# Pasos seguidos en el desarrollo del proyecto

## Fase 1:

### Análisis de requisitos

**Duración:** 2 dias

**Desarrollo:**

Para el desarrollo de la aplicación se va a necesitar de unos conocimientos previos en JavaScript, base de datos y acceso a datos, y conocimientos previos acerca del funcionamiento de la empresa y de como se realizan los cálculos de recursos, costes, beneficios y márgenes.

Además, para poder realizar la aplicación de escritorio se va a utilizar Electron, y como framework para el diseño de la interfaz se utilizará tambien Bootstrap.

### Diseño de arquitectura.

**Duración:** 2 semanas

**Desarrollo:**

Explicación de como va a funcionar la aplicación y que elementos necesitamos

Estructura de carpetas [explicar estructura de carpetas]

Para la creación del proyecto el primer paso será **instalar node** mediante comando

npm init

A continuación, **instalaremos electron** para poder hacer que la aplicación sea de escritorio:

npm install –save electron

También instalamos otro modulo de node que nos ayudará a no estar cerrando y ejecutando el programa cada vez que hagamos una modificiación:

npm install electron-reload

Una vez instalado electron creamos **main.js** que será el punto de entrada de Electron, y se encarga de manejar las páginas web que vamos a mostrar de nuestra aplicación desplegándolas en ventanas de navegador.

const { app, BrowserWindow } = require("electron");

let mainWindow; *// Variable global para la ventana principal*

*// Función que crea la ventana principal*

function createWindow() {

    mainWindow = new BrowserWindow({

        width: 800,

        height: 600,

        webPreferences: {

            nodeIntegration: true,

            contextIsolation: false *//Permite que window y otros objetos del contexto de Electron puedan interactuar directamente con Node.js en el renderer*

        },

    });

*//quita menú por defecto de chromium*

    mainWindow.setMenu(null);

*// Carga el archivo HTML en la ventana*

    mainWindow.loadFile("./src/views/index.html");

}

*//Codigo para lanzar la pagina principal y para cerrar app cuando se cierre la ventana*

app.whenReady().then(() => {

    createWindow();

*//Este código cierra completamente la aplicación cuando todas las ventanas han sido cerradas, excepto en macOS*

    app.on("window-all-closed", () => {

        if (process.platform !== "darwin") app.quit();

    });

*/\**

*process.platform es una propiedad de Node.js que devuelve el sistema operativo en el que se ejecuta la aplicación.*

*"win32" → Windows*

*"linux" → Linux*

*"darwin" → macOS*

*si es darwin no llamamos a app.quit() para seguir la convencion de mac y dejar la aplicacion en segundo plano*

*\*/*

*// En el caso de Mac, si la aplicación se queda en segundo plano y se reactiva este evento que se dispara*

    app.on("activate", () => {

*// Si no hay ventanas abiertas, creamos una nueva*

        if (BrowserWindow.getAllWindows().length === 0) {

            createWindow();

        }

    });

});

*//Codigo para que funcione el paquete electron-reload (solo cuando esté en desarrollo, no cuando esté en producción)*

*//if para saber si estamos en entorno distinto al de producción (desarrollo)*

if (process.env.NODE\_ENV !== 'production') {

    require('electron-reload')(\_\_dirname, {

    })

}

*\*\*Para crear las ventanas que alojarán las páginas web de procesos renderer, hay que crear instancias BrowserWindow en nuestro proceso main y pasar la ubicación del archivo HTML que queremos mostrar.*

Para poder utilizar la librería de iconos de bootstrap

npm i bootstrap-icons

### Diseño del mockup

## Fase 2:

### Desarrollo del backend y conexión con la API de la empresa (4 semanas).

Configuración de la base de datos:

¿Por dónde empezar?

1. Empieza con config: Configura la base de datos en db.js y carga variables en config.js.
2. Crea models: Define cómo se almacenarán y consultarán los datos en la base de datos.
3. Crea controllers: Define la lógica de negocio que usará los modelos.
4. Configura routes: Define los endpoints de la API y conéctalos con los controladores.
5. Si necesitas APIs externas, crea services.

Configurar login:

📂 **middlewares/** almacena funciones que interceptan las solicitudes antes de llegar a los controladores.

* authMiddleware.js → Protege rutas verificando tokens de autenticación.
* errorHandler.js → Captura errores globalmente para evitar fallos en el servidor.

## Fase 3:

### Creación del frontend y dashboard interactivo (4 semanas).

## Fase 4:

### Pruebas, optimización y documentación (2 semanas).

## 1. Introducción

En la actualidad, el análisis de los recursos productivos y financieros en la empresa se realiza a través de un informe en Excel desde el cual, se extraen manualmente los datos del sistema de gestión interno de la empresa, se realizan los cálculos pertinentes mediante Excel y PowerQuery para estructurar los datos y poder, posteriormente, realizar el análisis. Este proceso, aunque funcional, presenta limitaciones en términos de automatización, escalabilidad y visualización de datos.

El presente proyecto tiene como objetivo desarrollar una aplicación web/escritorio que automatice y optimice estos cálculos, permitiendo una mejor toma de decisiones mediante un dashboard interactivo y opciones de simulación.

## 2. Objetivos del Proyecto

* **Automatizar cálculos** de trabajadores disponibles, horas productivas, costes, beneficios y márgenes empresariales.
* **Conectar la aplicación al sistema de gestión** mediante API para mantener los datos actualizados en tiempo real.
* **Incorporar un dashboard interactivo** que facilite la visualización de datos clave.
* **Permitir simulaciones** de posibles escenarios habilitando elementos de inputs manuales (ej. contratación de empleados, aumento de ausencias, cambios en facturación).
* **Creación y mantenimiento de base de datos** para los datos que actualmente no se registran en el sistema de gestión de la empresa (facturación y costes de empleados)
* **Optimizar la eficiencia y precisión** de los cálculos, reduciendo errores manuales y mejorando la accesibilidad de los datos.
* **Unificación y escalabilidad** para que sea adaptable a cualquier cliente de la empresa.

## 3. Tecnologías Utilizadas

* **Backend**: Node.js con Express.js para la lógica del servidor y la gestión de la API.
* **Base de Datos**: MySQL o MongoDB para almacenar datos de contratos, horarios y simulaciones.
* **Frontend**: Bootstrap para el diseño de la interfaz. //Segunda Fase: Implementación de Vue para aumentar interactividad
* **Visualización de Datos**: Chart.js o D3.js para gráficos interactivos.
* **Autenticación y Seguridad**: JWT para autenticación de usuarios y roles de acceso. //Segunda Fase
* **API**: Conexión con el sistema interno de la empresa para obtener datos en tiempo real.

## 4. Estructura del Proyecto

### 4.1 Backend (Node.js + Express)

* **/routes**: Definición de endpoints
* **/controllers:** Lógica de negocio para cálculos de disponibilidad y costes.
* **/models:** Definición de esquemas de la base de datos.
* **/services**: Integración con la API del sistema de gestión.
* **/middlewares:** Seguridad y validaciones. //Segunda Fase

### 4.2 Base de Datos

* **Empleados**: ID, nombre, contrato, horas asignadas, salario…
* **Timesheet(Planificación**): ID, empleado, tipo actividad, ausencia, duración…
* **Facturación**: ID, cliente, importe, mes…
* **Costes:** site, tipo de contrato, tipo de cualificación, coste hora…
* **Actividades:** nombre, campaña
* **Clientes:** id cliente
* **Ausencia:** Nombre Ausencia, site, tipo
* **Simulaciones:** parámetros modificados, resultado…

***4.3 Frontend***

* **Dashboard con gráficos** de disponibilidad, costes y beneficios.
* **Panel de simulaciones** para modificar variables clave y ver impacto.
* **Sección de reportes** con exportación a PDF/Excel. //Segunda Fase

### 4.4. Estructuración de carpetas en Visual Studio Code

**/BAZINGAPP**

│── 📂**/src**

│ │── 📂**/controllers** *# Controladores para la lógica de negocio para cada funcionalidad*

│ │ ├── empleadoController.js

│ │ ├── facturacionController.js

│ │ ├── simulacionController.js

│ │ ├── ...

│ │──📂 **/models** *# Definimos la estructura de los datos y la interacción con la base de datos*

│ │ ├── empleadoModel.js

│ │ ├── facturacionModel.js

│ │ ├── simulacionModel.js

│ │ ├── ...

│ │── 📂**/routes** *# Definición de rutas y endpoints del backend*

│ │ ├── empleadoRoutes.js

│ │ ├── facturacionRoutes.js

│ │ ├── simulacionRoutes.js

│ │ ├── ...

│ │──📂 **/services** *# Lógica para interactuar con la API del sistema de gestión de la empresa*

│ │ ├── apiService.js

│ │── 📂**/middlewares** *# Middleware para seguridad, autenticación y validaciones*

│ │ ├── authMiddleware.js

│ │ ├── errorHandler.js

│ │──📂 **/config** *# Configuración de la base de datos y variables globales*

│ │ ├── db.js

│ │ ├── config.js

│ │──📂 **/views** *# Archivos de vista (si usamos plantillas como EJS), si es muy complicado utilizaremos archivos .html*

│ │ ├── dashboard.ejs

│ │ ├── simulaciones.ejs

│ │ ├── ...

│ │──📂 **/public** *# Archivos estáticos (CSS, JS, imágenes)*

│ │ ├── /css

│ │ ├── /js

│ │ ├── /images

│ ├── app.js *# Punto de entrada de la aplicación*

│──📂 **/tests** *# Pruebas unitarias y de integración*

│──📂 **/docs**  *# Documentación del proyecto*

│── package.json *# Configuración de dependencias*

│── .env *# Variables de entorno*

│── README.md *# Descripción del proyecto*

**Diferencias clave entre usar EJS y archivos HTML estáticos:**

**- EJS**: Los archivos son procesados en el servidor, lo que te permite insertar dinámicamente datos en el HTML (por ejemplo, de bases de datos o desde el backend).

**- Archivos HTML estáticos**: Los archivos son enviados tal cual al cliente, sin ninguna lógica dinámica (aunque puedes usar JavaScript del lado del cliente para hacer que la página sea interactiva).

## 5. Metodología y Cronograma

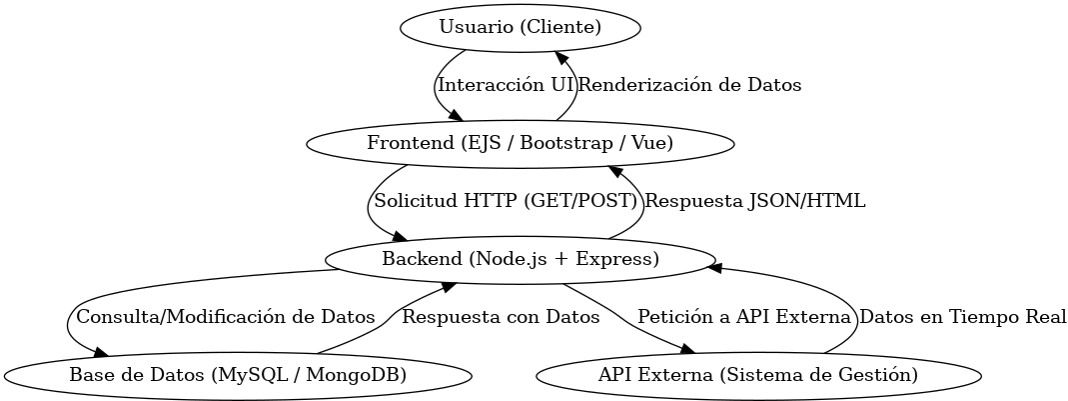
* **Fase 1:** Análisis de requisitos y diseño de arquitectura. Diseño del mockup (2 semanas).
* **Fase 2:** Desarrollo del backend y conexión con la API de la empresa (4 semanas).
* **Fase 3:** Creación del frontend y dashboard interactivo (4 semanas).
* **Fase 4:** Pruebas, optimización y documentación (2 semanas).

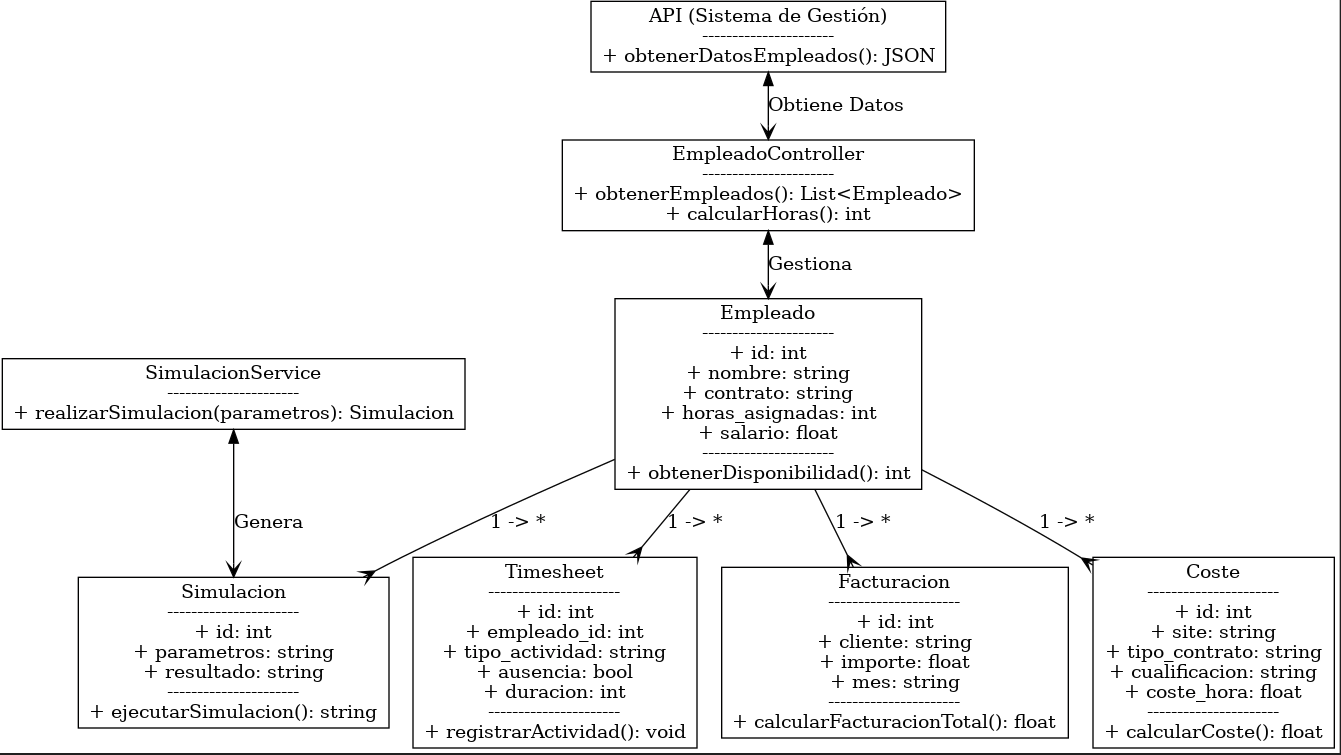
## 6. Conclusión

Esta aplicación mejorará la eficiencia en la gestión de los recursos productivos y financieros, proporcionando datos actualizados y simulaciones en tiempo real. Además, unificará el análisis productivo y financiero de la empresa al incluir las métricas de ambas partes en una única plataforma y ser escalable a todos los clientes de la empresa.

Por otro lado, este proyecto creará una base de datos estructurada y unificada para albergar los datos que actualmente financeros (costes y beneficios), los cuales, actualmente la empresa no cuenta con una base de datos para ellos. De tal forma que podrán ser accesibles y utilizados para este y otros proyectos de la empresa.

Diagrama de flujo:



Diagrama de clases: