DUIT

R1L2

Diseño Tecnico

Autores:
Aleixo Fernández Cuevas
Cristo Manuel Navarro Martín

Nombre del fichero:
DAW_PRW_R1L2_UT01.3 – Diseño Tecnico

Fecha de esta versión:
01/02/2026

Ciclo Formativo:
Desarrollo de Aplicaciones Web – Semipresencial (DAWN)

Módulo:
Proyecto de Desarrollo de Aplicaciones Web (PRW)

Historial de revisiones

Fecha	Descripción	Autores
21/12/2025	Fase de análisis. Versión inicial	Aleixo F. Cuevas / Cristo N. Martín
10/01/2026	Primera corrección	Aleixo F. Cuevas / Cristo N. Martín

Índice

1. Introducción

El presente documento describe el **diseño técnico de la aplicación Duit**, una plataforma desarrollada utilizando **Spring Boot 3.5.10** y **Java 21**, siguiendo una arquitectura **MVC (Modelo–Vista–Controlador)** y empleando **PostgreSQL** como sistema de gestión de bases de datos.
El objetivo principal de este documento es servir como **referencia técnica** para el desarrollo, mantenimiento y despliegue de la aplicación, proporcionando una visión clara de la arquitectura, los componentes del sistema y las decisiones técnicas adoptadas durante su implementación.
Asimismo, este documento está orientado tanto a un **contexto académico**, facilitando la comprensión del funcionamiento interno de la aplicación, como a un **entorno profesional**, permitiendo que otros desarrolladores puedan comprender, ampliar o mantener el sistema de forma eficiente.

2. Información del Proyecto

- **Nombre**: Duit
- **Versión**: 0.0.1-SNAPSHOT
- **Framework**: Spring Boot 3.5.10
- **Java Version**: 21 (LTS)
- **Base de Datos**: PostgreSQL (Neon)
- **Servidor de Aplicaciones**: Tomcat Embebido
- **Motor de Plantillas**: Thymeleaf
- **Despliegue Producción**: Koyeb
- **Control de Versiones**: Git / GitHub
- **Repositorio**: https://github.com/FerCueA/Duit

3. Requisitos Técnicos

Plataforma y Herramientas de Desarrollo

La aplicación **Duit** se desarrolla utilizando tecnologías modernas ampliamente adoptadas en entornos profesionales, garantizando **mantenibilidad, escalabilidad y compatibilidad**.

Herramienta	Tipo	Propósito
Visual Studio Code	IDE	Desarrollo frontend y edición general
Firefox / Brave	Navegadores	Pruebas funcionales y depuración
Git	Control de versiones	Gestión del código fuente
Figma	Diseño	Prototipado de interfaces
Trello	Gestión	Planificación y seguimiento de tareas
DBeaver	Gestión BD	Administración de base de datos

Deployment

Tecnología	Tipo	Propósito
Koyeb	PaaS	Hosting de la aplicación web
Neon	PostgreSQL	Hosting PostgreSQL autocontenida

Tecnologías principales frontend

Tecnología	Versión	Tipo	Propósito
HTML5	–	Lenguaje de marcado	Estructura semántica de las páginas web
CSS3	–	Estilos	Diseño visual y responsive
JavaScript	ES6	Lenguaje	Interactividad del lado del cliente
Thymeleaf	3.1.x	Template Engine	Renderizado dinámico de vistas
Bootstrap	5.3.8	CSS Framework	Diseño responsive y componentes UI
Bootstrap Icons	1.13.1	Iconos	Conjunto de iconos vectoriales

Plataforma de desarrollo

Tecnología	Versión	Tipo	Propósito
Java	21 LTS	Lenguaje	Base del proyecto, soporte hasta 2029
Spring Boot	3.5.10	Framework	Framework principal y autoconfiguración
Spring Framework	6.2.x	Framework	Núcleo de Spring (dependencia transitiva)
Maven	3.9+	Build Tool	Gestión de dependencias y proceso de build
Tomcat (embebido)	10.1.x	Servidor	Servidor web integrado

Persistencia

Tecnología	Versión	Tipo	Propósito
Spring Data JPA	3.5.x	Framework	Abstracción de acceso a datos
Hibernate ORM	6.6.x	ORM	Implementación JPA
Hibernate Validator	8.0.x	Validación	Bean Validation
PostgreSQL Driver	42.7.x	Driver JDBC	Conexión con PostgreSQL
HikariCP	5.1.x	Pool	Pool de conexiones
Jakarta Persistence API	3.1.0	Especificación	JPA Spec

Seguridad

Tecnología	Versión	Tipo	Propósito
Spring Security	6.4.x	Framework	Autenticación y autorización
Spring Security Web	6.4.x	Módulo	Filtros HTTP
Spring Security Config	6.4.x	Módulo	Configuración de seguridad
BCrypt	Incluido	Algoritmo	Hash seguro de contraseñas

Validación y Email

Tecnología	Versión	Tipo	Propósito
Spring Validation	3.5.x	Framework	Validación de datos
Jakarta Validation	3.0.2	Especificación	Bean Validation
Spring Mail	3.5.x	Framework	Envío de correos
Jakarta Mail	2.0.x	API	JavaMail API

Testing (scope: test)

Tecnología	Versión	Tipo	Propósito
JUnit Jupiter	5.11.x	Framework	Testing unitario
Mockito	5.14.x	Framework	Mocking
AssertJ	3.26.x	Librería	Assertions fluidas
Spring Boot Test	3.5.10	Framework	Testing de integración
Spring Security Test	6.4.x	Framework	Testing de seguridad
Hamcrest	3.0.x	Librería	Matchers

4. Gestión de Dependencias

La gestión de dependencias se realiza mediante **Maven**, utilizando los *starters* de Spring Boot para garantizar compatibilidad y simplicidad en la configuración.

Dependencia	Versión	Propósito
Spring Boot	3.5.10	Framework principal
Spring Data JPA	Incluida	Persistencia
Spring Security	Incluida	Seguridad
Spring Web	Incluida	Controladores REST/MVC
Spring Mail	Incluida	Envío de correos
Thymeleaf	Incluida	Motor de plantillas
Thymeleaf Security	6.x	Integración con Security
PostgreSQL Driver	Latest	Conector de base de datos
Lombok	Latest	Reducción de boilerplate
Jakarta Validation	Incluida	Validación de beans
Spring Dotenv	3.0.0	Variables de entorno

5 Arquitectura y Patrones

Durante el desarrollo de la aplicación **Duit** se han aplicado patrones de diseño y principios arquitectónicos ampliamente utilizados en el ecosistema **Spring**, con el objetivo de garantizar un sistema **modular, mantenible y escalable**.

5.1 Arquitectura MVC

La aplicación sigue el patrón MVC, separando claramente las responsabilidades entre:

Modelo: Entidades JPA y lógica de dominio.
Vista: Plantillas Thymeleaf encargadas de la presentación.
Controlador: Componentes responsables de gestionar las peticiones HTTP y coordinar la lógica de negocio.

Esta separación mejora la mantenibilidad del código y facilita la evolución independiente de cada capa.

5.2 Patrón Repository

Se utiliza el patrón Repository mediante **Spring Data JPA** para desacoplar el acceso a datos de la lógica de negocio.
Los repositorios actúan como una abstracción sobre la capa de persistencia, permitiendo:

- Reducir el acoplamiento entre capas.
- Facilitar el cambio o ampliación del sistema de persistencia.
- Simplificar la escritura y el mantenimiento del código.

5.3 Inyección de dependencias

La gestión de componentes se realiza mediante el mecanismo de **inyección de dependencias** proporcionado por el Spring Framework.
Este enfoque permite:

- Reducir dependencias directas entre clases.
- Facilitar la reutilización y el testeo de componentes.
- Centralizar la configuración y el ciclo de vida de los objetos.
-

En conjunto, estas decisiones arquitectónicas contribuyen a un diseño limpio y alineado con las buenas prácticas de desarrollo de aplicaciones empresariales en Java.

6. Seguridad

6.1. Configuración de Spring Security

El sistema implementa mecanismos de **autenticación y autorización** mediante **Spring Security 6**, centralizados en la clase de configuración SecurityConfig.

1. Autenticación basada en formulario

- **Página de login:** /login
- **URL de procesamiento:** /login
- **Redirección tras login correcto:** /home
- **Redirección en caso de error:** /login?error=true

2. Rutas públicas (sin autenticación)

- Rutas principales: /, /index, /public/**
- Autenticación y registro: /login, /registro, /register
- Recursos estáticos: /css/**, /js/**, /img/**
- Páginas informativas: /privacidad, /terminos, /ayuda

3. Funcionalidad Remember Me

- **Validez del token:** 24 horas (86.400 segundos)
- **Gestión automática de cookies**
- **Persistencia segura de sesión**

4. Logout

- Invalidación de la sesión HTTP
- Eliminación de cookies de autenticación
- Redirección a /login?logout=true

5. Protección CSRF

- **Estado actual:** deshabilitado en entorno de desarrollo
- **Previsión:** habilitación en entorno de producción

6.2 CustomUserDetailsService

Clase responsable de cargar los datos del usuario desde la base de datos para el proceso de autenticación.
Funcionalidades

- Implementa la interfaz UserDetailsService de Spring Security
- Carga usuarios mediante **username (email)**
- Conversión de la entidad AppUser a UserDetails
- Gestión de **roles y autoridades**

6.3. Encriptación de Contraseñas

La aplicación utiliza el algoritmo **BCrypt** para el cifrado seguro de contraseñas.

```
Configuración
@Bean
public PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
}
```

Características

- Generación automática de *salt* por contraseña
- Alta resistencia a ataques de fuerza bruta
- Coste computacional configurable

6.4 Control de Acceso Basado en Roles

La aplicación implementa un **sistema de control de acceso basado en roles**, restringiendo el acceso a funcionalidades según los permisos asignados al usuario autenticado.

Rol	Permisos
ADMIN	Acceso total al sistema y panel de administración
PROFESSIONAL	Gestión de postulaciones y perfil profesional
USER	Creación de solicitudes y contratación de servicios

6.5 Auditoría de Seguridad

El sistema incorpora mecanismos de auditoría para el seguimiento de accesos mediante un **Access Log**.

Información registrada

- Intentos de acceso al sistema
- Dirección IP de origen
- Marca temporal (*timestamp*)
- Resultado del acceso (éxito o fallo)

6.6 Auditoría Automática de Entidades (BaseEntity)

Todas las entidades principales heredan de una clase base común encargada de la **auditoría automática**.

```
@MappedSuperclass
@EntityListeners(AuditingEntityListener.class)
public abstract class BaseEntity {

    @CreatedBy
    private String createdBy;

    @CreatedDate
    private LocalDateTime createdAt;

    @LastModifiedBy
    private String updatedBy;

    @LastModifiedDate
    private LocalDateTime updatedAt;
}
```

Gracias a este mecanismo, el sistema registra automáticamente:

- El usuario responsable de la creación y última modificación de cada entidad.
- La fecha y hora de creación.
- La fecha y hora de la última actualización.

Esta estrategia mejora la trazabilidad, facilita la auditoría de cambios y aporta un mayor control sobre la evolución de los datos dentro de la aplicación, sin necesidad de implementar lógica adicional en cada entidad.

7. Integraciones Externas

En la versión actual, el sistema **no hace uso efectivo de APIs externas ni servicios de terceros**, manteniendo una arquitectura autocontenida.

8. Justificación de Tecnologías

Las herramientas y tecnologías utilizadas se han elegido principalmente en base a los contenidos trabajados durante el ciclo formativo, utilizando aquellas vistas en clase y que se adaptan al tipo de aplicación web que se quiere desarrollar.

8.1. Frontend

HTML, CSS y JavaScript junto con **Bootstrap 5** facilitan el montaje de interfaces de forma rápida y responsive. Bootstrap permite aprovechar componentes reutilizables (formularios, navegación, layouts) que han sido adaptados a las necesidades del proyecto. **Bootstrap Icons** mantiene coherencia visual sin necesidad de librerías adicionales.
Thymeleaf como motor de plantillas permite integrar el frontend con el backend de forma sencilla, generando vistas dinámicas a partir de los datos del servidor. Esta forma de trabajo se ha utilizado durante el curso junto con Spring Boot.

8.2. Backend

Java 21 junto con **Spring Boot 3.5.10** permite organizar el proyecto con una arquitectura MVC clara, separando la lógica de negocio, el acceso a datos y las vistas. **PostgreSQL** se utiliza como sistema gestor de base de datos relacional por su robustez, escalabilidad y compatibilidad con el entorno cloud.

8.3. Herramientas

Git y GitHub proporcionan control de versiones profesional. **Trello** facilita la planificación y gestión de tareas. **Figma** permite diseñar y prototipar interfaces antes de implementarlas.

8.4. Despliegue

Koyeb se utiliza para el despliegue en producción por su facilidad de uso y compatibilidad con aplicaciones Spring Boot. **Neon** proporciona hosting serverless para PostgreSQL con alta disponibilidad y backups automáticos, constituyendo una oportunidad de aprendizaje en tecnologías cloud.

9. Reparto de tareas y responsabilidades

El desarrollo de la aplicación **Duit** se ha realizado de forma colaborativa, distribuyendo las tareas entre los miembros del equipo según sus responsabilidades y aportaciones al proyecto, garantizando un desarrollo equilibrado y coordinado.

Cristo Manuel Navarro Martín

Responsable del **desarrollo principal de la aplicación y del despliegue en producción**.

Tareas realizadas:

- Diseño de la arquitectura general del sistema (MVC).
- Implementación de la lógica de negocio y servicios.
- Desarrollo de la capa de persistencia mediante **Spring Data JPA** y **Hibernate**.
- Configuración e implementación de **Spring Security** (autenticación, autorización y roles).
- Implementación del sistema de auditoría automática de entidades (BaseEntity).

Aleixo Fernández Cuevas

Responsable del **diseño de la interfaz de usuario, apoyo al desarrollo y documentación**.

Tareas realizadas:

- Diseño de la interfaz de usuario con **HTML5, CSS3 y Bootstrap**.
- Prototipado de interfaces utilizando **Figma**.
- Configuración del despliegue en producción mediante **Koyeb**.
- Integración y gestión de la base de datos **PostgreSQL en Neon**.
- Redacción principal y documentación asociada.
- Revisión y corrección de la documentación del proyecto.

Coordinación y Trabajo en Equipo

Ambos integrantes han participado conjuntamente en:

- Definición de los requisitos técnicos y funcionales.
- Desarrollo parcial del backend con **Spring Boot** y **Java 21**.
- Creación y adaptación de vistas dinámicas mediante **Thymeleaf**.
- Toma de decisiones tecnológicas.
- Control de versiones mediante **Git y GitHub**.