación
│   ├── models/                 # Define las clases y modelos (entidades/tablas)
│   │   ├── __init__.py         # Permite tratar el directorio como módulo
│   │   ├── rol.py              # Modelo de datos para roles de usuario
│   │   └── usuario.py          # Modelo de datos para usuarios
│   │
│   ├── routes/                 # Maneja las rutas/endpoints de la API
│   │   ├── __init__.py         # Inicialización del módulo de rutas
│   │   ├── rol_routes.py       # Endpoints relacionados con roles
│   │   └── usuario_routes.py   # Endpoints relacionados con usuarios
│   │
│   ├── services/               # Lógica de negocio y conexión con los modelos
│   │   ├── __init__.py         # Inicialización del módulo de servicios
│   │   ├── rol_service.py      # Lógica y operaciones sobre roles
│   │   └── usuario_service.py  # Lógica y operaciones sobre usuarios
│   │
│   ├── __init__.py             # Inicialización de la aplicación Flask
│   └── settings.py             # Configuración (DB, variables de entorno, etc.)
├── Venv/                       # Entorno virtual de Python (dependencias aisladas)
├── .env                        # Variables de entorno (ej: claves, URL DB, etc.)
├── database.db                 # Base de datos SQLite (persistencia local)
├── README.md                   # Documentación del proyecto
├── requirements.txt            # Librerías y dependencias necesarias
└── run.py                      # Punto de entrada para ejecutar la aplicación
```

VALLE GRANDE

Estructura de carpeta Frontend (Vite + React + Tailwind).

```
AS241S4_PII_T01-fE/
├── public/                    # Archivos estáticos que se copian tal cual al build final
│
├── src/                       # Código fuente principal de la aplicación
│   ├── app/                   # Punto de entrada y estilos globales
│   │   ├── App.css            # Estilos principales del componente App
│   │   ├── App.jsx            # Componente raíz de la aplicación
│   │   ├── index.css          # Estilos globales
│   │   └── main.jsx           # Archivo inicial que renderiza React en el DOM
│   │
│   ├── assets/                # Recursos estáticos (imágenes, fuentes, íconosetc.)
│   │
│   ├── components/            # Componentes reutilizables (botones, headers, inputs, etc.)
│   │
│   ├── pages/                 # Vistas o páginas principales de la aplicación
│   │                          # (ej: Home.jsx, Login.jsx, Dashboard.jsx)
│   │
│   ├── shared/                # Código compartido y utilidades
│   │   ├── api/               # Funciones para llamadas a APIs (fetch, axios, etc.)
│   │
│   └── widgets/        # Componentes pequeños o módulos específicos reutilizables
│
├── .env                       # Variables de entorno (ejemplo: API_URL, claves)
├── .gitignore                 # Archivos/carpetas que Git debe ignorar
├── eslint.config.js           # Configuración de ESLint para buenas prácticas de código
├── index.html                 # HTML principal donde React se monta
├── package.json               # Configuración del proyecto y dependencias
├── package-lock.json          # Versión exacta de dependencias instaladas
├── README.md                  # Documentación del proyecto
└── vite.config.js             # Configuración de Vite (build, plugins, alias, etc.)
```