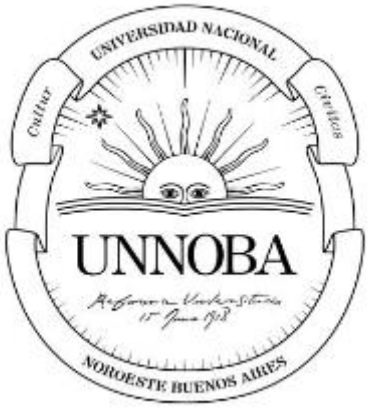

Universidad Nacional del Noroeste de la Provincia de Buenos Aires

Título: Desarrollo de Soluciones de Software Aplicadas en Entorno Profesional

Carrera: Ingeniería en informática

Práctica Profesional Supervisada

Estudiante: Guillermo Luis Casanova

Tutor Docente: Lencina Paula

Tutor de Empresa/Institución/Organización: Basile Matias

Fecha de presentación:17/06/2025

Contenido

Introducción

En el presente informe se detalla la actividad que se desarrolló dentro de la empresa Varcreative, dedicada al desarrollo de soluciones de software a medida, ofreciendo a sus clientes herramientas tecnológicas personalizadas, alineadas con los requerimientos específicos de cada uno.

El propósito del trabajo que se llevó a cabo se centra en el desarrollo y mantenimiento de sistemas, abarcando tareas de programación tanto del lado del cliente (front-end) como del servidor (back-end), la administración de bases de datos y la aplicación de buenas prácticas de desarrollo, incluyendo el uso de arquitectura hexagonal en uno de los proyectos asignados.

Objetivos

Objetivo general

Realizar una práctica profesional orientada al desarrollo y mantenimiento de soluciones de software, aplicando conocimientos técnicos y metodológicos que permitan mejorar la calidad y eficiencia de los sistemas que se implementan en la empresa Varcreative.

Objetivos específicos

- Aplicar buenas prácticas de programación en el desarrollo de funcionalidades tanto del front-end como del back-end.
- Administrar y optimizar bases de datos para garantizar un funcionamiento estable y eficiente de las aplicaciones.
- Implementar y mantener arquitecturas escalables, como la arquitectura hexagonal, en proyectos en curso.
- Documentar adecuadamente los procesos y herramientas que se utilizan durante el desarrollo.
- Colaborar con el equipo de trabajo en la resolución de problemas técnicos y en la mejora continua de los proyectos.
- Integrar tecnologías y herramientas actuales en el flujo de trabajo, evaluando su impacto en el rendimiento y la calidad del producto final.
- Participar en reuniones periódicas con los clientes para relevar requerimientos, presentar avances y alinear los desarrollos con sus necesidades.

Plan de Trabajo y Carga Horaria

El presente plan de trabajo se organiza sobre la base de un esquema continuo de actividades que se desarrollaron diariamente en el ámbito laboral. Las tareas realizadas durante la Práctica Profesional Supervisada se estructuran en tres ejes principales: reuniones técnicas, desarrollo de funcionalidades y ejecución de pruebas e implementación. Estas actividades se

llevaron a cabo de manera integrada y constante, respondiendo a las necesidades de cada proyecto.

De acuerdo al cronograma propuesto, las actividades correspondientes a la Práctica Profesional Supervisada se distribuyeron a lo largo de cinco semanas, cumpliendo con los requerimientos establecidos por la institución. Durante ese período, se mantuvo un ritmo de trabajo sostenido y coherente con la modalidad de la empresa, en la que las tareas se actualizaban a diario y se priorizaban según las demandas del equipo técnico y del cliente. Cabe destacar que, si bien la PPS se desarrolló dentro de ese marco temporal, mi participación en los proyectos continúa hasta la actualidad, formando parte de un proceso cíclico de diseño, implementación, validación y mejora continua.

Cabe destacar que no se presentaron desfases con lo planificado inicialmente, ya que el entorno de trabajo donde se desarrolló la práctica se basa en una dinámica de trabajo diaria y estable. Esto permitía adaptarse con flexibilidad a los requerimientos del proyecto y asegurar el cumplimiento de los objetivos definidos en tiempo y forma.

En cuanto a la carga horaria, se cumplió una jornada laboral de 8 horas diarias, de lunes a viernes, lo que representa un total de 40 horas semanales dedicadas exclusivamente al desarrollo de las tareas vinculadas a los proyectos. Esta organización contribuyó a consolidar una experiencia profesional sólida, alineada con el ritmo real del desarrollo de software en un entorno profesional.

Entorno de Trabajo

La práctica profesional se llevó a cabo en la empresa Varcreative, una organización de tamaño mediano ubicada en la ciudad de Chacabuco con más de diez años de antigüedad, dedicada al desarrollo de soluciones de software tanto a nivel local como para clientes del exterior. A pesar de su estructura compacta, la empresa abarca diversas áreas del desarrollo tecnológico, incluyendo el desarrollo de aplicaciones back-end y front-end, mantenimiento y actualización de sistemas existentes, así como asesoramiento técnico a terceros.

El enfoque de trabajo se basó en la implementación de buenas prácticas de desarrollo, utilizando herramientas modernas y metodologías que permitieron mantener un flujo de trabajo ordenado y colaborativo. Entre las tecnologías que se emplearon se destacaron el uso de PHP[1] como lenguaje principal, en conjunto con bases de datos MySQL[2] y PostgreSQL[3], y el patrón de arquitectura hexagonal para estructurar los proyectos de manera clara y escalable.

Asimismo, se emplearon sistemas de control de versiones como Git[4], con repositorios alojados en Bitbucket, lo cual facilitó el trabajo colaborativo y el seguimiento de los cambios realizados en el código a lo largo del ciclo de desarrollo. Este entorno permitió no solo aplicar conocimientos técnicos, sino también desarrollar habilidades de organización, comunicación y trabajo en equipo.

En cuanto a los proyectos desarrollados durante la práctica, trabajé para dos clientes internacionales con base en España. El primero de ellos, un empresario del rubro turístico, que gestionaba una plataforma de venta de experiencias que incluía pasajes para cruceros, entradas a espectáculos, actividades recreativas y hospedajes. El segundo cliente fue una empresa con

múltiples proyectos vinculados al ámbito aeronáutico, que cuenta con un equipo especializado de arquitectos de sistemas y para la cual se desarrollaron soluciones orientadas a la gestión de vuelos y operaciones aeroportuarias.

Descripción de la Práctica Profesional Efectuada

Metodologías de Trabajo

La modalidad de trabajo se organizó en una serie de etapas bien definidas, orientadas a garantizar un desarrollo ágil, ordenado y alineado con las necesidades del cliente. Particularmente en el proyecto actual —de mayor complejidad técnica y estructural— este flujo de trabajo cobró especial importancia para sostener la calidad del código y la coordinación entre áreas.

Durante el desarrollo del primer proyecto se mantuvieron reuniones periódicas (habitualmente diarias) con el cliente y con el equipo técnico para definir prioridades, relevar nuevas necesidades y realizar seguimiento del avance. Las tareas se gestionaron a través de Trello, donde se organizó el backlog y se asignaron actividades a los distintos miembros del equipo.

Las prioridades fueron establecidas directamente por el cliente, y en función de ello se inició el proceso de desarrollo. Una vez finalizada cada funcionalidad o módulo, el código era revisado internamente por el jefe de equipo, y si corresponde, se realizaba una etapa de corrección de errores.

En el caso del proyecto actual, toda nueva funcionalidad se incorporaba mediante el envío de una Pull Request (PR), la cual era evaluada por el equipo de arquitectura de software. Solo tras su aprobación formal, los cambios se integraban al repositorio principal en promedio de un día laboral debido a la diferencia horaria entre Argentina y España. Este proceso aseguraba el cumplimiento de estándares de calidad, buenas prácticas y una estructura de código mantenible.

El trabajo se gestionaba utilizando Git para el control de versiones, con repositorios privados alojados en Bitbucket. Los canales de comunicación fueron directos, mediante Google Chat (mensajes individuales o grupales), y se reforzaron en las reuniones periódicas donde se revisaban avances y se validaban decisiones técnicas.

Proyectos Asignados

Durante la práctica profesional, el trabajo se dividió en dos grandes etapas, definidas por la participación en dos proyectos diferentes dentro de la empresa Varcreative. Cada uno de estos proyectos presentaron desafíos particulares, tanto a nivel técnico como organizacional, y permitieron aplicar distintos enfoques de desarrollo.

Primer proyecto: Plataforma de gestión de entretenimiento y turismo

En la primera etapa, se trabajó en un proyecto destinado a un cliente de la empresa que opera en el rubro del turismo y entretenimiento. El sistema en cuestión se encentraba dividido en

varias partes interrelacionadas, entre ellas un sistema de gestión de clientes y empresas, un BackOffice destinado a la administración y monitoreo de los viajes, y una web publica para clientes.

Dentro de mis actividades asignadas, desarrollé una parte de front-end público, accesible para usuarios finales, que permitía la compra de pasajes para cruceros, así como la adquisición de entradas para cines, espectáculos, musicales, parques y reservas en hoteles. Esta parte del sistema estuvo orientada a la experiencia del usuario, con un diseño amigable y funcionalidades claras para facilitar el proceso de reserva o compra.

Por otro lado, se tenía un BackOffice conocido como “Dashboard”, centrado en la gestión de empresas proveedoras y de los espectáculos ofrecidos. Esta sección estuvo integrada con Prestashop[5] y permitió a los administradores configurar productos, visualizar estadísticas, consultar informes gráficos (varios de los cuales desarrollé) y acceder a herramientas específicas de administración y monitoreo del sistema.

Adicionalmente, el sistema contaba con un segundo BackOffice llamado “Panel”, enfocado en la gestión específica de los viajes de cruceros. En este módulo se administraba la información de los clientes que realizaban reservas o compras de pasajes, se monitorizaba la salida y llegada de los pasajeros, se descargaban reportes personalizados (también desarrollados por mí) y se gestionaban datos de contacto e historial de actividad de los usuarios.

El enfoque arquitectónico de este primer proyecto fue más tradicional, basado en PHP con frameworks como Symfony[5] y CodeIgniter[6], estructurado bajo el modelo MVC[7] (Modelo, Vista, Controlador). Las tareas que se realizaron en esta fase incluyeron programación

back-end y front-end, mantenimiento de bases de datos, implementación de funcionalidades, ajustes visuales y contribuciones a reportes y estructuras administrativa.

Segundo proyecto: sistema de gestión de vuelos y mantenimiento

Posteriormente, se me incorporó al segundo proyecto, el cual representaba un desafío de mayor complejidad técnica. Este sistema es desarrollado para una división de una empresa que gestiona operaciones aeroportuarias en distintas ciudades del mundo, y es desarrollado en su mayoría en PHP, adoptando la arquitectura hexagonal[9-11] como base estructural.

El proyecto se centra en la gestión de vuelos que operan desde o hacia ciudades como Madrid, Lisboa o Bangkok, entre muchas otras. La solución está orientada al trabajo de los supervisores de pista en aeropuertos, quienes utilizan una aplicación móvil para controlar tareas relacionadas con la preparación de los vuelos, la identificación de demoras, la verificación de tiempos de salida, la coordinación de mantenimientos y la generación de reportes específicos.

Este sistema se divide en tres componentes principales:

Back-end, basado en PHP y Symfony, con implementación de APIs que siguen principios de desacoplamiento y escalabilidad.

Aplicación móvil, desarrollada en React Native[12], que se conecta directamente con las APIs y permite la visualización y edición de información en tiempo real desde el entorno aeroportuario.

Interfaz web, desarrollada por otro integrante del equipo, que también consume las mismas APIs para funciones administrativas.

Mis responsabilidades en este proyecto incluyeron el desarrollo de los endpoints, la administración de la base de datos PostgreSQL, el maquetado de la aplicación móvil, la integración entre los distintos sistemas, y la ejecución de pruebas para asegurar la calidad de cada nueva funcionalidad. Asimismo, participé en reuniones técnicas diarias donde se definieron prioridades, se revisaban avances y se validaban decisiones junto al equipo de arquitectura.

A diferencia del primer proyecto, este segundo sistema está diseñado desde su base con una arquitectura hexagonal que permite un mayor grado de aislamiento entre las capas del sistema, facilitando las pruebas unitarias, el mantenimiento y la escalabilidad. Esta transición desde una arquitectura MVC hacia un modelo hexagonal representó un importante crecimiento profesional, ya que me desafió a repensar la estructura de cada módulo, aplicar buenas prácticas más estrictas y trabajar en entornos más complejos y exigentes.

Comparación entre ambos proyectos

Ambos proyectos presentan contextos y objetivos distintos, lo que me permitió abordar el desarrollo de software desde enfoques complementarios. El primer proyecto, más tradicional y orientado al comercio electrónico y entretenimiento, me introdujo al flujo de trabajo en equipo, la integración de múltiples tecnologías como Prestashop y la construcción de herramientas administrativas orientadas al cliente. Su arquitectura basada en MVC me permitió afianzar los conceptos fundamentales de separación de responsabilidades y mantener una lógica de desarrollo más lineal.

Por su parte, el segundo proyecto implicó un nivel técnico más avanzado, tanto en lo conceptual como en lo práctico. La adopción de la arquitectura hexagonal exigió una mayor

rigurosidad en la estructuración del código, así como un trabajo más profundo sobre la lógica del dominio y su aislamiento frente a tecnologías externas. Además, el contexto crítico en el que se utiliza (operaciones aeroportuarias en tiempo real) implica una mayor responsabilidad sobre la calidad, estabilidad y escalabilidad del sistema.

Esta transición entre ambos proyectos representó un salto significativo en mi formación como desarrollador, consolidando habilidades técnicas y metodológicas que son fundamentales para asumir con solvencia nuevos desafíos profesionales.

Tareas Realizadas

Durante la práctica profesional (de aquí en adelante segundo proyecto asignado), se realizaron múltiples tareas de desarrollo tanto en el área de back-end como en el front-end, además de otras actividades complementarias vinculadas a la documentación, control de versiones y reuniones técnicas.

Back-End

El desarrollo del lado del servidor se realizó principalmente en PHP, utilizando el framework Symfony y adoptando una arquitectura hexagonal. Esta arquitectura permite estructurar el software en capas desacopladas, facilitando la mantenibilidad, escalabilidad y testeo del sistema.

Capas Principales

La arquitectura en capas tradicional organiza el sistema en una serie de capas jerárquicas, como presentación, lógica de negocio y acceso a datos. Cada capa depende de la inmediatamente inferior, lo que puede generar un acoplamiento estrecho y dificultades para realizar pruebas aisladas o cambios en componentes específicos.

Por otro lado, la arquitectura hexagonal, también conocida como arquitectura de puertos y adaptadores, propone una separación clara entre el núcleo de la lógica de negocio y los elementos externos, como bases de datos, interfaces de usuario o servicios de terceros. Esta estructura permite una mayor flexibilidad, facilidad para realizar pruebas y una mejor mantenibilidad del sistema, ya que cada componente cumple un rol específico sin generar dependencias rígidas.

En la arquitectura hexagonal, el núcleo de la aplicación interactúa con el exterior a través de "puertos", que son interfaces abstractas. Los "adaptadores" implementan estos puertos para comunicarse con tecnologías específicas, como bases de datos o interfaces de usuario. Este enfoque facilita la sustitución o modificación de componentes externos sin afectar la lógica central del sistema.

A continuación, se presenta una imagen que ilustra la arquitectura hexagonal según Alistair Cockburn:

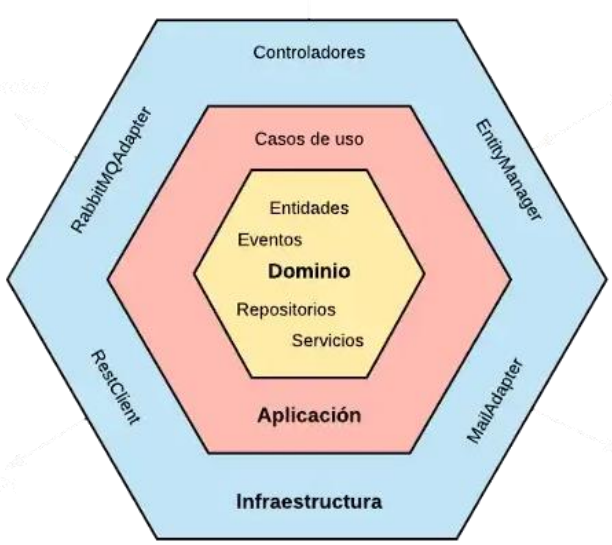

Imagen 1: Arquitectura hexagonal (puertos y Adaptadores)

En nuestro proyecto, adoptamos esta arquitectura hexagonal, implementando las siguientes capas:

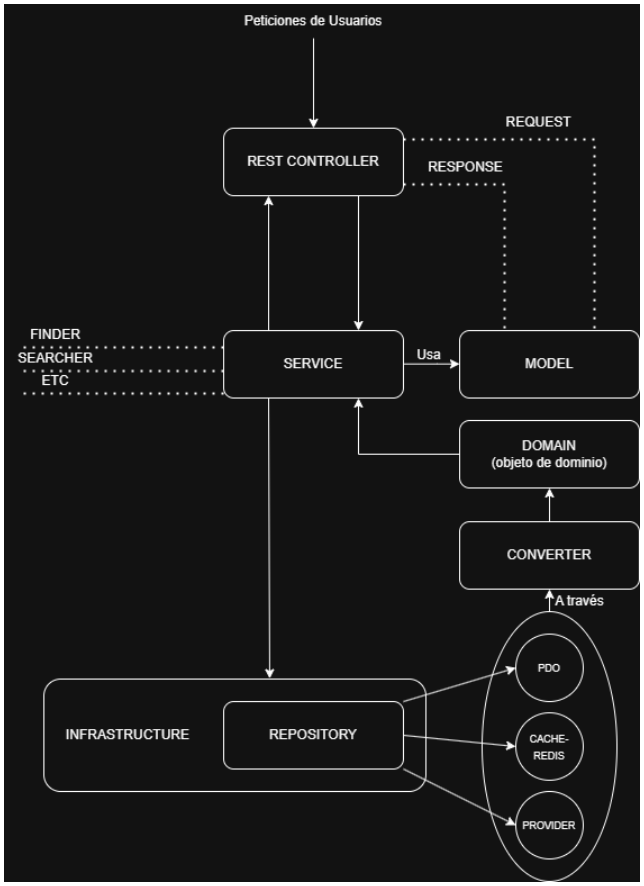

Imagen 2: Arquitectura hexagonal aplicada en el proyecto

La capa del controlador actúa como punto de entrada del sistema. Su función principal es manejar las solicitudes del usuario, habitualmente a través del protocolo HTTP, utilizando métodos como GET, POST, PUT o DELETE, y delegar el procesamiento a los servicios correspondientes. En este sentido, sirve de intermediario entre el mundo exterior y la lógica interna del sistema.

La capa de servicio, o capa de aplicación, contiene la lógica de negocio aplicada. Aquí se definen los casos de uso y se orquesta el comportamiento del sistema, coordinando la interacción entre modelos, convertidores y entidades del dominio. Esta capa determina lo que el sistema puede hacer, sin preocuparse por cómo se ejecutan esas acciones a nivel técnico.

Los convertidores tienen la responsabilidad de transformar datos primitivos, como strings o arrays, en objetos del dominio, y viceversa. De esta manera, se establece una conversión clara y controlada entre las estructuras internas y externas, asegurando la integridad de los datos y evitando el acoplamiento entre capas.

Por su parte, la capa de modelo representa las estructuras de datos utilizadas para transportar información entre capas. Generalmente adoptan la forma de objetos de transferencia de datos (DTOs), y no contienen lógica de negocio. Su propósito es facilitar la comunicación eficiente sin introducir dependencias innecesarias.

La capa de dominio representa el núcleo funcional del sistema y se enfoca exclusivamente en el modelo de negocio. En ella se definen las entidades, interfaces y reglas que determinan el comportamiento fundamental de la aplicación. Esta capa se mantiene completamente aislada de tecnologías externas, frameworks o herramientas de infraestructura, lo que permite preservar una lógica de negocio limpia, coherente y fácilmente adaptable a distintos contextos tecnológicos.

Por ejemplo, una interfaz como EntidadRepository se define en esta capa para establecer las operaciones que deben poder realizarse sobre una entidad del sistema, sin preocuparse por cómo serán implementadas técnicamente. Este nivel de abstracción permite desacoplar las decisiones de infraestructura del núcleo de la lógica de negocio, facilitando así la evolución y mantenimiento del sistema.

Por su parte, la capa de infraestructura se encarga de implementar los mecanismos técnicos necesarios para conectar la lógica de negocio con el mundo externo. Esto incluye, por ejemplo, el acceso a bases de datos, servicios externos, sistemas de archivos o redes.

En esta capa se ubican las implementaciones concretas de las interfaces definidas en el dominio. Tal es el caso de la clase que implementa EntidadRepository, que utiliza SQL nativo, convertidores y conexiones a través de PDO para recuperar o persistir datos. De este modo, la infraestructura cumple con los contratos establecidos en el dominio, manteniendo la lógica de negocio aislada de los detalles técnicos específicos y garantizando una arquitectura escalable y modular.

Para la interacción con la base de datos se utiliza PDO de forma explícita, combinando SQL nativo y métodos estructurados, como se muestran en los siguientes ejemplos:

```
public function find(int $param1, string $param2): ?object
{
$query = <<<NATIVE_SQL
        SELECT
```

```
            T.value1,
            T.value2,
            T.value3
        FROM schema.table_name T
        WHERE value1 = :param1
        AND value2 = :param2
    NATIVE_SQL;
    $result = $this->execute(
        $query,
        [
            new PDOParam(':param1', DataType::INT, $param1),
            new PDOParam(':param2', DataType::STRING, $param2),
        ],
        $this->converter
    );
    return Arr::first($result);}
```

Imagen 3: Ejemplo búsqueda de un elemento para el servicio "find"

```
public function update(object $entity): void
{
    $query = <<<NATIVE_SQL
        UPDATE schema.table_name
        SET
            value2 = :param2,
            value3 = :param3
```

```
        WHERE value1 = :param1
    NATIVE_SQL;
    $this->execute(
        $query,
       [new PDOParam(':param1', DataType::INT, $entity->param1()),
     new PDOParam(':param2', DataType::STRING, $entity->param2()),
       new PDOParam(':param3', DataType::BOOL, $entity->param3()),
        ]
    );
}
```

Imagen 4: Ejemplo actualización de un elemento

Además del desarrollo de las APIs, también se trabaja en su integración tanto con la app móvil como con el sistema web.

Otras tareas complementarias

Además de las actividades técnicas directamente relacionadas con el desarrollo de funcionalidades, durante la práctica profesional también se llevaron a cabo tareas complementarias esenciales para sostener la calidad, organización y coherencia del proyecto. Estas se agrupan en tres bloques temáticos:

Documentación y estándares

Con cada PR aprobada, se acompaña la funcionalidad desarrollada con una descripción técnica y funcional, habitualmente registrada en documentos internos. Este enfoque permite dejar

trazabilidad sobre los cambios y facilita el mantenimiento y la comprensión del sistema por parte del equipo. A la fecha, se han desarrollado y aprobado 25 integraciones a través de este proceso, lo que refleja el uso sistemático y consistente del flujo de trabajo definido.

Estandarización de convenciones: uso de nombres descriptivos, patrones de arquitectura definidos por el equipo técnico y validación de estilo mediante herramientas de análisis.
Esta sistematización reduce el tiempo de incorporación de nuevos desarrolladores y facilita el mantenimiento evolutivo del sistema.

Comunicación y coordinación

Reuniones técnicas diarias**:** revisión de avances, validación de soluciones y reasignación de tareas según prioridades.

Canales de comunicación: Google Chat para consultas inmediatas y Trello para la gestión del backlog y el seguimiento del flujo de trabajo.
Esta combinación de herramientas y reuniones permitieron resolver bloqueos en menos de 3 o 4 horas en promedio y mantiene al equipo alineado con los objetivos del proyecto.

Tal como se menciona en “Principales Dificultades”, el rigor en los tests y el control de calidad han sido clave para superar la curva de adopción de la arquitectura hexagonal. Estas tareas complementarias refuerzan las buenas prácticas de trabajo en equipo y desarrollo de software seguro y mantenible.

El desarrollo del sistema se organizó de manera colaborativa, dividiendo responsabilidades entre distintos miembros del equipo según su área de especialización. Esta

modalidad permite un flujo de trabajo más ágil y ordenado, donde cada componente del sistema es abordado por quienes tienen mayor dominio sobre la tecnología o la parte funcional correspondiente.

En este contexto, mi trabajo se centró principalmente en el desarrollo de APIs, las cuales implementé junto a mi jefe directo. Estas APIs funcionan como núcleo de comunicación y lógica entre los distintos componentes del sistema, siendo consumidas tanto por la aplicación móvil como por la interfaz web.

Además, se me asigno el cargo del desarrollo de la aplicación móvil, tomando como base un trabajo previo realizado por otro desarrollador. En este proyecto, soy responsable tanto de la parte de back-end (lógica, integración con APIs y estructura de datos), como del front-end (maquetado, interfaz y experiencia de usuario). La app interactúa directamente con las APIs que desarrollé, permitiendo gestionar tareas relacionadas con vuelos, mantenimientos y reportes operativos, especialmente orientadas a supervisores de pista en aeropuertos.

Por otra parte, existe una plataforma web que también consulta y consume las APIs implementadas. Esta interfaz web tiene un enfoque más administrativo, permitiendo visualizar, consultar y gestionar la misma información tratada por la app. El desarrollo del front-end web no está a mi cargo, sino que es realizado por otro integrante del equipo, especializado en tecnologías de diseño y desarrollo de interfaces. Esta parte se construye principalmente con JavaScript, utilizando TypeScript junto con el framework Vue 3, y herramientas modernas como Vite, Tailwind CSS, DaisyUI y otras librerías orientadas a la gestión de datos, y pruebas automatizadas.

Control de calidad y versión

Gestión de ramas: Se mantiene un máximo de 10 ramas remotas activas, las cuales se eliminan automáticamente al cerrarse cada pull request. A nivel local, se conservan también hasta 10 ramas, que se limpian manualmente una vez finalizada cada tarea. Esta organización permite reducir los conflictos de merge y agiliza las integraciones, contribuyendo así a un flujo de trabajo ordenado y sostenible.

Testing riguroso: en el flujo de trabajo del proyecto actual, cada nueva funcionalidad debe ser probada localmente antes de ser enviada en una Pull Request (PR). Posteriormente, el código se somete a un análisis automatizado mediante Sonar[13], ejecutado desde Bitbucket. Este análisis evalúa calidad del código, errores potenciales y, en particular, la cobertura de pruebas. Para que el código pueda ser considerado (revisado) por el equipo de arquitectura, debe alcanzar una cobertura mínima del 80%, garantizando que la lógica desarrollada esté debidamente testeada y cumpla con los estándares de calidad. Este proceso completo puede demorar hasta 10 minutos.

Gracias a esta organización y limpieza continua de ramas, se evitan conflictos de merge y se aceleran las integraciones, garantizando la calidad del código.

A continuación de describe un ejemplo de un test sobre un servicio de manera genérica.

```
final class EntityUpdaterServiceTest extends TestCase
{
```

```
    private EntityUpdaterService $service;

    private EntityRepository|MockInterface $repository;

    protected function setUp(): void

    {

        parent::setUp();


      $this->repository = \Mockery::mock(EntityRepository::class);

        $this->service = new EntityUpdaterService(

            $this->repository

        );

    }

    public function test_it_should_update_an_entity(): void

    {

        $request = $this->createRequest();

        $expectedEntity = $this->createEntity();

        $this->repository

            ->shouldReceive('find')

            ->once()

            ->with($request->param1, $request->param2)

            ->andReturn($expectedEntity);

        $this->repository

            ->shouldReceive('update')

            ->once()

            ->with($expectedEntity);

        $this->service->update($request);

    }
```

```
    public function test_it_should_throw_exception_when_entity_not_found(): void
    {
        $request = $this->createRequest();

        $this->repository
            ->shouldReceive('find')
            ->once()
            ->with($request->param1, $request->param2)
            ->andReturn(null);
        $this->expectException(EntityNotFoundException::class);
        $this->service->update($request);
    }
    private function createRequest(): EntityRequest
    {
        // Crear un request dummy. Adaptar según datos reales.
        return new EntityRequest(param1: 1, param2: 'abc', otherParam: true);
    }
    private function createEntity(): Entity
    {
        // Crear una entidad dummy. Adaptar según el dominio real.
        return new Entity(1, 'abc', true);
    }
}
```

Imagen 5: Ejemplo de Test

En este Test Unitario se logra observar:

- Se prueba que el servicio llama correctamente al repositorio (find + update).

- Se simula (mockea) el repositorio sin necesidad de base de datos.
- Se asegura que se lanza una excepción si la entidad no existe.
- Se separan claramente los datos de entrada (Request) y los datos del dominio (Entity).

Esta instrucción configura el comportamiento esperado de un método simulado (mock) dentro del test. En este caso, se está indicando que se espera que el método find() del repositorio simulado sea llamado exactamente una vez (once()), con los (with()) parámetros `$request->param1, $request->param2`, y que, al hacerlo, retorna (`andReturn()`) el contenido `$expectedEntity`.

Esta técnica permite verificar que el servicio bajo prueba (por ejemplo, un FinderService) interactúa correctamente con el repositorio, sin necesidad de acceder a una base de datos real. Además, asegura que se están pasando los parámetros correctos y que la respuesta simulada se maneja como se espera en el flujo de ejecución.

Una vez codificados los tests de los servicios correspondientes, se debe realizar una prueba local ejecutando los tests unitarios mediante PHPUnit, el framework de pruebas utilizado. Para ello, puede emplearse un comando como:

```
docker exec -it <nombre-del-contenedor-php> php vendor/bin/phpunit
```

Este paso asegura que el código cumple con el comportamiento esperado antes de continuar. Si todos los tests pasan correctamente, se crea una pull request (PR) para su revisión.

Previamente a su validación final, la PR debe superar los tests automáticos y cumplir con los criterios de calidad definidos en SonarQube, incluyendo los umbrales de cobertura previamente establecidos. Todo este proceso se ejecuta automáticamente dentro de Bitbucket al activarse un pipeline cada vez que se sube una PR asociada a una rama, que generalmente sigue el formato feature/nombre-del-desarrollo.

Las buenas prácticas constituyen uno de los pilares fundamentales en el desarrollo de soluciones de software. En particular, dentro del proyecto en el que me encuentro trabajando, estas prácticas son promovidas y supervisadas activamente por el equipo de Arquitectura de Sistemas.

El seguimiento constante de los arquitectos no solo se limita a la revisión técnica de las tareas realizadas, sino que también implica la aprobación formal de cada desarrollo, o bien su devolución con observaciones y sugerencias para su mejora. Esta dinámica de trabajo garantiza que el código entregado cumpla con estándares de calidad, escalabilidad, seguridad y mantenibilidad.

Dado que se trata de un proyecto de escala mediana, se busca mantener una línea de desarrollo uniforme entre todos los miembros del equipo. Para ello, se hace hincapié en la aplicación coherente de buenas prácticas a lo largo de todo el ciclo de vida del desarrollo.

A continuación, se detallan algunas de las buenas prácticas aplicadas durante la experiencia profesional, acompañadas de ejemplos concretos.

Entre las buenas prácticas aplicadas durante el desarrollo del proyecto, se destaca el uso de Git como herramienta de control de versiones. Git permite llevar un seguimiento detallado y ordenado de los cambios realizados en el código, facilita el trabajo colaborativo y evita conflictos al integrar desarrollos de distintos miembros del equipo. Gracias a su uso, es posible mantener un historial de versiones, revertir errores y organizar el código en ramas que representan diferentes funcionalidades o correcciones.

Otra práctica fundamental es la separación rigurosa de cada capa dentro de la arquitectura hexagonal. Este enfoque permite desacoplar la lógica de negocio del resto de las dependencias del sistema, generando un código más limpio, flexible y mantenible. Dentro de esta estructura se aplican lineamientos como realizar únicamente consultas SQL sobre los elementos necesarios, evitando el uso de "SELECT *" y optando por la selección explícita de campos. Esto contribuye a mejorar el rendimiento y la claridad del código.

También se procura que los nombres de los servicios sean descriptivos y reflejen con claridad su responsabilidad, lo cual mejora la legibilidad general del sistema y facilita la comprensión por parte de otros desarrolladores. Además, se busca respetar los principios SOLID durante el desarrollo, en particular el principio de responsabilidad única, asegurando que cada servicio esté enfocado en cumplir una sola tarea específica. Esto promueve un diseño más modular, mantenible y fácil de testear.

En cuanto al uso de enumeraciones (enum), se promueve su implementación en casos donde un conjunto limitado de valores posibles debe ser representado, como por ejemplo en la definición de tipos de logs que pueden ser "insert" o "update". Esto permite validar y controlar mejor los datos, evitando errores por valores inesperados.

Se destaca asimismo la conversión de tipos primitivos a objetos del dominio, lo cual permite encapsular lógica y validaciones propias del negocio dentro de entidades robustas, mejorando la coherencia y reutilización del código. Otro principio promovido es limitar la cantidad de parámetros en las funciones a un máximo de siete, un máximo de 120 caracteres por línea de código y no más de tres returns en métodos. Este criterio, establecido por el equipo de arquitectura, busca mejorar la legibilidad, simplificar el testing y reducir el acoplamiento entre componentes.

Se evita la duplicación de código mediante su refactorización en funciones reutilizables, aunque procurando no fragmentar en exceso para no dificultar la comprensión del flujo. Para casos en los que una función requiere más de siete parámetros, se sugiere el uso de builders como patrón para construir objetos de forma clara, sin sobrecargar la lógica del código principal.

Una recomendación importante es la no utilización de comentarios como método para explicar código innecesariamente complejo. En su lugar, se promueve escribir código limpio, autoexplicativo y con nombres significativos. Del mismo modo, se evita declarar variables que no serán utilizadas, lo cual reduce el ruido y mejora la eficiencia. También se sugiere retornar directamente el resultado de una función, por ejemplo, return toEntity($arrayExample); en lugar

de almacenar primero el valor en una variable y luego retornarla, lo cual simplifica la lógica sin perder claridad.

La utilización de nombres descriptivos y acordes a cada servicio resulta fundamental. Por ejemplo, en aquellos servicios que se encargan de obtener múltiples registros desde la base de datos —como puede ser un listado de usuarios—, se utiliza la siguiente estructura: UserSearcherService, siguiendo el formato NombreDeLaEntidadAccionService. Esta convención normaliza la estructura de todos los archivos dentro del proyecto y facilita su identificación y mantenimiento.

```
<?php
declare(strict_types=1);
namespace App\Service\Entity;
use App\Domain\Entity\EntityRepository;
use App\Converter\Entity\EntityToResponseConverter;
use App\Model\Response\Entity\EntityResponse;

class EntitySearcherService
{
    public function __construct(
        private readonly EntityRepository $repository,
        private readonly EntityToResponseConverter $converter
    ) {}
```

```
    /** @return EntityResponse[] */
    public function search(string $param): array
    {
        $result = $this->repository->search($param);
        return array_map($this->converter, $result);
    }
}
```

Imagen 6: Ejemplo Estructura básica y nomenclatura de un servicio

Lo mismo se aplica a los archivos de prueba (tests), donde se utiliza una nomenclatura como UserSearcherServiceTest, es decir, NombreDeLaEntidadAccionServiceTest.

Por último, se recomienda el uso de readonly en los objetos del dominio para declarar propiedades inmutables. Esto asegura que los valores definidos al momento de la construcción del objeto no sean alterados durante su ciclo de vida, fortaleciendo la integridad del modelo de negocio.

Principales Dificultades

Durante el desarrollo de la práctica profesional se presentaron diversas dificultades, principalmente vinculadas a la adaptación técnica y metodológica a un entorno de trabajo más riguroso y estructurado. Una de las principales barreras iniciales fue ajustarse a la arquitectura hexagonal, la cual posee una división clara y estricta de responsabilidades entre capas. Esto representó un contraste significativo con experiencias previas en proyectos más desestructurados, como ocurrió en el primer proyecto descrito

Otra dificultad fue la necesidad de seguir los lineamientos definidos por los arquitectos de software, quienes establecen cómo debe escribirse el código para que cumpla con criterios de limpieza, escalabilidad, seguridad y funcionalidad. Esta forma de trabajo implicó no solo aplicar buenas prácticas, sino también desarrollar un mayor criterio técnico al momento de estructurar soluciones.

El cumplimiento de los tiempos de desarrollo fue otro aspecto desafiante, especialmente debido a las constantes actualizaciones y mejoras requeridas por el proyecto. Para abordar estos obstáculos, se adoptó una metodología de mejora continua, tomando como base el análisis de otros proyectos ya realizados, así como consultas frecuentes a los arquitectos y revisión de documentación de compañeros y programadores anteriores.

Estas experiencias derivaron en importantes aprendizajes: comprender en mayor profundidad los fundamentos de la arquitectura hexagonal, mantener una consistencia estructural y de estilo a lo largo de todo el código y el proyecto, así como aprender a delegar responsabilidades adecuadamente entre capas, lo cual facilita el mantenimiento y evolución del sistema.

Contribuciones al Proyecto

A lo largo de mi participación en el proyecto, pude realizar diversos aportes significativos tanto en la continuidad de desarrollos existentes como en la implementación de nuevas funcionalidades. Se desarrollaron nuevas soluciones que no estaban implementadas previamente, así como se dio continuidad a partes del sistema que ya se encontraban en funcionamiento, adaptándolas y mejorándolas.

Además del trabajo técnico, también se realizaron algunas sugerencias vinculadas tanto al diseño visual como a la organización del trabajo en equipo. Durante la primera etapa de la práctica, se propusieron mejoras en distintos apartados visuales del sistema web, como ajustes en la disposición de secciones, creación de informes más claros y recomendaciones relacionadas con la experiencia de usuario. Asimismo, se sugirió la incorporación de herramientas de organización como Trello para la gestión de tareas y Slack para la comunicación del equipo, propuestas que fueron bien recibidas y adoptadas en el entorno de trabajo.

En el proyecto actual, si bien las oportunidades de sugerencias han sido más acotadas debido a la participación de una diseñadora encargada específicamente del aspecto visual, se realizaron observaciones puntuales relacionadas con la maquetación, como la eliminación de funcionalidades que no se utilizan, el ajuste de tamaños de texto o la mejora de algunos elementos visuales. Si bien no todas fueron implementadas, se valoró el interés por contribuir a la mejora continua del sistema respetando los roles asignados dentro del equipo.

Si bien el producto aún se encuentra en desarrollo activo, ya se han publicado versiones estables (actualmente versión 2), lo que demuestra un avance sostenido y una madurez creciente del proyecto. Esto ha sido posible, en gran parte, gracias a que la arquitectura adoptada permite un sistema modular y escalable, capaz de incorporar nuevas funcionalidades sin comprometer la estabilidad del software existente.

Conclusión

Como conclusión, y mirando en retrospectiva todo lo trabajado, puedo afirmar que he adquirido nuevos conocimientos no solo en lo técnico y programático, sino también en metodologías de trabajo, prácticas colaborativas y experiencias propias del entorno informático real. Esto incluye tanto el trato con clientes como la interacción fluida con distintos miembros del equipo —ya sean internos o externos—, todos enfocados en alcanzar soluciones de software a medida según los requerimientos específicos de cada proyecto.

Además, el proceso de aprendizaje relacionado con la arquitectura hexagonal y sus buenas prácticas me permite proyectar su aplicación en futuros desarrollos, incorporando desde el inicio una estructura más clara, escalable y mantenible.

Por último, haciendo un análisis general sobre los proyectos en los que participé, considero que logré adaptarme a las distintas dinámicas de trabajo y reflejar, en la práctica, uno de los pilares fundamentales de la empresa: desarrollar soluciones de software personalizadas y orientadas a resultados adaptados al cliente.

Bibliografía

[1] PHP Documentation, “PHP Manual,” php.net, última actualización 11-jun-2025. [En línea]. Disponible en: https://www.php.net/manual/es/.
[2] MySQL, “MySQL 8.4 Reference Manual,” Oracle, documento generado el 10-jun-2025. [En línea]. Disponible en: https://dev.mysql.com/doc/refman/8.4/en/.
[3] PostgreSQL Global Development Group, “PostgreSQL 17.5 Documentation,” postgresql.org, mayo 2025. [En línea]. Disponible en: https://www.postgresql.org/docs/17/.
[4] GIT Documentation, “GIT DOC”, última actualización 10-ene-2025. [En línea]. Disponible en: https://git-scm.com/doc.
[5] PrestaShop, “PrestaShop 9 Developer Documentation,” última actualización 12-jun-2025. [En línea]. Disponible en: https://devdocs.prestashop-project.org/.
[6] Symfony SAS, “Symfony 7.3 Documentation,” symfony.com, versión estable junio 2025. [En línea]. Disponible en: https://symfony.com/doc/current/index.html.
[7] CodeIgniter Foundation, “CodeIgniter 4.6.1 — User Guide,” CodeIgniter.com, 2-may-2025. [En línea]. Disponible en: https://codeigniter.com/user_guide/
[8] GeeksforGeeks, “MVC Design Pattern,” GeeksforGeeks, 3-ene-2025. [En línea]. Disponible en: https://www.geeksforgeeks.org/mvc-design-pattern/
[9] A. Cockburn, “Hexagonal Architecture,” Alistair.Cockburn.us, 4-sep-2005. [En línea]. Disponible en: https://alistair.cockburn.us/Hexagonal+architecture.
[10] Secture, “Arquitectura Hexagonal 101,” Secture.com, publicación 2023-2024. [En línea]. Disponible en: https://secture.com/arquitectura-hexagonal-101/.
[11] E. Novoseltseva, “¿Qué es la arquitectura Hexagonal? Definición y Ejemplos,” Apiumhub, 20-abr-2020. [En línea]. Disponible en: https://apiumhub.com/es/tech-blog-barcelona/arquitectura-hexagonal/.
[12] React Native, “React Native · A framework for building native apps using React,” React Native, 14-abr-2025. [En línea]. Disponible en: https://reactnative.dev/docs/getting-started
[13] SonarSource, “SonarQube Server Documentation – latest (2025.3),” SonarSource, 29-may-2025. [En línea]. Disponible en: https://docs.sonarsource.com/sonarqube-server/latest/

Agradecimientos

En primer lugar, agradezco profundamente a mi familia, por su respaldo incondicional, su paciencia y por acompañarme en cada etapa de este camino académico y profesional. Sin su apoyo, esta experiencia no habría sido posible. Gracias a mis padres, Guillermo Luis Casanova y Silvia Noemí Molina.

Asimismo, agradezco la Universidad Nacional del Noroeste de la Provincia de Buenos Aires, por brindarme la oportunidad de formarme académicamente y acompañarme durante estos años de estudio. Esta institución ha sido fundamental en mi crecimiento personal y profesional.

Extiendo también mi agradecimiento a mi docente tutora, Paula Lencina, por su acompañamiento, orientación y compromiso durante todo el proceso de práctica profesional. Su guía fue clave para integrar el aprendizaje académico con la experiencia en el campo laboral.

Deseo expresar mi profunda gratitud a la empresa Varcreative, por permitirme realizar la práctica profesional en un entorno de trabajo real, dinámico y enriquecedor. En especial, agradezco a cada uno de los integrantes del equipo con quienes compartí el día a día, por su apoyo, predisposición y enseñanzas constantes.

Agradezco especialmente a Matías Basile y Adrián Sosa, fundadores de la empresa, por abrirme las puertas de Varcreative, brindarme la posibilidad de aprender y continuar creciendo como profesional. También por acompañarme en el día a día, y estar presentes ante cualquier inquietud o situación que se presentara.

Mi agradecimiento también a Matías Schettino, Analista de Sistemas y desarrollador, por su permanente predisposición a guiarme, enseñarme distintas formas de trabajo en los proyectos, y por su atención constante ante cualquier desafío técnico o necesidad.

Además, quiero agradecer a todos mis compañeros de equipo, ya que gracias a sus consejos, ejemplos y colaboración pude — y sigo teniendo la oportunidad de— aprender y crecer en este rubro. Gracias: Elías Centurión, Mariano Garialdi, Martín Nicolás Niz, Bautista Igarzabal, Nicolás García, Valentín Oyhamburu y Leon Stéffano.

Agradezco también a mis familiares, amigos, compañeros de curso y profesores, cuyo apoyo y acompañamiento resultaron fundamentales para completar esta etapa con compromiso y entusiasmo.

MORAN Sebastian
Administrativo
29/07/2025 09:59
hora estándar de Argentina