



# Presentación de proyecto MynimaList

Equipo:

Álvaro Sánchez Hernández: diseñador, implementador

David Ramírez Palacios: Scrum Master, diseñador

Isidro Javier García Fernández: diseñador, Product Owner

Jacobo Elichá Garrucho: implementador, Scrum Master

Jesús Escudero Moreno: diseñador, Product Owner

José Antonio Luque Salguero: implementador, tester

Juan Manuel García Delgado: tester, implementador

Julia Pérez Barreales: Scrum Master, diseñadora



# Índice de contenidos



- Introducción – El problema
- La solución
- El equipo y el trabajo en equipo
- Actividades de Ingeniería de Software
  - Requisitos
  - Planificación
  - Arquitectura
  - Modelos
  - Patrones/Principios
  - Pruebas
- Desarrollo/Despliegue
  - Estrategias y herramientas
  - Modelo de Implementación
  - Despliegue
- Resultados
- Conclusiones



# Introducción



- ¿Dónde sueles anotar las tareas pendientes?
- ¿Y cuándo las quieres consultar?
- ¿Cómo te gustaría realmente?



# La solución: Mynimalist

Una aplicación web **minimalista** e **intuitiva** para realizar anotaciones como listas de tareas, ideas de proyectos o recordatorios.



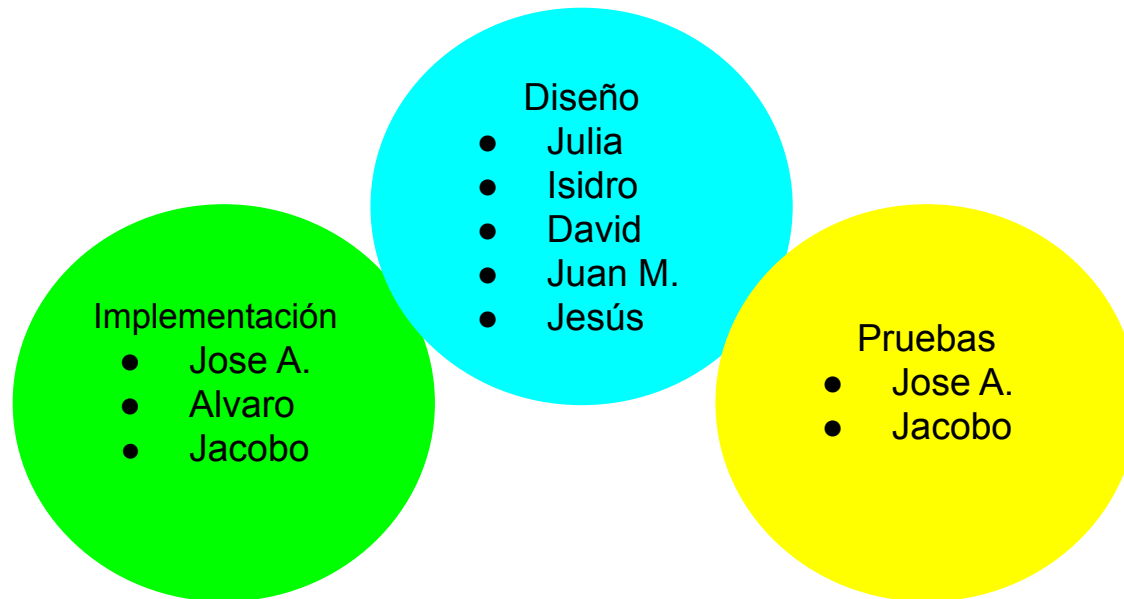
MynimaList



# El equipo y el trabajo en equipo



Siguiendo la idea de Divide & Conquer, nos dividimos en subgrupos para la realización de las tareas:





# Actividades Ingeniería del Software



- Requisitos
- Planificación
- Arquitectura
- Modelos
- Patrones
- Pruebas



# Requisitos



- Funcionales
- No Funcionales



# Requisitos Funcionales

- Recuadros inicio de sesión
  - Botón inicio de sesión
- Botón para ir a la página de registro

MynimaList

Usuario

Contraseña

INICIAR SESION    REGISTRARSE

☾    Sobre nosotros





# Requisitos Funcionales

- Recuadros de registro
  - Botón de registro

MynimaList

Correo electrónico

---

Usuario

---

Contraseña

---

REGISTRARSE VOLVER

☾



# Requisitos Funcionales

- Página principal de listas
  - Menú de listas
    - Crear lista
    - Editar lista
    - Borrar lista
      - Confirmar borrado de lista
    - Cierre de sesión

**Desea eliminar la lista**



**Confirmar**






# Requisitos Funcionales


- Crear tarea
- Editar Tarea
- Borrar tarea
- Completar tarea


**MynimaList**  

Listas 

Lista1

**Lista1**

☐ Tarea1 

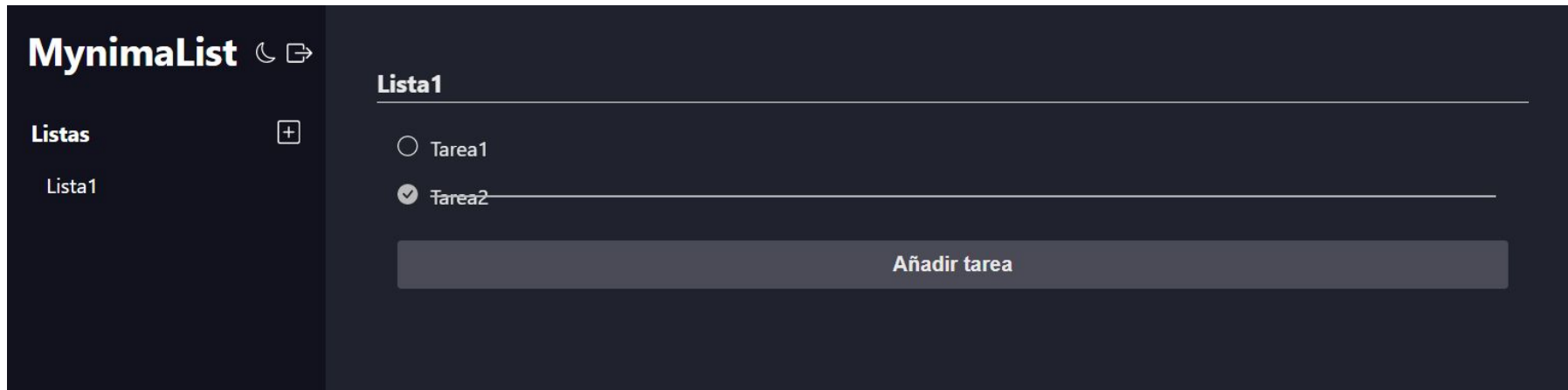
☒ Tarea2 

Añadir tarea



# Requisitos Funcionales

- Modo oscuro





# Requisitos Funcionales

- About Us

## Sobre nosotros



*MynimaList* nace como una solución simple y efectiva para todos aquellos que quieren generar listas de tareas de forma sencilla y rápida.

Ha sido desarrollada por un grupo de alumnos del Doble Grado de Ingeniería Informática + Matemáticas de la Universidad de Málaga, como proyecto final para la asignatura de *Introducción a la Ingeniería del Software*, donde hemos aprendido metodologías de desarrollo de software como Scrum, también hemos aprendido a documentar el proyecto de una forma más completa y parecida a lo que se hace en la realidad y además, hemos tenido un primer contacto con tecnologías como HTML, CSS, React, Java para el backend, Git...





# Requisitos No Funcionales



- Encriptado de contraseñas
- Almacenamiento en nuestra BBDD
- Página intuitiva y sencilla
- Licencia de software
- Control de correo de usuario
- Guardar datos de usuario



# Planificación

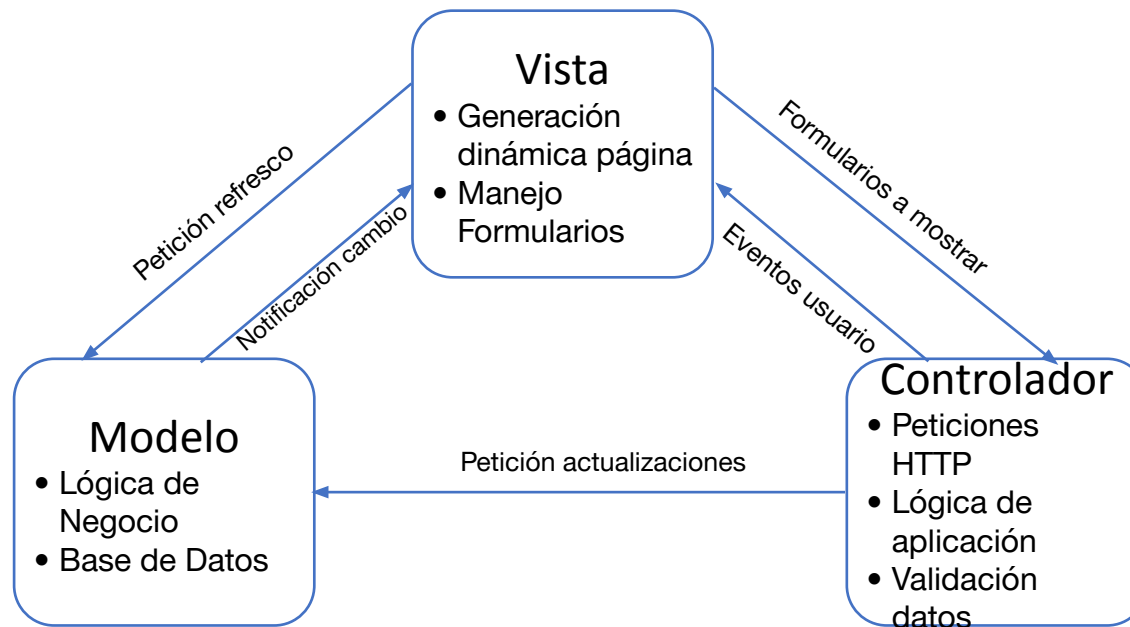


## Metodología Scrum

- Comunicación entre los miembros del grupo
- Proyecto pequeño y flexible
- Poca experiencia
- Ciclos cortos de Sprint
- Roles:
  - Product Owner, Scrum Master, desarrollador, implementador y tester

# Arquitectura

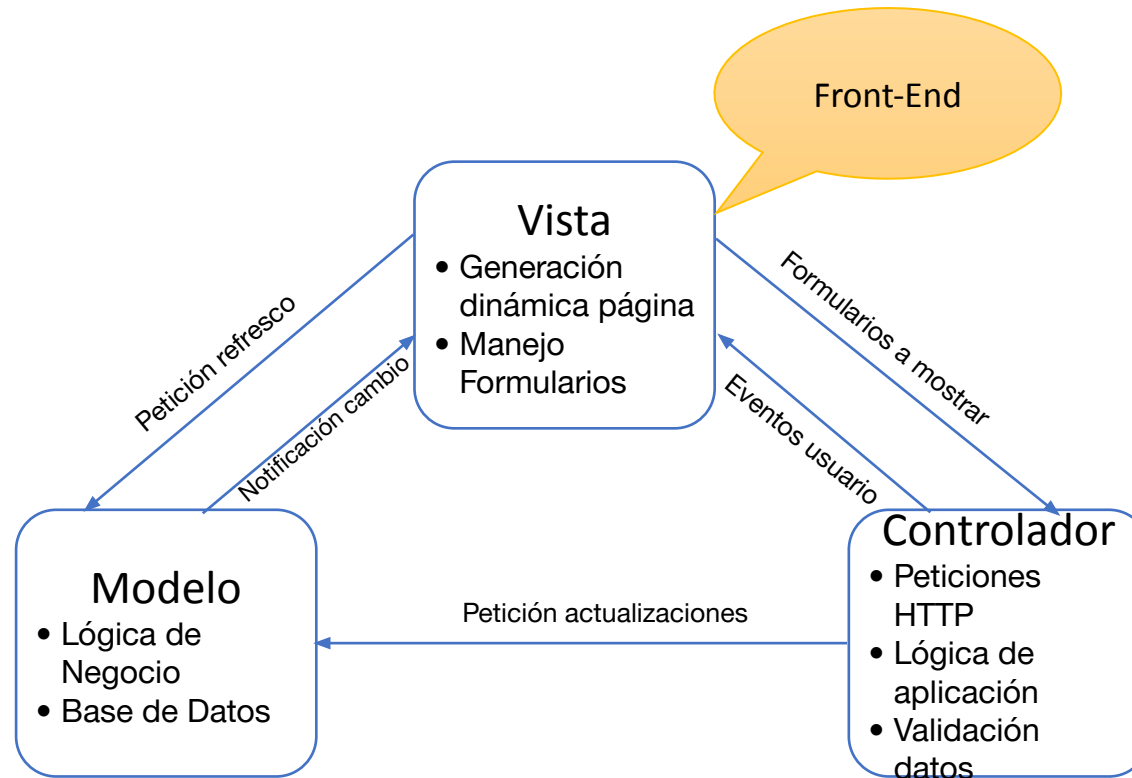
## Modelo Vista Controlador





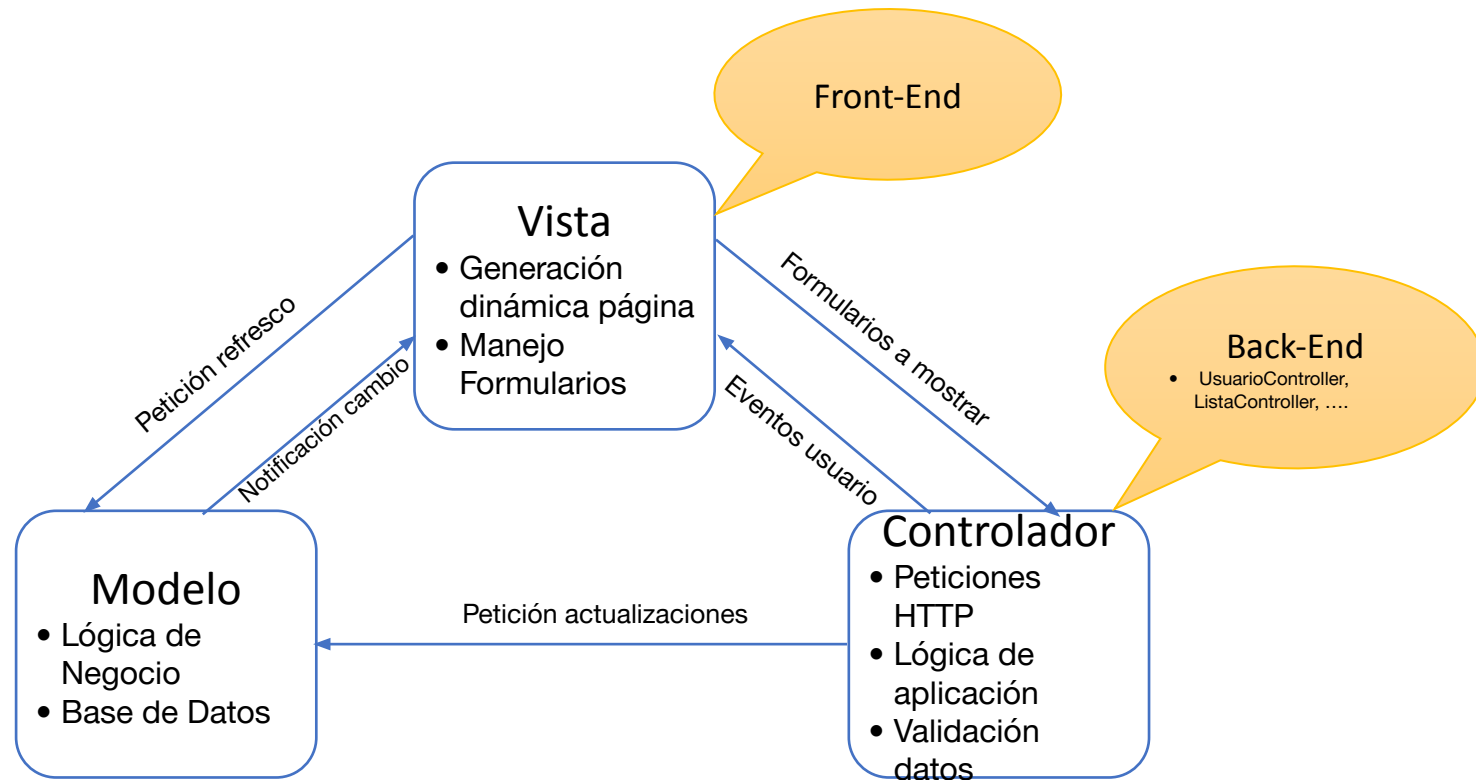
# Arquitectura

## Modelo Vista Controlador

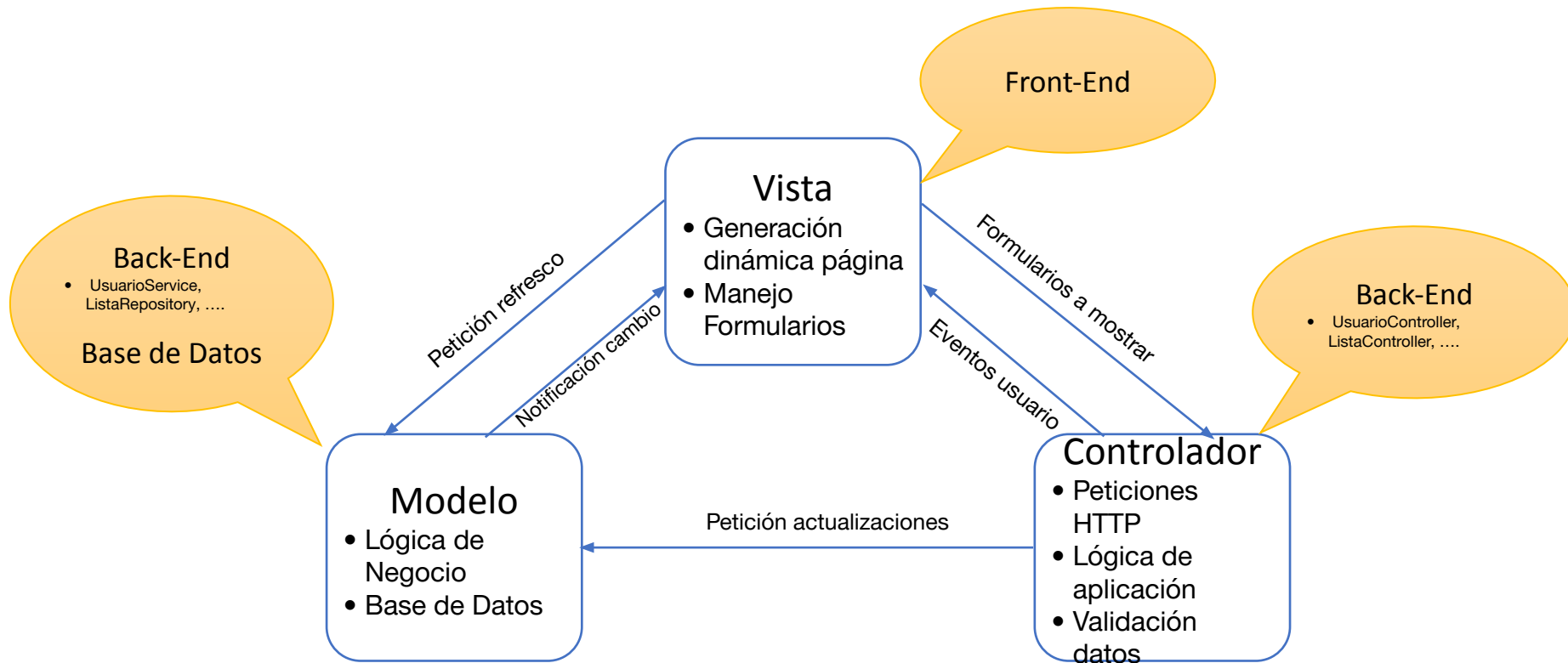


# Arquitectura

## Modelo Vista Controlador



## Modelo Vista Controlador

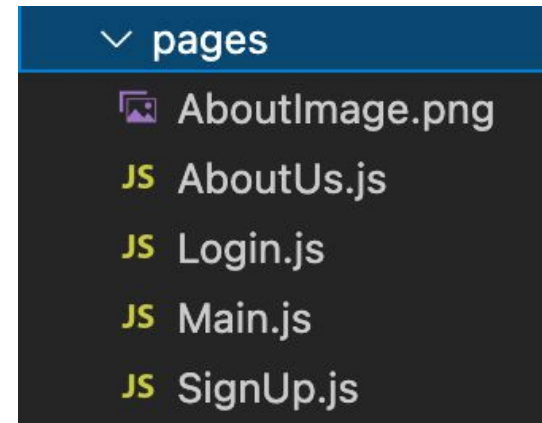




# Arquitectura



- Cada una de las páginas renderiza los componentes necesarios.
- Lo hacemos haciendo uso de react router de modo que sea una SWA.





## ▼ components

- JS AddNewList.js
- JS AddNewTask.js
- JS Body.js
- JS DeleteList.js
- JS EditTaskForm.js
- JS Header.js
- JS List.js
- JS ListButtons.js
- JS ListForm.js
- JS Lists.js
- JS LogoWriting.js
- JS Modal.js
- JS RenameList.js
- JS Sidebar.js
- JS Task.js
- JS TaskForm.js
- JS Tasks.js
- JS Theme.js

- Todos los componentes tienen un alto nivel de reutilización, se ha desarrollado pensando en la escalabilidad.
- Cuando un componente hace una petición, espera una respuesta y cuando la recibe actualiza todos los componentes necesarios.



# Arquitectura



MynimaList

☾ ➡

Listas

+

Exámenes java

Exámenes java

☐ Septiembre 2017

☐ Junio 2017

☐ Examen de prueba

Añadir tarea

Tasks

Podemos ver los componentes de Task y AddNewTask



# Arquitectura



MynimaList

☾

📄

Listas

+

Exámenes java

Exámenes java

☐ Septiembre 2017

☐ Junio 2017

☐ Examen de prueba

Añadir nueva tarea

Junio 2018

Confirmar

Si pulsamos en Añadir tarea veremos otros dos componentes, el modal y el TaskForm, reutilizables desde el punto de vista del código.



# Modelos



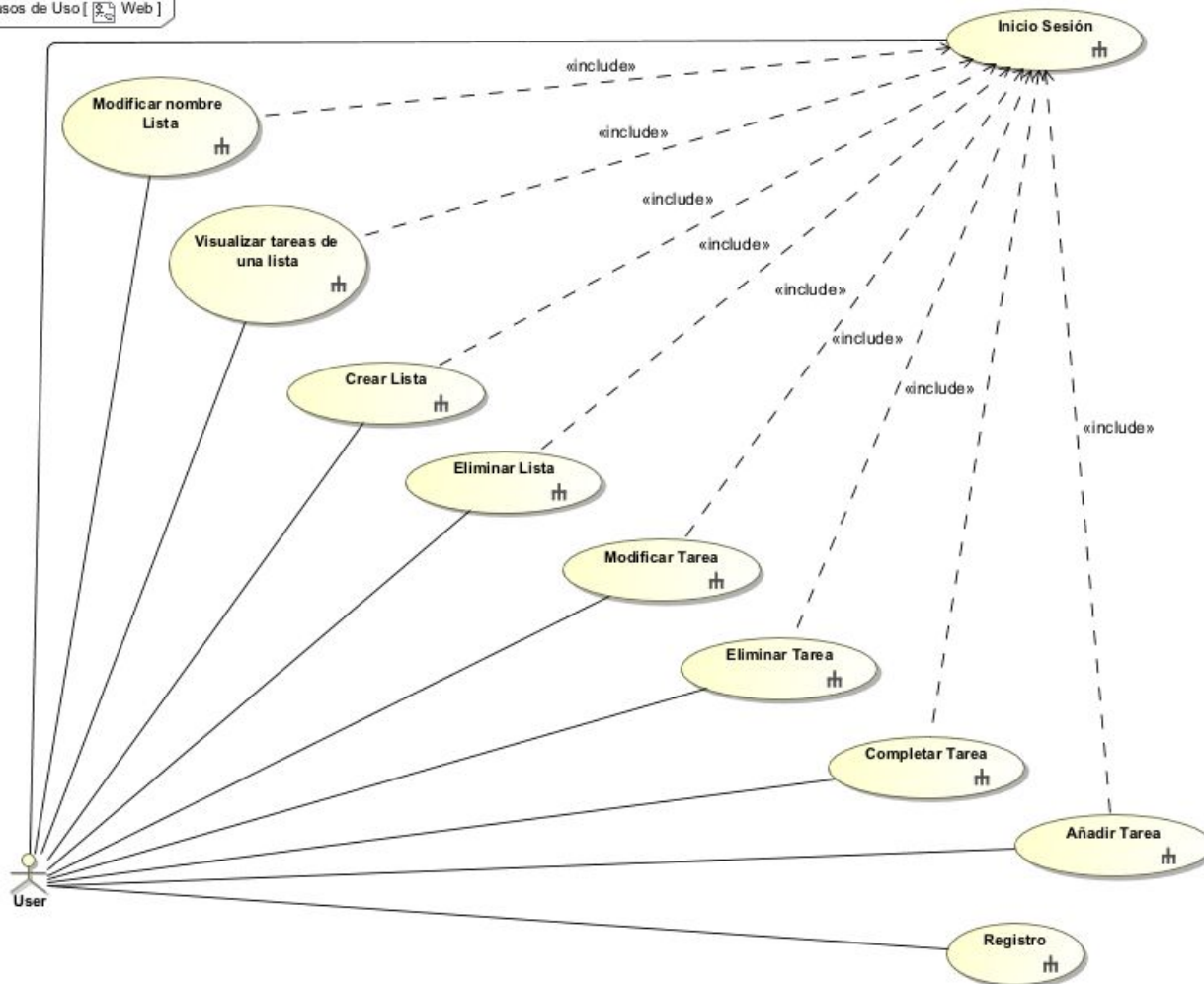
- Diagrama de Casos de Uso
- Diagrama de Secuencias
- Diagrama de Clases



# Diagrama de Casos de Uso



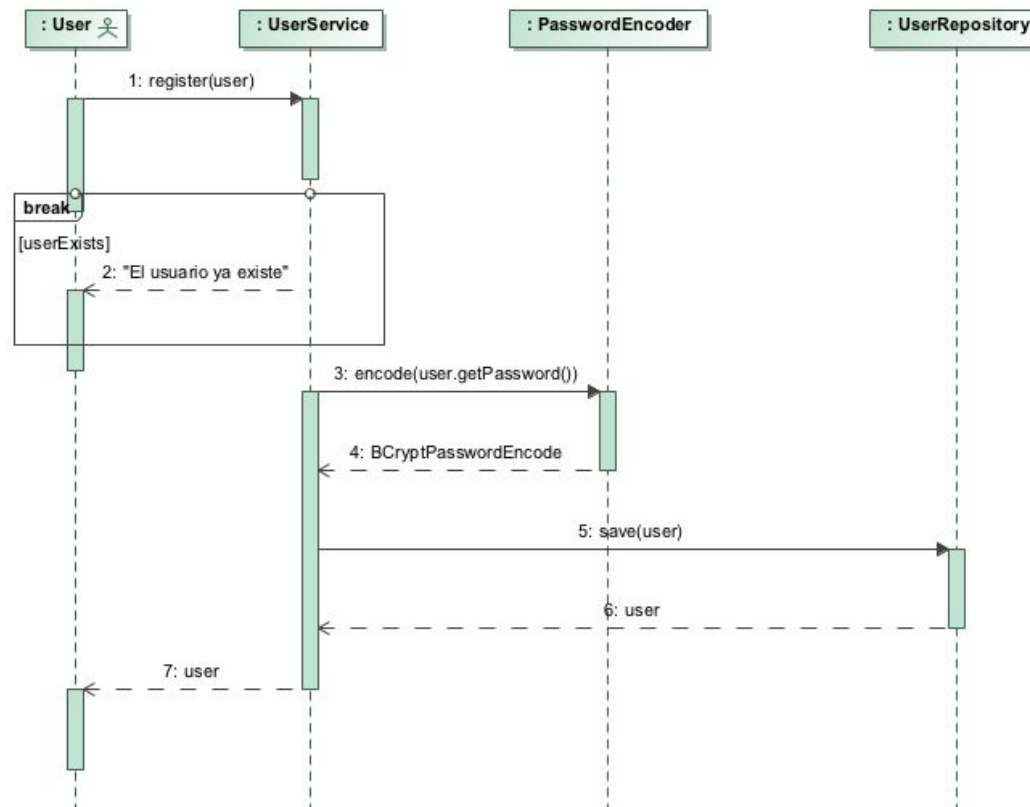
package 02 Casos de Uso [ Web ]



# Diagrama de Secuencia

Tenemos un diagrama de secuencia por cada caso de uso, veamos algunos ejemplos:

(Registro)

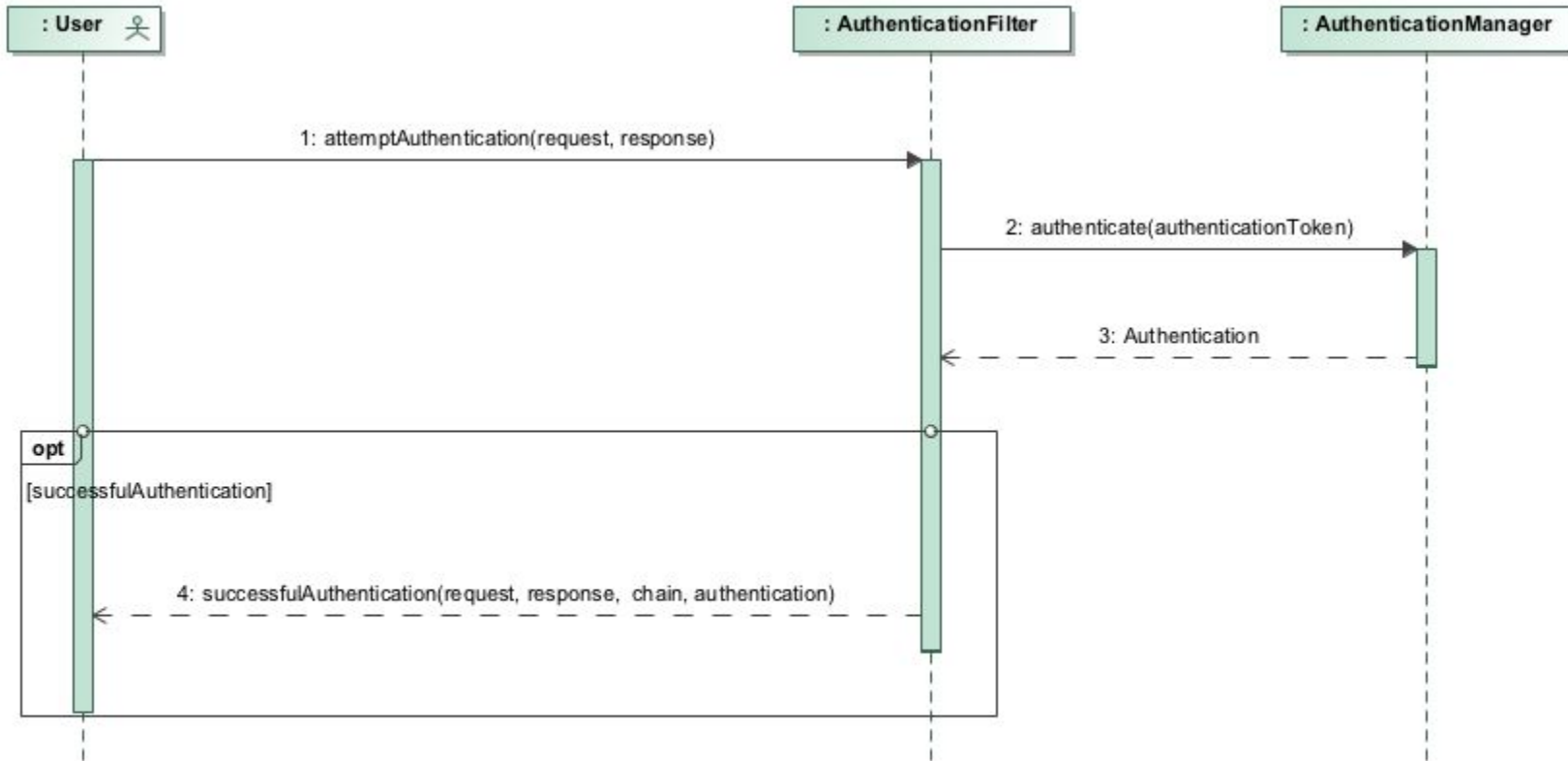




# Diagrama de Secuencia



(Inicio de Sesion)

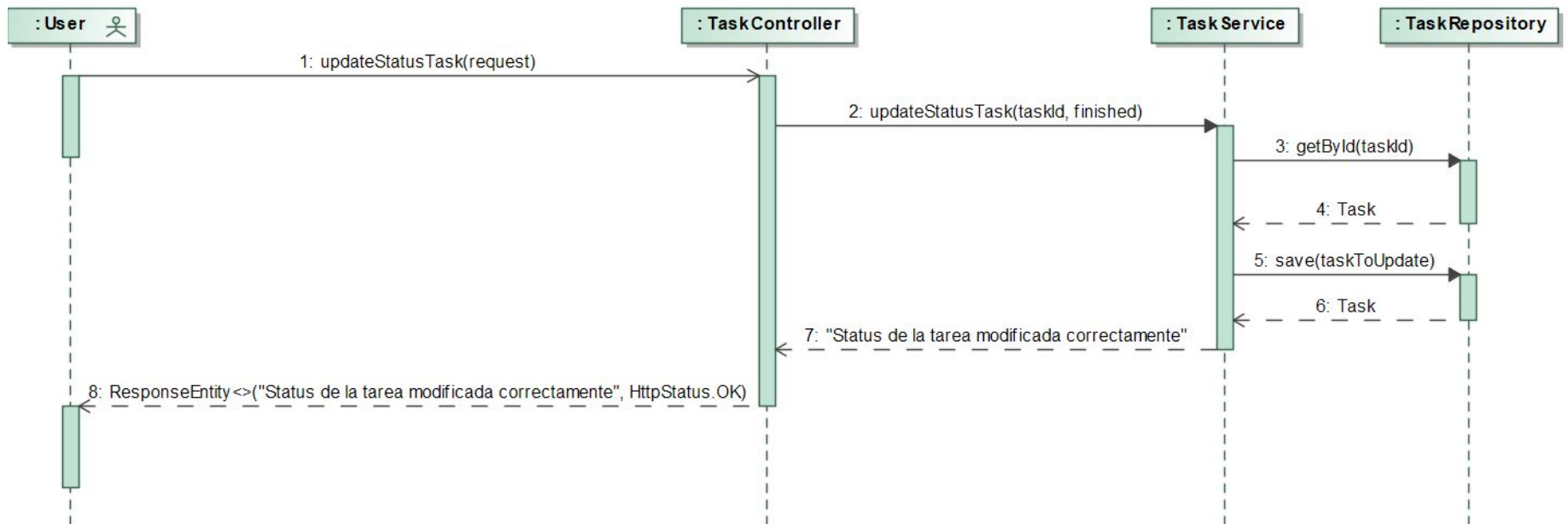




# Diagrama de Secuencia



(Completar tarea)

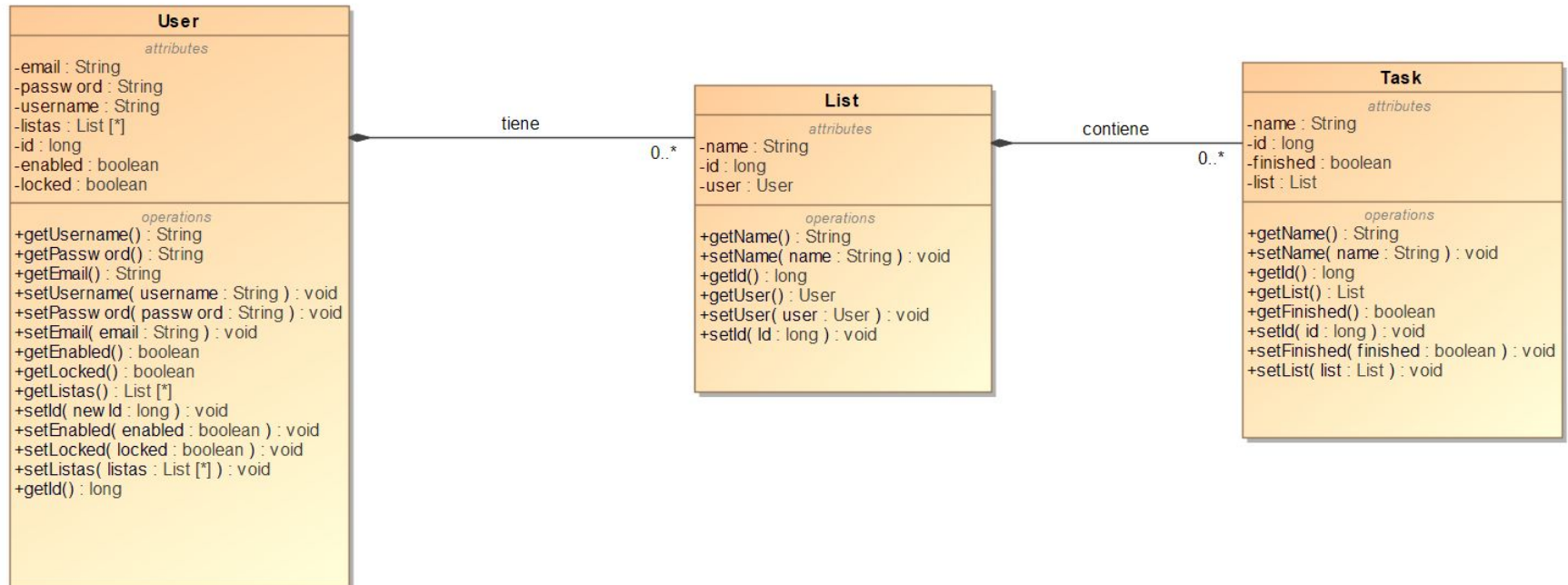




# Diagrama de Clases



package 03 Diagrama de clases [ 03 Diagrama de clases ]





# Patrones



## Strategy

- Encriptado de firma de JWT

```
Algorithm algorithm = Algorithm.HMAC256(secret.getBytes());  
JWTVerifier verifier = JWT.require(algorithm).build();
```

## Chain of Responsibility

- Filtro de autenticación

```
public class AuthenticationFilter
```



# Pruebas



Hemos realizado pruebas unitarias tanto de las clases principales (User, List y Task), así como de las clases Service correspondientes (más interesantes), en concreto se tratan de test de caja blanca.

Hemos usado Mockito para simular el comportamiento de clases ajenas a la implicada en la prueba.



Veamos algunos ejemplos:



# Pruebas



## UserServiceTest.java

- `void testLoadUserByUsernameForNonExistingUsername()`  
Comprueba que al buscar a un usuario por un nombre de usuario no registrado salta una excepción
- `void testLoadUserByUsernameForExistingUsername()`  
Comprueba que al buscar a un usuario por un nombre de usuario registrado se devuelve el usuario correcto
- `void testRegisterExistingUser()`  
Comprueba que al intentar registrar un usuario con un nombre ya registrado, salta una excepción
- `void testRegisterNewUser()`  
Comprueba que al intentar registrar un nuevo usuario con las correctas credenciales, este se registra correctamente





# Pruebas



```
1 @Override
2 public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
3     return userRepository.findByUsername(username)
4         .orElseThrow(() -> new UsernameNotFoundException(
5             String.format(USER_NOT_FOUND_MSG, username)));
6 }
```



```
1 @Transactional
2 public User register(User user){
3     boolean userExists = userRepository.findByUsername(user.getUsername()).isPresent();
4     if (userExists) {
5         throw new IllegalStateException("El usuario ya existe");
6     }
7
8     String encodedPassword = bCryptPasswordEncoder.encode(user.getPassword());
9     user.setPassword(encodedPassword);
10
11     return userRepository.save(user);
12 }
```



# Pruebas



## ListServiceTest.java



```
1  @Mock
2  private ListRepository listRepository;
```



```
1  @Test
2  void testUpdateNameList(){
3      when(listRepository.getById(RECORD.getId())).thenReturn(RECORD);
4      when(listRepository.save(any(List.class))).thenReturn(EXPECTED_RESULT);
5      List result = listService.updateNameList(RECORD.getId(), "newListName");
6      assertNotNull(result);
7      assertEquals("newListName", result.getName());
8      assertEquals(RECORD.getId(), result.getId());
9      assertEquals(RECORD.getUser(), result.getUser());
10 }
```



# Desarrollo



- Estrategias y herramientas
- Modelo de implementación
- Despliegue



# Estrategias y herramientas



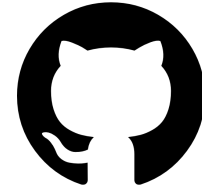
- Herramientas de comunicación:

- Discord
- WhatsApp



- Herramientas de trabajo colaborativo

- GitHub
- Trello



- Herramientas de elaboración de documentos

- Google Docs
- Visual Studio Code
- IntelliJ Idea

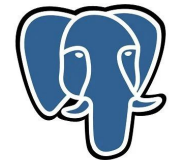
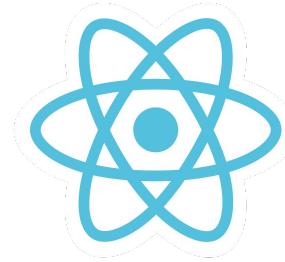




# Modelo de implementación



- Front-End:
  - HTML / JavaScript
  - React
  - React Bootstrap Icons
  - Axios
  - React router
  - CSS
- Back-End:
  - Java 17
  - Spring Boot (framework)
  - Lombok
  - Websecurity
  - Postgre (driver)
  - JSON Web Token

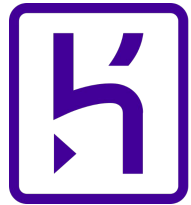


PostgreSQL





# Despliegue



**HEROKU**



**MynimaList**

<http://mynimalist.herokuapp.com/>



# Resultados



Finalmente hemos creado una página web minimalista con las siguientes funcionalidades:

- Registro
- Inicio de sesión
- Página principal
  - Menú de listas
    - Crear lista
    - Editar lista
    - Borrar lista
  - Listas
    - Crear tarea
    - Editar tarea
    - Borrar tarea
    - Completar tarea



# Resultados



- Modo oscuro
- Subida a Internet
- Responsive
- About us





# Conclusiones



*No es sencillo desarrollar un proyecto software, pero una buena planificación y un buen espíritu de equipo lo facilita inmensamente.*