



Documentación técnica y de usuario del backend y frontend: Sadimi

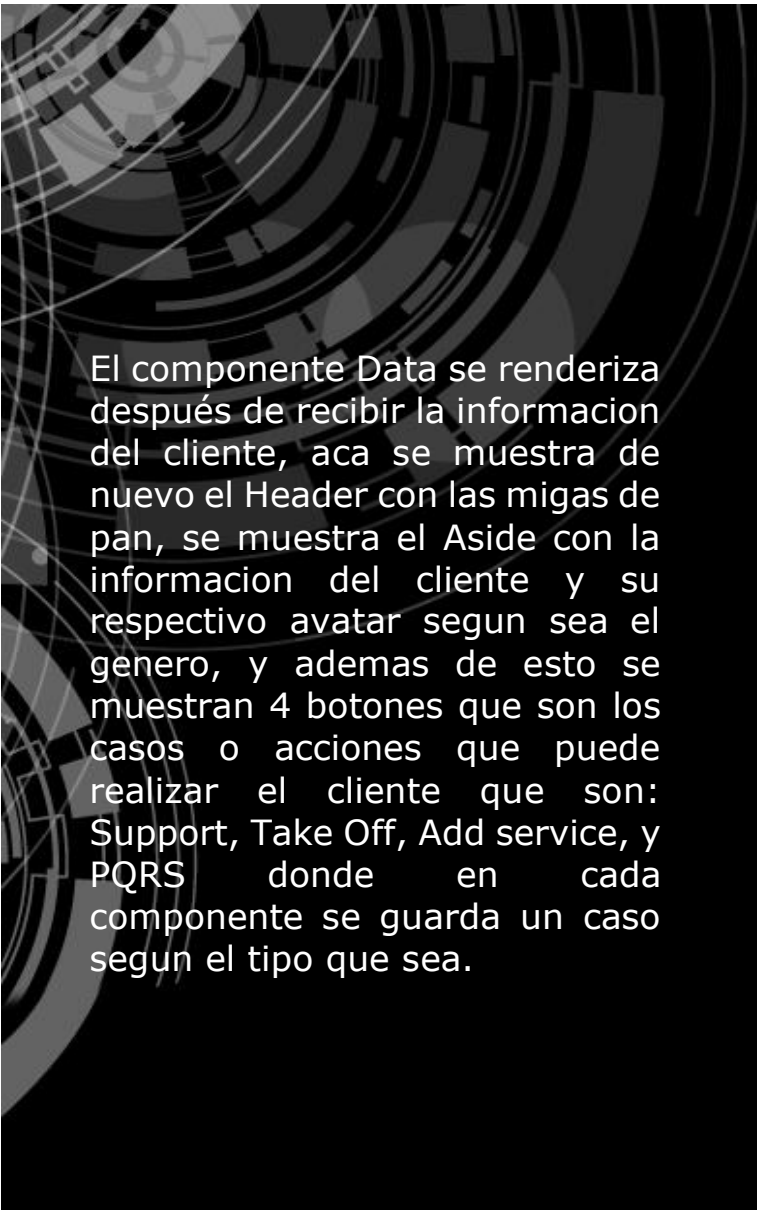
Sadimi es un CRM de tecnología el cual brinda servicios específicos a sus clientes asociados de la membresía de Microsoft. Sadimi ofrece el servicio de compra de licencias, de Xbox y computadoras. Es posible solicitar un servicio técnico a domicilio cuando un cliente lo requiera y así mismo con cada caso ocurrido, Sadimi tendrá registro de cada actividad que un cliente pueda realizar.

1 Doc. Técnica

Frontend

La aplicación se carga en el index.js que llama al app.js que es donde se cargan los componentes y se define el Router de toda la aplicación, haciendo referencia a la ruta / o ruta raíz como el componente Login que es donde se valida y se llama al empleado en la API del backend, en caso de darle olvidé mi contraseña se reutiliza el componente Form que es donde se piden los datos para cambiar la contraseña.

Después de haberse registrado correctamente se renderiza el componente Home, este contiene el Header que es otro componente que se va a reutilizar en toda la aplicación, el avatar del empleado según sea "Hombre", "Mujer" o "No binario". Debajo de este se renderiza el componente Búsqueda que es el input donde se va a pedir la cédula del usuario que está en comunicación con el CRM el cual tiene contacto con el asesor. Más abajo de este componente está el botón Search que desencadena el evento Onclick y llama a la API para verificar que el documento exista en la base de datos y posteriormente en caso de ser existente cargar el componente Data y guarda el objeto cliente en el SessionStorage para utilizarlo y no tener que estar llamando a la API para cada petición.



El componente Data se renderiza después de recibir la información del cliente, acá se muestra de nuevo el Header con las migas de pan, se muestra el Aside con la información del cliente y su respectivo avatar según sea el género, y además de esto se muestran 4 botones que son los casos o acciones que puede realizar el cliente que son: Support, Take Off, Add service, y PQRS donde en cada componente se guarda un caso según el tipo que sea.

1 Doc. Técnica

Frontend

El componente support valida la direccion y el telefono para enviar un tecnico a la direccion del cliente que se tiene en el Session Storage el cual es el mismo que se busco en el componente busqueda , si el cliente presiona aceptar se guarda el caso con typo support y con una descripcion de que se envio un tecnico a la direccion del cliente.

El componente Add service busca los servicios que tiene activos el cliente para ofrecer solo los que puede comprar que no sean de su propiedad, ejemplo si tiene Xbox, solo se le oferta License y Computer.

El componente Take Off es para cancelar el servicio que tiene el cliente y lo que hace es un patch a la API y luego en la API se elimina el servicio

El componente PQRS añade un textarea para que el cliente se desahogue y escriba sus peticiones, quejas, reclamos o sugerencias, luego se guardan los casos en la colección de casos.

Finalmente el componente AllCases que es para mostrar todos los casos que ha tenido ese cliente en donde se hace un llamado a la api con el metodo get para traerlos y posteriormente cuando se presione el boton enviar, se genera el pdf y se envia por correo al cliente.

Adicional se hizo una pagina de Error para que muestre que la ruta no es correcta y se desplego la aplicación en el servidor gratuito render:

URL: <https://sadimi-suj6.onrender.com>

2 Doc. Técnica

Backend

A continuación, se describen los componentes principales del proyecto:

El proyecto que mencionas es un repositorio de código en GitHub que contiene una aplicación backend escrita en JavaScript utilizando el framework Express.js. El propósito de la aplicación es proporcionar un API para una plataforma de comercio electrónico, en particular para manejar los pedidos y los productos.

El proyecto utiliza MongoDB como base de datos y el ORM Mongoose para interactuar con la base de datos. También utiliza la librería JSON Web Token (JWT) para la autenticación de usuarios.

`server.js`: este es el archivo principal de la aplicación. Configura el servidor Express y define las rutas de la API. También se encarga de conectarse a la base de datos MongoDB y de escuchar las solicitudes entrantes en el puerto especificado.

`routes/`: esta carpeta contiene los archivos que definen las rutas de la API. Cada archivo define las rutas y los controladores asociados para una entidad específica de la aplicación, como pedidos o productos.

`models/`: esta carpeta contiene los archivos que definen los modelos de datos para cada entidad de la aplicación. Cada modelo es una representación de un documento en la base de datos MongoDB y utiliza el ORM Mongoose para interactuar con la base de datos.

2 Doc. Técnica

Backend

controllers/: esta carpeta contiene los archivos que definen los controladores de la API. Cada controlador es una función que maneja las solicitudes entrantes a la API y llama a los métodos correspondientes en los modelos.

middlewares/: esta carpeta contiene los archivos que definen los middlewares de la aplicación. Un middleware es una función que se ejecuta antes o después de una solicitud entrante a la API. Los middlewares pueden usarse para agregar funcionalidad común a varias rutas, como la autenticación de usuarios.

config/: esta carpeta contiene los archivos de configuración de la aplicación, como la configuración de la base de datos y la configuración de JWT.

En general, el proyecto sigue un patrón de arquitectura MVC (Modelo-Vista-Controlador) para separar la lógica de la aplicación en capas separadas. Cada capa tiene una responsabilidad específica y se comunica con las demás a través de interfaces bien definidas.

A continuación, se detalla la documentación de cada una de las líneas de código, del index.js:

```
const express = require('express'); const morgan = require('morgan'); const cors = require('cors'); const path = require('path'); const app = express (); const bodyParser = require('body-parser'); const { mongoose } = require('./database'); const userRoutes = require("./routes/user"); const casesRoutes = require("./routes/cases"); const personRoutes = require("./routes/person"); const employeeRoutes = require("./routes/employee");
```

2 Doc. Técnica

Backend

En las primeras líneas, se importan las dependencias necesarias para el proyecto, incluyendo Express, Morgan, Cors, Path, Body-parser, y mongoose, una biblioteca de objetos modelo de MongoDB para Node.js. Además, se importan las rutas de los diferentes componentes del proyecto, cada uno en su archivo correspondiente en la carpeta "routes".

```
app.use(cors()); // Permite todas las conexiones
```

Se utiliza la función use() de Express para permitir todas las conexiones CORS.

```
app.set('port', process.env.PORT || 3000); Se utiliza la función set() de Express para configurar el puerto del servidor web.
```

```
app.use(morgan('dev'));
```

```
app.use(express.json());
```

```
app.use(bodyParser.json());
```

Se utilizan varias funciones use() de Express para configurar los middleware que serán ejecutados en cada solicitud recibida por el servidor.

El middleware de Morgan se utiliza para imprimir mensajes de registro en la consola, express.json() se utiliza para analizar los cuerpos de las solicitudes en formato JSON, y bodyParser.json() se utiliza para analizar los cuerpos de las solicitudes en formato JSON.

```
app.use('/api', userRoutes);
```

```
app.use('/api', casesRoutes);
```

```
app.use('/api', personRoutes);
```

```
app.use('/api', employeeRoutes);
```

Se utiliza la función use() de Express para definir las rutas que se utilizarán para acceder a cada componente del proyecto. Cada ruta comienza con "/api" seguida del nombre de la ruta correspondiente.

```
app.use('/tasks', require('./routes/user'));
```

```
app.use('/tasks', require('./routes/cases'));
```

```
app.use('/tasks', require('./routes/person'));
```

```
app.use('/tasks', require('./routes/employee'));
```

Se utilizan nuevamente las funciones use() de Express para definir las rutas que se utilizarán para acceder a cada componente del proyecto. Cada ruta comienza con "/tasks" seguida del nombre de la ruta correspondiente.

2 Doc. Técnica

Backend

```
app.get("/", (req,res) => { res.send("Welcome to sadimi"); });
```

Se utiliza la función `get()` de Express para definir la ruta raíz del proyecto. Cuando un usuario visita la página principal del proyecto, se muestra un mensaje de bienvenida.

```
app.use(express.static(path.join(__dirname, 'public')))
```

Se utiliza la función `use()` de Express para definir la ruta donde se almacenarán los archivos estáticos del proyecto. En este caso, la carpeta "public" en la ruta principal del proyecto.

```
app.listen(app.get('port'), () => { console.log(`Server on port  
${app.get('port')}`); });
```

Se utiliza la función `listen()` de Express para iniciar el servidor web. La aplicación se ejecutará en el puerto especificado. Cuando el servidor esté listo, se mostrará un mensaje diciendo "Server on port 10000" y el siguiente mensaje cuando este conectado a la base de datos "Connected to MongoDB Atlas".

Frontend and backend technical and user documentation: Sadimi

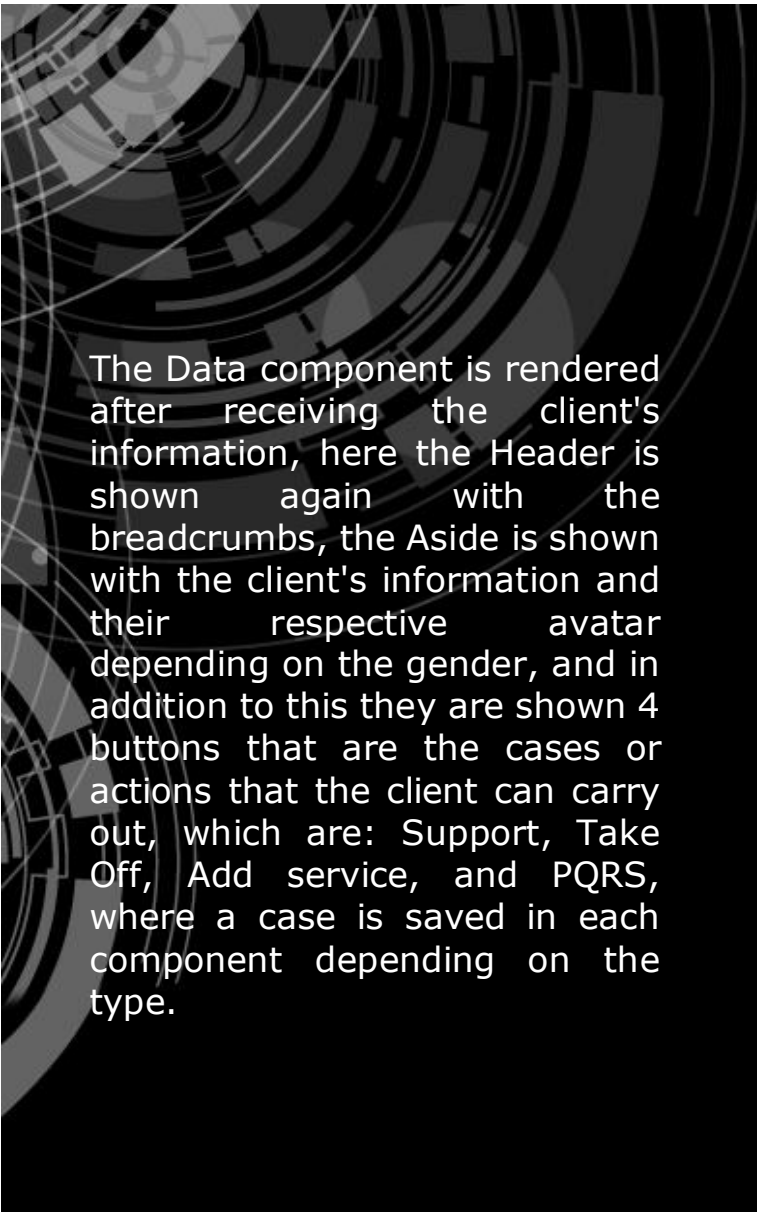
Sadimi is a technology CRM which provides specific services to its Microsoft membership associate customers. Sadimi offers the service to buy licenses, Xbox and computers. It is possible to request a technical service at home when a client requires it and likewise with each case, Sadimi will have a record of each activity that a client can carry out.

1 Technical Doc.

Frontend

The application is loaded in the index.js which calls the app.js which is where the components are loaded and the Router of the entire application is defined, referring to the path / or root path as the Login component which is where it is validated and the employee is called in the backend API, if I forgot my password, the Form component is reused, which is where the data is requested to change the password.

After successfully registering, the Home component is rendered, it contains the Header, which is another component that will be reused throughout the application, the employee's avatar depending on whether it is "Male", "Female" or "Non-binary". Below this, the Search component is rendered, which is the input where the ID of the user who is in communication with the CRM who has contact with the advisor is going to be requested. Below this component is the Search button that triggers the Onclick event and calls the API to verify that the document exists in the database and later, if it exists, load the Data component and save the client object in the SessionStorage to use it and not have to call the api for each request.



The Data component is rendered after receiving the client's information, here the Header is shown again with the breadcrumbs, the Aside is shown with the client's information and their respective avatar depending on the gender, and in addition to this they are shown 4 buttons that are the cases or actions that the client can carry out, which are: Support, Take Off, Add service, and PQRS, where a case is saved in each component depending on the type.

1 Technical Doc.

Frontend

The support component validates the address and telephone number to send a technician to the client's address in the Session Storage, which is the same as the one searched for in the search component. If the client presses accept, the case is saved with typo support and with a description that a technician was sent to the customer's address.

The Add service component looks for the services that the client has active to offer only those that he can buy that are not owned by him, for example if he has Xbox, only License and Computer are offered.

The Take Off component is to cancel the service that the client has and what it does is a patch to the API and then the service is removed in the API

The PQRS component adds a textarea for the client to vent and write their requests, complaints, claims or suggestions, then the cases are saved in the cases collection.

Finally, the AllCases component, which is to show all the cases that this client has had where an api call is made with the get method to bring them and later when the send button is pressed, the pdf is generated and sent by mail to the customer.

Additionally, an Error page was made to show that the route is not correct and the application was deployed on the free server render:

2 Technical Doc.

Backend

The project you mention is a code repository on GitHub that contains a backend application written in JavaScript using the Express.js framework. The purpose of the application is to provide an API for an e-commerce platform, in particular to handle orders and products.

The project uses MongoDB as the database and the Mongoose ORM to interact with the database. It also uses the JSON Web Token (JWT) library for user authentication.

server.js – This is the main file of the application. Configure the Express server and define the API routes. It also takes care of connecting to the MongoDB database and listening for incoming requests on the specified port.

routes/ – This folder contains the files that define the API routes. Each file defines the paths and associated controllers for a specific entity in the application, such as orders or products.

models/ – This folder contains the files that define the data models for each entity in the application. Each model is a representation of a document in the MongoDB database and uses the Mongoose ORM to interact with the database.

2 Technical Doc.

Backend

controllers/ – This folder contains the files that define the controllers for the API. Each controller is a function that handles incoming API requests and calls the corresponding methods on models.

middlewares/: This folder contains the files that define the application's middlewares. A middleware is a function that is executed before or after an incoming API request. Middlewares can be used to add common functionality to multiple routes, such as user authentication.

config/ – This folder contains the application's configuration files, such as database configuration and JWT configuration.

In general, the project follows an MVC (Model-View-Controller) architecture pattern to separate the application logic into separate layers. Each layer has a specific responsibility and communicates with the others through well-defined interfaces.

Below is the documentation for each of the lines of code, from index.js:

```
const express = require('express'); const morgan = require('morgan'); const cors = require('cors'); const path = require('path'); const app = express(); const bodyParser = require('body-parser'); const { mongoose } = require('./database'); const userRoutes = require("./routes/user"); const casesRoutes = require("./routes/cases"); const personRoutes = require("./routes/person"); const employeeRoutes = require("./routes/employee");
```

2 Technical Doc.

Backend

In the first few lines, the necessary dependencies for the project are imported, including Express, Morgan, Cors, Path, Body-parser, and mongoose, a MongoDB model object library for Node.js. In addition, the routes of the different components of the project are imported, each one in its corresponding file in the "routes" folder.

```
app.use(cors()); // Allow all connections
```

The Express function use() is used to allow all CORS connections.

```
app.set('port', process.env.PORT || 3000);
```

The Express function set() is used to set the web server port.

```
app.use(morgan('dev'));
```

```
app.use(express.json());
```

```
app.use(bodyParser.json());
```

Various Express use() functions are used to configure the middleware that will be executed on each request received by the server.

The Morgan middleware is used to print log messages to the console, express.json() is used to parse request bodies in JSON format, and bodyParser.json() is used to parse request bodies in JSON format. JSON.

```
app.use('/api', userRoutes);
```

```
app.use('/api', casesRoutes);
```

```
app.use('/api', personRoutes);
```

```
app.use('/api', employeeRoutes);
```

Express's use() function is used to define the paths that will be used to access each component in the project. Each route starts with "/api" followed by the corresponding route name.

```
app.use('/tasks', require('./routes/user'));
```

```
app.use('/tasks', require('./routes/cases'));
```

```
app.use('/tasks', require('./routes/person'));
```

```
app.use('/tasks', require('./routes/employee'));
```

The Express use() functions are again used to define the paths that will be used to access each component of the project. Each path begins with "/tasks" followed by the name of the corresponding path.

2 Technical Doc.

Backend

```
app.get("/", (req,res) => { res.send("Welcome to sadimi"); });
```

The Express function `get()` is used to define the root path of the project. When a user visits the main page of the project, a welcome message is displayed.

```
app.use(express.static(path.join(__dirname, 'public')))
```

The Express function `use()` is used to define the path where the project's static files will be stored. In this case, the "public" folder in the main path of the project.

```
app.listen(app.get('port'), () => { console.log(`Server on port  
${app.get('port')}`); });
```

The Express function `listen()` is used to start the web server. The application will run on the specified port. When the server is ready, it will show a message saying "Server on port 10000" and the following message when it is connected to the database "Connected to MongoDB Atlas".