

Memoria del proyecto

BusJerez



Busjerez

Indice:

1. Estudio del problema y análisis del sistema
 - 1.1. Introducción breve descripción del proyecto.
 - 1.2. Implementación.
 - 1.3. Objetivos qué servicios ofrecerá el sistema una vez implementado.
2. Modelado de la solución
 - 2.1. Recursos humanos
 - 2.2. Recursos hardware
 - 2.3. Recursos software
3. Planificación.
 - 3.1. Ejecución del proyecto
4. Conclusiones finales
 - 4.1. Grado de cumplimiento de los objetivos fijados.
 - 4.2. Propuesta de modificaciones o ampliaciones futuras .
5. Documentación del sistema desarrollado
 - 5.1. Manual de Instalación.
 - 5.2. Manual de uso.

1. Estudio del problema y análisis del sistema

1.1. Introducción breve

Este proyecto “BusJerez” está enfocado a dar una solución práctica a o y sencilla a la falta de exactitud de los autobuses de Jerez. Se busca dar información al usuario tanto como de las líneas y horarios de los autobuses además de una aplicación donde poder consultar la ruta más rápida por medio de estos.

Esta idea viene a raíz de la experiencia de estar por la ciudad, querer montarte en un autobús para ir a “x” lugar y no saber qué línea coger ni qué parada, automáticamente uno puede pensar en Google Maps ya que esta trae la función de desplazamiento por medio de transporte público pero no es exacto ya que no tiene los autobuses de dentro de la ciudad (línea 1, línea 2, línea 3, etc).

1.2. Implementación

Informar al usuario de las líneas y sus recorridos en cualquier momento. Posibilidad de ver las rutas, paradas y horarios de las líneas. Velocidad, fácil de utilizar y dinámico.

1.3. Objetivos que servicios ofrecerá el sistema una vez implementado

Servicio de información.

2. Modelado de la solución

2.1. Recursos hardware

El equipo utilizado es un portátil **MSI Modern 14 B10RBSW-420XES**

- Especificaciones:

- Procesador Intel Core i7-10510U
- Memoria DDR IV 16GB (2666MHz)
- Almacenamiento 512GB NVMe PCIe SSD
- Unidad óptica No
- Display 14" FHD (1920*1080), 60Hz 45%NTSC IPS-Level
- Controlador gráfico nVidia MX 350 2GB GDDR5
- Conectividad
 - WiFi 802.11 ac
 - Bluetooth v5
- Cámara de portátil si

- Microfono Si
- Bateria 3 cell , 52Whr
- Conexiones
 - 1 x Type-C USB3.2 Gen1
 - 2 x Type-A USB 2.0 Gen1
 - 1 x HDMI
 - Lector de tarjetas Micro SD
- Teclado retro iluminado Teclado retro iluminado (monocolor, blanco)
- Sistema operativo SIN SISTEMA OPERATIVO
- Dimensiones 319 x 220.2 x 16.9 mm
- Peso 1.30 kg
- Color Gris Carbon

2.2. Recursos software

Sistema operativo Windows 10

<u>Lo que se usara</u>	<u>Versión</u>
Visual Studio Code	1.65.2
Android Studio	Bumblebee 2021.1.1 Patch 1
Node	v16.13.1
Npm	8.1.2
Vue	3.0.1
Ionic	v6

Sobre la base de datos se usara mongoDB en la nube pro medio de su propia web.

<https://cloud.mongodb.com/>

He preferido esta base de datos por el volumen pequeño volumen de datos que tendré que manejar ya que en un principio solo serán las líneas de autobuses, horarios y paradas. Esto se traduce en la creación de 3 tablas, en un principio no se creara método de registro a la web ya que no veo útil que los usuarios se registren.

MongoDB ofrece una forma rápida y eficaz de hacer consultas y jugar con datos para proyectos como este, en el caso que necesite más fuerza optare por SQL Server.

Como framework usare ionic+vue ya que ionic aporta lo necesario referente a la geolocalización y vue.js es para crear una interfaz de usuario más elegante y sencilla.

Los lenguajes usados principalmente serán javascript, typescript y java (conversión de ionic a android).

Como lenguajes de marcas: html y css.

3. Planificación

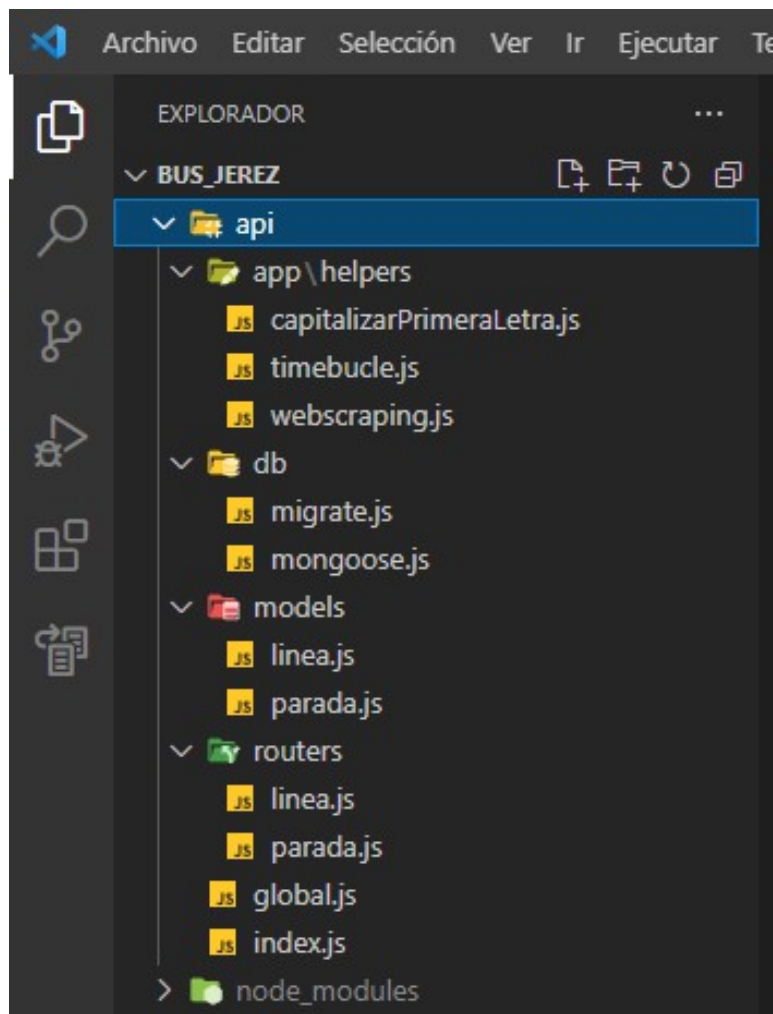
3.1. Ejecución del proyecto

Como ya se menciona anteriormente, BusJerez esta realizado en dos partes. Por un lado tenemos el backend realizado en node.js y fronted con ionic+vue.js.

Primero explicaremos un poco el backend.


Backend

Estructura de backend:



Node.js + el paquete de express se ha creado una api para atender llamadas y devolver de mi base de datos propia los modelos de las lineas y las paradas.

El uso de express ha sido fundamental para poder crear un servidor, en la siguiente imagen podemos ver el archivo principal index.js.

```
api >  index.js > ...
1  const config = require('./global')
2  const express = require('express')
3  const webscrap = require('./app/helpers/webscraping')
4  const timer = require('./app/helpers/timebucle')
5  require('./db/mongoose')
6
7  webscrap()
8
9  const lineaRouter = require('./routers/linea')
10 const paradaRouter = require('./routers/parada')
11
12 const app = express()
13
14 const host = config.HOST
15 const port = config.PORTAPI
16
17 app.use(express.json())
18 app.use(lineaRouter)
19 app.use(paradaRouter)
20
21 app.listen(port, () => {
22   console.log(`Servidor abierto en el puerto http://${host}:${port}/`)
23 })
24
25 setInterval(webscrap, timer(5));
```

Pasando vistazo rápido por el código se ve el uso de variables de entornos alojadas en un archivo .env y llamadas en el archivo global.js. También podemos ver el uso de webscraping personalizado para obtener los datos desde la web de autobuses para alojarlos en mi bbdd (mongodb), por otro lado tenemos rutas y por último una función que se ejecutara cada 5 días para comprobar que los datos no hayan cambiado.

Conexión a bbdd es rápida y se realiza en “./db/mongoose.js” tal como se enseñó en clase y haciendo uso de las variables de entorno que más tarde mostrare. El archivo migrate.js es solamente para la creación de las bases de datos.

Modelo Linea:

```
> models > lineajs > ...
1 const { ObjectId } = require('mongodb')
2 const mongoose = require('mongoose')
3 const Parada = require('./parada')
4
5 const lineaSchema = new mongoose.Schema({
6   nombre: {
7     type: String,
8     required: true,
9     trim: true,
10    unique: true
11  },
12  paradasPrimerSentido: {
13    type: [mongoose.Schema.Types.ObjectId],
14    ref: 'Parada'
15  },
16  paradasSegundoSentido: {
17    type: [mongoose.Schema.Types.ObjectId],
18    ref: 'Parada'
19  },
20  nombrePrimerSentido: {
21    type: String,
22    trim: true
23  },
24  nombreSegundoSentido: {
25    type: String,
26    trim: true
27  }
28 })
29
30 lineaSchema.post('save', async function (next) {
31
32   const paradalistPS = this.paradasPrimerSentido
33   paradalistPS.forEach(async e => {
34     let parada = await Parada.findOne({_id: e._id})
35     if(!parada.lineas.includes(this._id)){
36       parada.lineas.push(this._id)
37     }
38     parada.save()
39   })
40
41   const paradalistSS = this.paradasSegundoSentido
42   paradalistSS.forEach(async e => {
43     let parada = await Parada.findOne({_id: e._id})
44     if(!parada.lineas.includes(this._id)){
45       parada.lineas.push(this._id)
46     }
47     parada.save()
48   })
49 })
```

Modelo Parada:

```
> models > paradajs > ...
1 const mongoose = require('mongoose')
2
3 const paradaSchema = new mongoose.Schema({
4   nombre: {
5     type: String,
6     unique: true,
7     required: true,
8     trim: true
9   },
10  coordenadas: {
11    type: [String],
12    required: true,
13    trim: true
14  },
15  lineas: {
16    type: [mongoose.Schema.Types.ObjectId],
17    ref: 'Linea'
18  }
19 })
20
21 const Parada = mongoose.model('Parada', paradaSchema)
22
23 module.exports = Parada
```

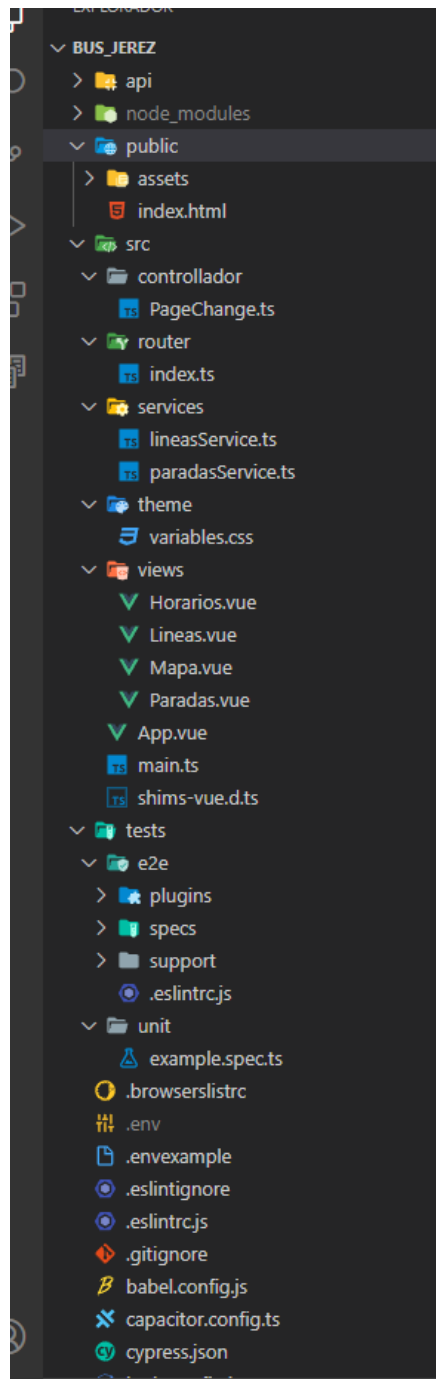
Luego el archivo “webscraping.js” es donde ocurre la magia de la api, aquí vamos a hacer peticiones a la web de autobuses y extraer los datos que necesitamos y luego guardarlos en la base de datos.

La ultima parte sería las rutas. Estas nos sirven para poder realizar la conexión con nuestro frontend. Tenemos creada dos rutas, linea y parada en las que simplemente nos devuelve los datos que necesitamos.

Frontend

Por otra parte tenemos el frontend, realizado en ionic + vue.js, elegí con vue y no angular por su versatilidad y aprender un framework nuevo, ionic es para poder portar luego la app a aplicación android de forma inmediata pero no es nativa aún así para este caso no se necesita de la potencia de una aplicación nativa.

Estructura frontend:



El frontend al ser muy extenso tocare los puntos esenciales sin entrar en código.

Principalmente tenemos nuestro archivo App.vue que es donde se inicia todo el proyecto y a raíz de este se derivan las demás vistas. En vue tenemos componentes y vistas aunque luego es prácticamente lo mismo solo que se utilizan de forma diferente pero en esencia todo son componentes. En la carpeta de views es donde tendremos nuestros archivos que vamos a enseñar al público. Vue hace que tengamos que mezclar en un mismo archivo tanto como scrpirt (en mi caso es typescrip), estyle y template que es nuestro html. Este html no contiene etiquetas de las que ya conocemos, al estar usando ionic utilizamos sus etiquetas para que quede bien luego al pasarlo a aplicación de movil.

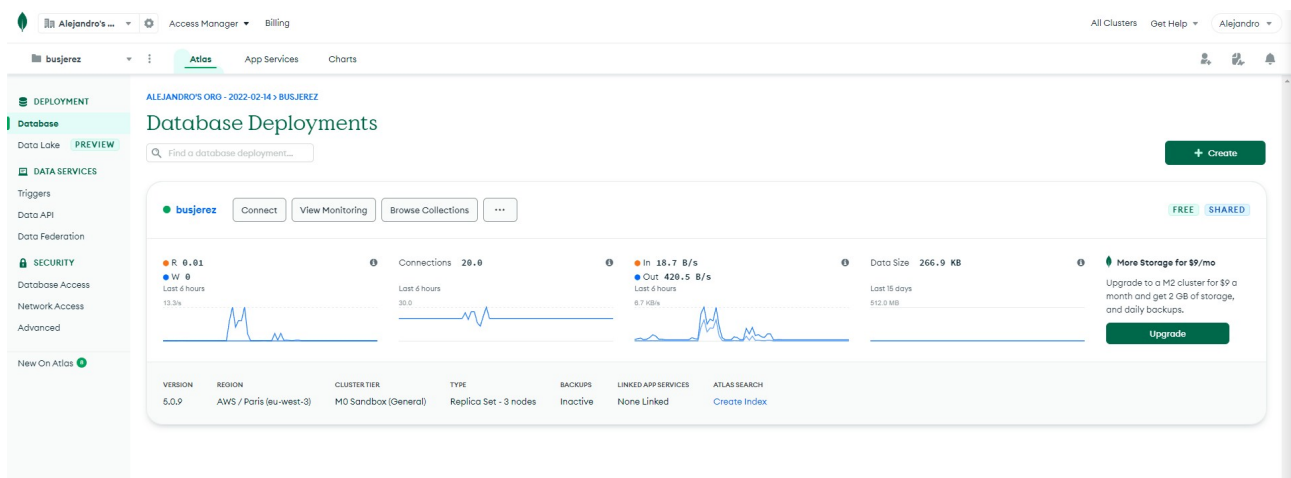
En la carpeta router es donde se define la rutas para la app.

La carpeta services es donde hacemos la conexión con node por medio de async/await.

Mongodb

Base de datos de esta app, no relacional, totalmente en la nube.

Realizamos conexión directa con [atlas](#).



Colecciones:

Lineas

Alejandro's ...

Access Manager

Billing

All ClustersGet HelpAlejandro

busjerezAtlasApp ServicesCharts

DEPLOYMENT

Database

DATA SERVICES

SECURITY

+ Create Database

Search Namespaces

test

lineas

paradas

test.lineas

STORAGE SIZE: 40KB TOTAL DOCUMENTS: 18 INDEXES TOTAL SIZE: 72KB

FindIndexesSchema Anti-PatternsAggregationSearch Indexes

INSERT DOCUMENT

FILTER { field: 'value' }

QUERY RESULTS: 1-18 OF 18

_id:ObjectId("62af38da2869f0eeffa448")

nombre:"Linea 1"

paradasPrimerSentido:Array

paradasSegundoSentido:Array

nombrePrimerSentido:"Sentido = San Telmo"

nombreSegundoSentido:"Sentido = Plaza Esteve"

__v:0

_id:ObjectId("62af38fa2869f0eeffa737")

nombre:"Linea 6"

paradasPrimerSentido:Array

paradasSegundoSentido:Array

nombrePrimerSentido:"Sentido = Almeria"

nombreSegundoSentido:"Sentido = Convento Agustias"

__v:0

_id:ObjectId("62af38ba2869f0eeffa79f")

System Status: All Good

©2022 MongoDB, Inc. Status Terms Privacy Atlas Blog Contact Sales

Paradas

Alejandro's ...

Access Manager

Billing

All ClustersGet HelpAlejandro

busjerezAtlasApp ServicesCharts

DEPLOYMENT

Database

DATA SERVICES

SECURITY

+ Create Database

Search Namespaces

test

lineas

paradas

test.paradas

STORAGE SIZE: 56KB TOTAL DOCUMENTS: 305 INDEXES TOTAL SIZE: 120KB

FindIndexesSchema Anti-PatternsAggregationSearch Indexes

INSERT DOCUMENT

FILTER { field: 'value' }

QUERY RESULTS: 1-20 OF MANY

_id:ObjectId("62af38da2869f0eeffa2ae")

nombre:"corredera"

coordenadas:Array

lineas:Array

horariosLaborales:Array

horariosAbados:Array

horariosAbadosyFestivos:Array

__v:0

_id:ObjectId("62af38ba2869f0eeffa2ae")

nombre:"plaza esteve"

coordenadas:Array

lineas:Array

horariosLaborales:Array

horariosAbados:Array

horariosAbadosyFestivos:Array

__v:0

PREVIOUS

1-20 of many results

NEXT

4. Conclusiones finales

4.1. Grado de cumplimiento de los objetivos fijados.

Lamentablemente no he podido implementar el mapa para guiarte debido a las complicaciones que me a puesto la api de google ya que actualmente esta deprecated y de la nueva api apenas hay información para lo que yo quería realizar más las tecnología que estaba usando. Por lo demás esta como se planteo de un principio.

4.2. Propuesta de modificaciones o ampliaciones futuras.

La propuesta principal es añadir lo que no he podido y un menú de administrador para poder modificar y crear las paradas y lineas.

5. Documentación del sistema desarrollado

5.1. Manual de Instalación.

La instalación de esta aplicación es muy sencilla, tan solo la tendremos que descargar de mi [github](#) ya sea con git clone o descargando el comprimido. Una vez lo tengamos descargado tan solo tendríamos que hacer:

```
npm install
```

y luego ejecutar las dos partes backend y frontend.

```
npm run test (backend)
```

```
npm run serve (frontend)
```

5.2. Manual de uso.

Tenemos 3 sitios donde poder visitar.

Paradas: Es donde podremos consultar que lineas pasan por esa parada.

Horarios: Poder consultar los horarios de las lineas que tu elijas.

Lineas: Consultar el recorrido que sigue la linea seleccionada y sus paradas.