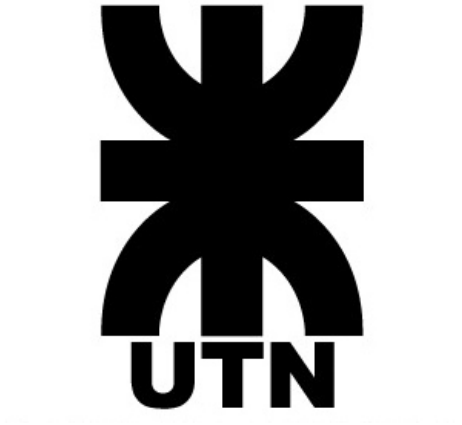
Test Driven Development (TDD)

Trabajo práctico 6



UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL CÓRDOBA

Ingeniería y Calidad de Software

GRUPO 3

Integrantes:

Barros, Enrique Agustín	98900
Capdevila Sauch, Catalina	77395
Figueroa, Francisco Emiliano	96809
Landi, Agostina Milena	94849
Lejtman, Uriel Benjamín	95903
Moreno, Matías Emanuel	95457
Moyano, Tomás	96846
Paris, Tomas	76791
Pedraza, Antonella	94868
Salzgeber, Ian	94173

Docentes: Ing. Laura Covaro - Ing. Cecilia Massano - Ing. Constanza Garnero

Curso: 4K3

Fecha de entrega: 24/10/2025

Documento de justificación de decisiones de diseño

1. Lenguaje de programación: JavaScript

Se decidió implementar el backend utilizando **JavaScript** debido a su amplia flexibilidad y compatibilidad con el entorno **Node.js**, lo que permite desarrollar aplicaciones del lado del servidor con buen rendimiento y una gran cantidad de librerías disponibles.
Además, su sintaxis accesible y su orientación a eventos facilitan la integración con APIs externas y la comunicación con el frontend.

2. Framework de pruebas: Jest

Se eligió **Jest** como herramienta de testing por ser un framework ligero, rápido y ampliamente utilizado en proyectos Node.js.
Permite realizar **tests unitarios y de integración** de forma sencilla, con una sintaxis clara y funcionalidades integradas como *mocking* y *coverage reports*, lo que garantiza la calidad y el correcto funcionamiento del código.

3. Convención de nomenclatura: snake_case

Se adoptó la convención **snake_case** para nombrar variables y métodos, con el objetivo de mantener **consistencia, legibilidad y coherencia** en el código.
Esta convención facilita la lectura especialmente en entornos donde predominan nombres compuestos y mejora la colaboración entre desarrolladores.

4. Envío de correos: Nodemailer

Para el envío de correos electrónicos, se implementó la librería **Nodemailer**, una de las soluciones más robustas y mantenidas en el ecosistema de Node.js.
Permite enviar mensajes a través de múltiples proveedores de correo de forma sencilla, ofreciendo soporte para autenticación, HTML, adjuntos y plantillas.
Su elección se justifica por su **simplicidad, confiabilidad y documentación completa**.

5. Generación de códigos QR: API qrserver

La generación de códigos QR se realiza mediante la **API pública de QRServer**, debido a que ofrece una forma **rápida, gratuita y sin dependencias adicionales** para generar imágenes de QR dinámicamente.
Esto reduce la carga en el servidor y facilita la integración con otros servicios, manteniendo una arquitectura ligera.

6. Frontend: Next.js, TypeScript y React

Para el frontend utilizamos **Next.js** y **TypeScrip**t. Se aplicó el desarrollo en componentes modulares para mantener un mejor orden y consistencia, además de separar más fácilmente las distintas responsabilidades del front (lógica de negocio, servicios y UI).

De **React**, se utilizaron únicamente algunas funcionalidades específicas, como **useState** y **useEffect**, destinadas al manejo de estados y efectos secundarios dentro de los componentes.

Por último, también utilizamos dos herramientas de desarrollo: **PNPM**, para gestionar de forma más eficiente los paquetes npm, y **PostCSS**, para transformar CSS con JavaScript, mejorando la compatibilidad y optimización.

Esto reduce la carga en el servidor y facilita la integración con otros servicios, manteniendo una arquitectura ligera y eficiente.