

Sistemas y Tecnologías Web:

Proyecto Final

Desarrollo de una página web para alquilar
películas.

Equipo 16

Diego Wiederkehr Bruno
[\(alu0101601830@ull.edu.es\)](mailto:alu0101601830@ull.edu.es)

Daniel Arbelo Hernández
[\(alu0101117621@ull.edu.es\)](mailto:alu0101117621@ull.edu.es)



Índice:

| | |
|---|-----------|
| 1. Introducción. | 2 |
| 2. Objetivos. | 2 |
| 2.1. Objetivos principales. | 2 |
| 2.2. Objetivos secundarios. | 2 |
| 3. Desarrollo. | 2 |
| 3.1. Estructura, arquitectura y stack. | 2 |
| 3.2. Mockups. | 5 |
| 3.3. Metodologías Ágiles. | 9 |
| 3.4. Modelo de datos. | 9 |
| 3.4. Pivotal Tracker. | 11 |
| 3.5. Vue y comienzo del proyecto. | 18 |
| 3.6. Sign-In y Sign-Up. | 22 |
| 3.7. Session Management. | 24 |
| 3.8. Tests. | 27 |
| 3.9. Subir películas a la base de datos | 29 |
| 3.10. Github. | 31 |
| 3.11. Integración continua. | 33 |
| 4. Resultados finales. | 33 |
| 4.1. Página de entrada. | 34 |
| 4.2. Login y Registro. | 34 |
| 4.3. Catálogo de películas y página para subirlas | 35 |
| 4.4. Detalles de una película. | 36 |
| 4.5. Cesta del usuario. | 36 |
| 5. Conclusión. | 37 |
| 5.1. Conclusión Formal. | 37 |
| 5.2. Conclusión Personal. | 37 |
| 6. Despliegue de Aplicación. | 37 |
| 7. Referencias. | 38 |



1. Introducción.

En este proyecto desarrollado en la asignatura de Sistemas y Tecnologías Web hemos tenido que utilizar distintas tecnologías que utilizaría un full-stack web developer para crear una página web que en nuestro caso funciona como una plataforma para que los usuarios puedan ver un catálogo de películas, ver los detalles sobre cada película y añadirlas a su cesta para comprarlas a posteriori.

2. Objetivos.

A continuación se describen los objetivos de nuestra aplicación web

2.1. Objetivos principales.

Los objetivos principales son:

- Que los usuarios se registren e inicien sesión
- Visualizar listado de películas y selección por categorías
- Poder entrar en la información de cada película
- Añadir películas a la cesta
- Comprar las películas de la cesta

2.2. Objetivos secundarios.

Los objetivos secundarios son:

- Un objetivo secundario es poder hacer lo anterior pero con series.
- Otro objetivo secundario sería hacer una red social dentro del propio videoclub online donde los usuarios valoren contenido y que pudieran hablar entre ellos por foros

3. Desarrollo.

Para desarrollar esta página web hemos utilizado distintas herramientas no solo para la creación de la misma sino también para el seguimiento del trabajo.

3.1. Estructura, arquitectura y stack.

Básicamente, al ser una aplicación web full stack, tendremos un backend que se conectará con la base de datos que será implementada con mongodb y a la vez



a la parte del frontend. La estructura quedaría como la de las prácticas iniciales individuales.



Para desarrollar la aplicación web utilizaremos el conjunto de subsistemas MEVN: las siglas MEVN significan MongoDB, Express.js, VueJS y Node.js.

El stack MEVN es el stack de JavaScript de código abierto que ha surgido como una forma nueva y evolutiva de crear aplicaciones web potentes y dinámicas. Sus componentes de software se pueden utilizar para diseñar eficazmente el desarrollo frontend y backend y mejorar la funcionalidad de su sitio web o aplicación:

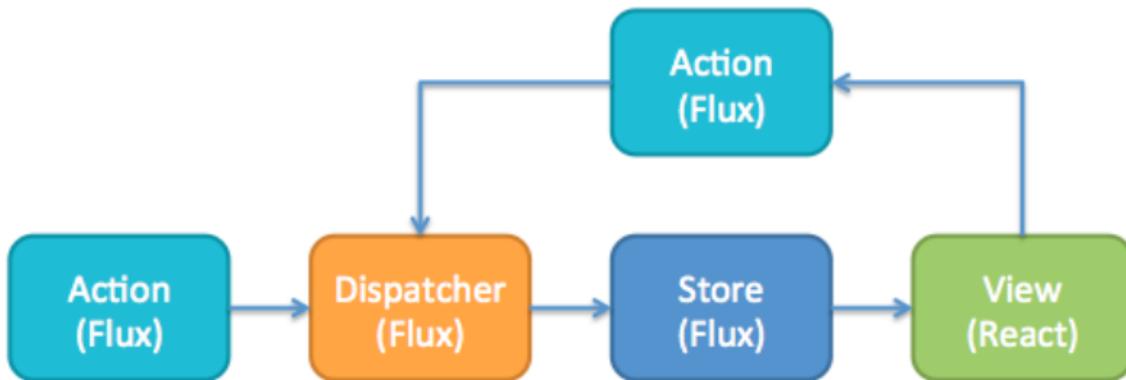
- **MongoDB:** una base de datos orientada a documentos, No-SQL utilizada para almacenar los datos de la aplicación en nuestro caso utilizaremos la base de datos de mongodb que está en la nube con colaboración de Atlas.
- **ExpressJS:** un marco de trabajo superpuesto a NodeJS, utilizado para construir el backend de un sitio utilizando funciones y estructuras de NodeJS. Dado que NodeJS se desarrolló principalmente para ejecutar JavaScript en una máquina en lugar de para crear sitios web, ExpressJS se creó para este último propósito.
- **VueJS:** se conoce como un framework del lado del cliente y se utiliza especialmente en el desarrollo web frontend. Tiene un enlace de datos bidireccional que permite el desarrollo frontend sin problemas junto con la capacidad MVC y aplicaciones interactivas del lado del servidor.
- **NodeJS:** el entorno de ejecución de JavaScript. Se utiliza para ejecutar JavaScript en una máquina en lugar de en un navegador.





La arquitectura que utilizaremos es la arquitectura FLUX. Para esto último con Vue utilizaremos la librería Vuex.

Flux propone una arquitectura en la que el flujo de datos es unidireccional. Los datos viajan desde la vista por medio de acciones y llegan a un Store desde el cual se actualizará la vista de nuevo.



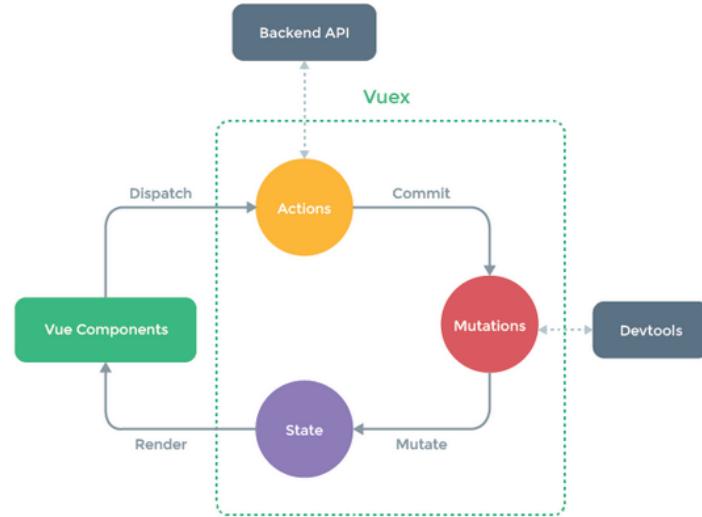
Teniendo un único camino, y un sitio donde se almacena el estado de la aplicación, es más sencillo depurar errores y saber qué está pasando en cada momento. Los distintos actores que tenemos en Flux son:

- **Vista:** la vista serían los componentes web, en nuestro caso, VueJS.
- **Store:** es lo más parecido al modelo de la aplicación. Guarda los datos/estado de la aplicación y en Flux puede haber varias.
- **Acciones:** una acción es simplemente un objeto JavaScript que indica una intención de realizar algo y que lleva datos asociados si es necesario.
- **Dispatcher:** es un mediador entre la Store o Stores y las acciones. Sirve para desacoplar la Store de la vista, ya que así no es necesario conocer qué Store maneja una acción concreta.

En resumen, el patrón Flux sigue el siguiente recorrido:

- La vista, mediante un evento envía una acción con la intención de realizar un cambio en el estado.
- La acción contiene el tipo y los datos (si los hubiere) y es enviada al dispatcher.
- El dispatcher propaga la acción al Store y se procesa en orden de llegada.
- El Store recibe la acción y dependiendo del tipo de recibido, actualiza el estado y notifica a las vistas de ese cambio.
- La vista recibe la notificación y se actualiza con los cambios.

Esta patrón lo hemos utilizado con Vuex que funciona de la siguiente manera:



3.2. Mockups.

Inicio: al entrar en la página web se puede observar un pequeño mensaje de bienvenida con dos botones distintos, uno para entrar en caso de que ya tengas una cuenta creada (Login) y el otro para crear la cuenta (Register).



Login: esta es la página de Login, en el caso de que el usuario tenga una cuenta ya creada solo deberá poner su nombre de usuario y su contraseña.



Registro: si el usuario no tiene una cuenta ya creada puede crearla aquí, solo debe introducir su email, nombre de usuario, contraseña y repetir la contraseña para asegurarse que está bien.





Página principal: aquí es donde se pueden ver todas las películas del catálogo disponibles. Se puede hacer scroll hacia los lados para ver todas las películas y pinchar en cualquiera de ellas para ver más detalles. Abajo del todo se puede ver el menú donde se puede ir al perfil, la cesta de la compra o la página principal.



Detalles de la película: al pinchar en cualquiera de las películas se pueden ver los detalles de la misma. A parte de la carátula, se puede observar el director, los actores, el año de la peli, una pequeña sinopsis, el precio y finalmente un botón para añadir la película al cesto de la compra.

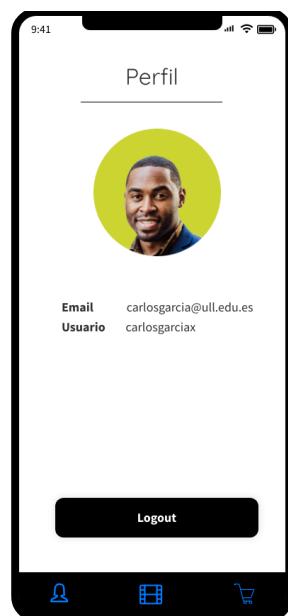




Cesta de la compra: aquí podemos ver todas las películas que el usuario ha añadido a la cesta de la compra, en cada película puede ver el precio de la misma y al final del todo se verá el precio total de la cesta. Además por película el usuario puede eliminarla dando al botón de la basura o ver más información de la misma dando al botón de info.



Perfil: en esta página podremos ver una foto de perfil que se puede cambiar como se quiera, el nombre del usuario, su email y finalmente un botón de logout para cerrar sesión.

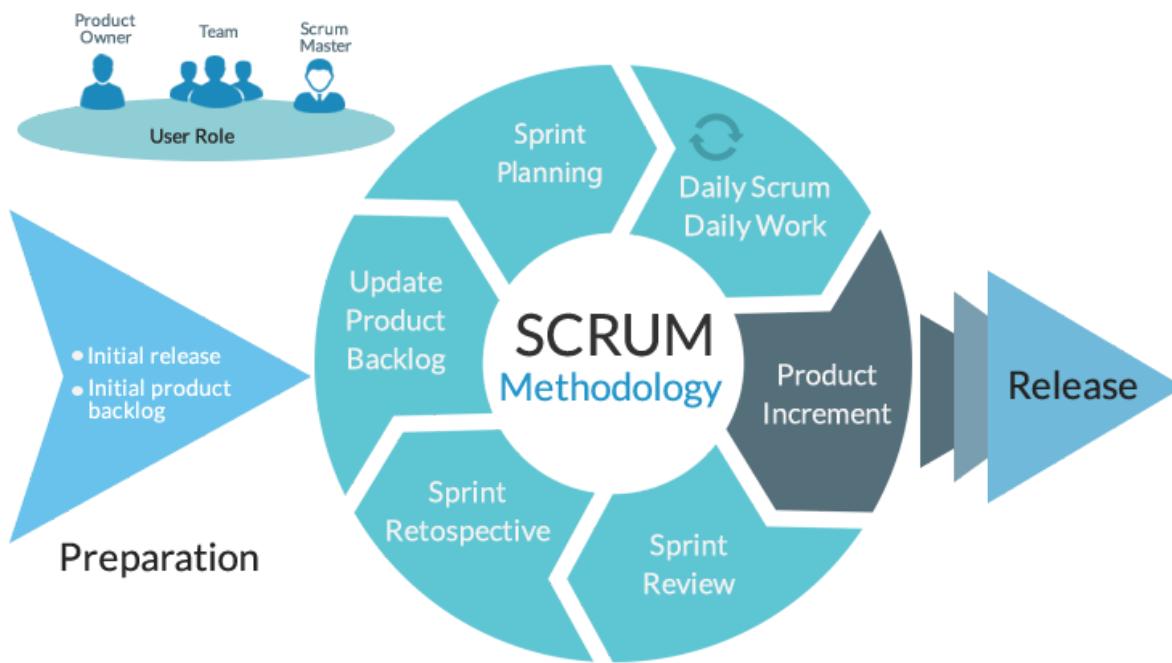




3.3. Metodologías Ágiles.

En nuestro caso, vamos a utilizar, como metodología ágil, SCRUM. Esta metodología se basa en una estructura de desarrollo incremental, esto es, cualquier ciclo de desarrollo de la página web se desgrana en “pequeños proyectos”. En la etapa de desarrollo encontramos lo que se conoce como interacciones del proceso o Sprint, es decir, entregas regulares y parciales del producto final que en nuestro caso iremos haciendo distintos objetivos por cada iteración que hicimos.

Esta metodología permite abordar proyectos complejos que exigen una flexibilidad y una rapidez esencial a la hora de ejecutar los resultados. La estrategia irá orientada a gestionar y normalizar los errores que se puedan producir en desarrollos demasiado largos, a través de reuniones frecuentes para asegurar el cumplimiento de los objetivos.



3.4. Modelo de datos.

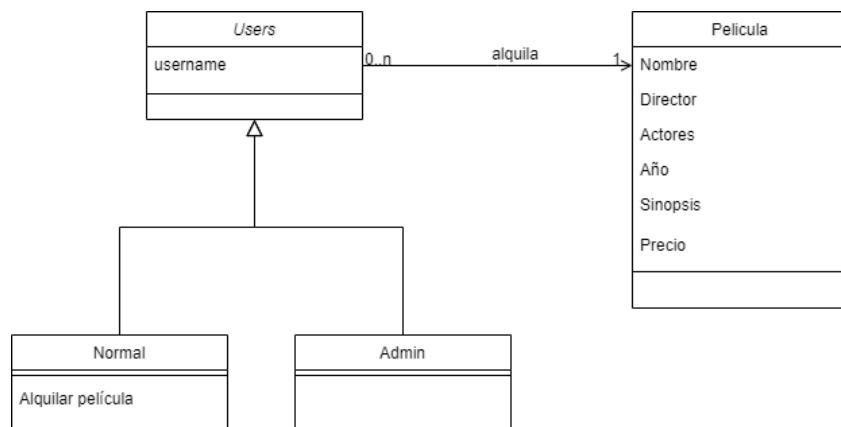
Creamos una base de datos en la nube con Mongodb Atlas y la organizamos basada en el siguiente modelo:

Para nuestra database, vamos a tener usuarios y películas. Los usuarios pueden ser normales o admins. Todos tendrán un nombre de usuario y una contraseña



con la que acceder. Un administrador tendrá la funcionalidad de poder manejar la base de datos así como eliminar o agregar usuarios o películas. Un usuario normal puede ver el listado de películas y añadir a la cesta de la compra. Cuando quiera alquilar las películas de la cesta de la compra, tendrá la opción de alquilarlas todas a la vez.

Una película tendrá nombre, director, actores, año, sinopsis y precio. El modelo relacional quedaría algo así:



La base de datos en Atlas nos quedaría así para los usuarios y las películas respectivamente:

The screenshot shows the MongoDB Atlas Data Services interface. The left sidebar includes sections for Project 0, Deployment, Database (selected), Data Lake, Services, Security, and Advanced. The main area has tabs for Data Services (selected), App Services, and Charts. A search bar at the top has the text "Find" and a dropdown menu with "Indexes", "Schema Anti-Patterns", "Aggregation", and "Search Indexes". Below the search bar is a "FILTER" button with the expression "{ field: 'value' }". To the right are "OPTIONS", "Apply", and "Reset" buttons. The results section shows a table with one row, labeled "QUERY RESULTS: 1-2 OF 2". The row contains fields: _id (ObjectId('63b578628dbfbe3cf8b0f00')), gmail (alg@mail.com), usuario (daniel), contrasena (1234), and __v (0). At the bottom, it says "System Status: All Good" and lists links for 2023 MongoDB, Inc., Status, Terms, Privacy, Atlas Blog, and Contact Sales.



The screenshot shows the MongoDB Atlas Data Services interface. On the left, there's a sidebar with sections like Deployment, Database (with a preview button), Services, and Security. The main area shows a database named 'test' with a collection named 'films'. Inside 'films', there are two documents: 'users' and 'vueexpress'. A specific document is selected, showing its details:

```
_id: ObjectId('63c98e13d43fc69dd2fc39c2')
titulo: "Pulp Fiction"
image: Object
  data: BinData(0, 'cHVsbgZpY3RpB24ucG5n')
  contentType: "image/png"
descripcion: "Historias de dos matones, un boxeador y una pareja de atracadores de p..."
ano: 1994
duracion: 154
genero: "Drama"
director: "Quentin Tarantino"
precio: 10
__v: 0
```

At the bottom of the interface, it says 'System Status: All Good' and includes links for Status, Terms, Privacy, Atlas Blog, and Contact Sales.

3.4. Pivotal Tracker.

Lo primero que hemos tenido que hacer ha sido registrarnos en la plataforma de Pivotal Tracker por separado para poder utilizar esta herramienta.

Con las cuentas creadas, nos hemos unido a la invitación de nuestro profesor para poder tener el proyecto:

The screenshot shows the Pivotal Tracker interface for the project '2022-E16'. The left sidebar has sections for Stories, Analytics, Members, Integrations, and More. The main area shows the 'Current Iteration/Backlog' with 10 stories, a 'Current/Backlog' section with 0 points, and an 'Icebox' section. The backlog section contains a message: 'Stories you are currently working on, and stories you've prioritized to work on next live here.' The icebox section contains a message: 'Loose ideas and stories that haven't been prioritized go here.'

Después hemos estado analizando y diseñando las funcionalidades necesarias para poder identificar las primeras tareas a desarrollar y poder escribir las fichas correspondientes y asignarles una valoración de coste en Pivotal Tracker.



Las tareas se agruparán en épicas, que son una gran tarea grande que engloba tareas más pequeñas, por ejemplo, el desarrollo del front-end, que tiene las tareas de crear las diferentes páginas de la aplicación. La otra épica que utilizaremos es la de backend.

La primera tarea será crear la estructura inicial del proyecto.

The screenshot shows a project management interface with a story creation dialog open. The story title is "Crear estructura del proyecto". The description field contains: "Se crea la estructura de la aplicación, el arbol de directorios y los ficheros necesarios". The labels field has "front-end" selected. The epic dropdown shows "Back-end" and "Front-end" options.

La segunda será crear la pantalla de inicio, que estará dentro de la épica de Front-end

The screenshot shows a project management interface with a story creation dialog open. The story title is "Crear pantalla de inicio". The description field contains: "Esta pantalla contendrá el botón para iniciar sesión, el de registrarse y una pequeña descripción de la aplicación". The labels field has "front-end" selected. The epic dropdown shows "Back-end" and "Front-end" options.

Después de esto se creará la página de login



The screenshot shows a Jira project titled '2022-E16'. In the center, a story card for 'Crear página de login' is being edited. The story is set to 'front-end', has an ID of #183836747, and is currently 'Unstarted'. The 'DESCRIPTION' field contains the following text:

Página de login donde se introducirá el nombre de usuario y la contraseña, y si son correctas se logeará a la página principal

The 'LABELS' field contains 'front-end'. The 'CODE' field is empty. The 'TASKS (0/0)' section is also empty.

Ahora hay que hacer el backend del login para que saque la información de la base de datos y compare con la información introducida a ver si existe el usuario y es correcta la contraseña.

The screenshot shows a Jira project titled '2022-E16'. In the center, a story card for 'Crear el backend del login' is being edited. The story is set to 'back-end', has an ID of #183836747, and is currently 'Unstarted'. The 'DESCRIPTION' field contains the following text:

Saca la información de la base de datos y compara con la información introducida a ver si existe el usuario y es correcta la contraseña.

The 'LABELS' field contains 'back-end'. The 'CODE' field is empty. The 'TASKS (0/0)' section is also empty.

Crear página de registro



The screenshot shows a Jira project interface for '2022-E16'. On the left, there's a sidebar with navigation links like 'My work', 'Current/backlog', 'Icebox', 'Done', 'Blocked', 'Epic', 'Labels', and 'Project history'. The main area displays a story card for 'Crear página de registro' (ID #183836811). The card details include:

- STATE:** Start (Unstarted)
- STORY TYPE:** Feature
- POINTS:** 1 Point
- REQUESTER:** alu0101117621
- OWNERS:** alu0101601830
- DESCRIPTION:** Página que contiene una entrada para introducir el gmail, el nombre de usuario, la contraseña y repetir la contraseña.
- LABELS:** front-end
- CODE:** Paste link to pull request or branch...

To the right of the story card are two panels: 'Icebox' and 'Epics'. The 'Icebox' panel contains a message: 'Loose ideas and stories that haven't been prioritized go here.' The 'Epics' panel lists 'Back-end' and 'Front-end' under the 'Epics' category.

Crear backend del registro que comprueba si ya existe un usuario con ese gmail, si el usuario ya existe y si la contraseña introducida es correcta, si no es así dará un error, si es correcto se registrará y llevará a la pantalla de login para que pueda loguearse el usuario.

The screenshot shows a Jira project interface for '2022-E16'. The sidebar and story card details are identical to the previous screenshot, except for the 'POINTS' value which is now 2 Points. The 'DESCRIPTION' field also includes additional text: 'Comprueba si ya existe un usuario con ese gmail, si el usuario ya existe y si la contraseña introducida es correcta, si no es así dará un error, si es correcto se registrará y llevará a la pantalla de login para que pueda loguearse el usuario.'

Crear backend para añadir y eliminar películas de la base de datos y poder acceder a ellas.



Current iteration... 10 + Add Story

STORY TYPE Feature

POINTS 2 Points

REQUESTER alu0101117621

OWNERS alu0101117621, alu0101601690

FOLLOW THIS STORY 2 followers

BLOCKERS + Add blocker or impediment

DESCRIPTION Se crea el backend para poder guardar películas en la base de datos, y poder eliminarlas.

LABELS back-end

CODE Paste link to pull request or branch...

TASKS (0/0) + Add a task

ACTIVITY Sort by Oldest to newest

Crear página inicial que contiene el listado de todas las películas.

Current iteration... 10 + Add Story

STORY TYPE Feature

POINTS 1 Point

REQUESTER alu0101117621

OWNERS alu0101601690

FOLLOW THIS STORY 2 followers

STATE Unstarted

REVIEWS + add review

BLOCKERS + Add blocker or impediment

DESCRIPTION Muestra todas las películas disponibles

LABELS front-end

CODE Paste link to pull request or branch...

Página de detalles de las películas



Current iteration... 10 + Add Story

0 of 10 points 1 · 14 - 20 Nov · 100%

Hacer página de detalles la película

STORY TYPE Feature

POINTS 1 Point

REQUESTER alu0101117621

OWNERS alu0101117621

FOLLOW THIS STORY 2 followers

BLOCKERS + Add blocker or impediment

DESCRIPTION Write Preview Formatting help

Cuando se accede a una película, redirige a esta página que contiene más información sobre esta

LABELS front-end

CODE Paste link to pull request or branch...

Tasks (0/0)

Icebox + Add Story

Epics Back-end Front-end

Backend para que se puedan guardar las películas que el usuario tiene añadidas a su cesta.

Current iteration... 10 + Add Story

0 of 10 points 1 · 14 - 20 Nov · 100%

Backend pra poder añadir películas a la cesta de la compra

STORY TYPE Feature

POINTS 2 Points

REQUESTER alu0101117621

OWNERS alu0101117621

FOLLOW THIS STORY 0 followers

BLOCKERS + Add blocker or impediment

DESCRIPTION Write Preview Formatting help

Hacer que al usuario abra algo a la cesta de la compra, lo guarde en la base de datos, para que cuando vuelva a ingresar se aparezcan las pelis que tiene añadidas

LABELS back-end

CODE Paste link to pull request or branch...

Tasks (0/0)

Icebox + Add Story

Epics Back-end Front-end

Página de la cesta de la compra.



The screenshot shows the Pivotal Tracker interface for a project titled "2022-E16". The left sidebar lists "My work" (5 items), "Current/backlog", "Done" (0 items), "Blocked", "Epic", "Labels", and "Project history". The main area has tabs for "STORIES", "ANALYTICS", "MEMBERS", "INTEGRATIONS", and "MORE". A "Current Iteration..." card shows 10 points, 14-20 Nov, and 100% completion. An "Add Story" dialog is open, titled "Página de la cesta de la compra". It includes fields for ID, Points (1 Point), Requester (alu0101117621), Owners (alu0101601830), and Description ("Página que contiene todo lo que el usuario ha añadido a su lista y pueda comprarlo"). Other sections include "BLOCKERS", "LABELS" (with "front-end" selected), "CODE", "TASKS (0/0)", and "ACTIVITY". To the right are "Icebox" and "Epics" panels. The URL <https://www.pivotaltracker.com/n/projects/2612899#> is visible at the bottom.

Página de perfil.

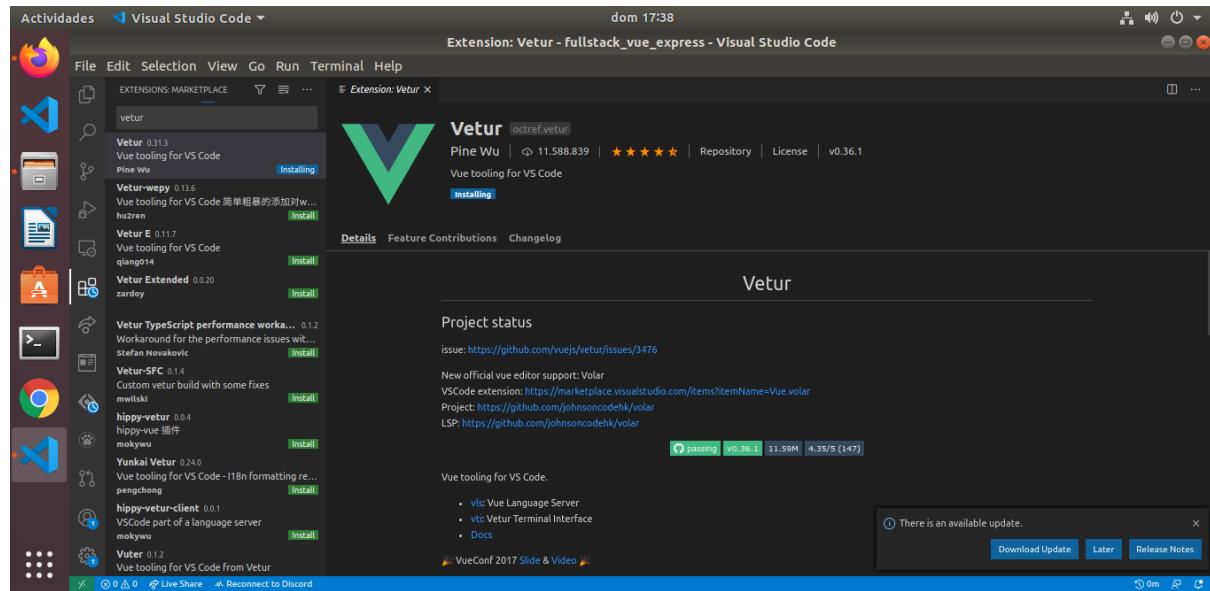
This screenshot is identical to the one above, showing the Pivotal Tracker interface for the same project. The "Description" field in the "Add Story" dialog now contains the text "Página donde el usuario puede ver su información". The URL <https://www.pivotaltracker.com/n/projects/2612899#> is also present at the bottom.

A medida que íbamos avanzando en el proyecto añadimos alguna historia más y nos iba quedando algo así:

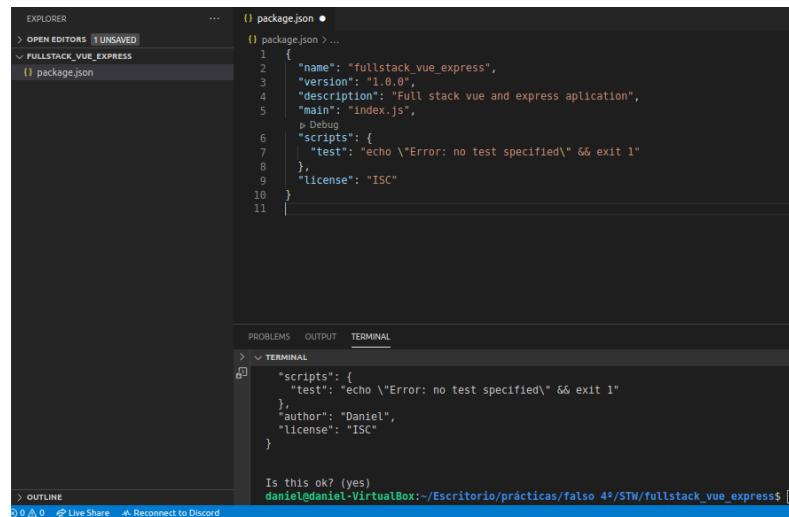


3.5. Vue y comienzo del proyecto.

En primer lugar, instalamos las extensiones siguientes para que nos facilite el desarrollo en vue:



Y ahora procedemos a crear los directorios y ficheros donde desarrollaremos el backend utilizando Express. Dado que es una aplicación node.js empezamos poniendo el comando “npm init”.



Y ahora instalamos algunas dependencias que vamos a necesitar:

```
{
  "name": "fullstack_vue_express",
  "version": "1.0.0",
  "description": "Full stack vue and express application",
  "main": "index.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "license": "ISC",
  "dependencies": {
    "body-parser": "^1.20.1",
    "cors": "^2.8.5",
    "express": "^4.18.2",
    "mongodb": "^4.12.1"
  },
  "devDependencies": {
    "nodemon": "^2.0.20"
  }
}
```



Ahora cambiamos los scripts y creamos un directorio llamado servidor que contendrá un index.js que tendrá la información del servidor backend.

```
{  
  "name": "fullstack_vue_express",  
  "version": "1.0.0",  
  "description": "Full stack vue and express application",  
  "main": "index.js",  
  "scripts": [  
    "start": "node server/index.js",  
    "dev": "nodemon server/index.js"  
  ],  
  "license": "ISC",  
  "dependencies": {  
    "body-parser": "^1.20.1",  
    "cors": "^2.8.5",  
    "express": "^4.18.2",  
    "mongodb": "^4.12.1"  
  },  
  "devDependencies": {  
    "nodemon": "^2.0.20"  
  }  
}
```

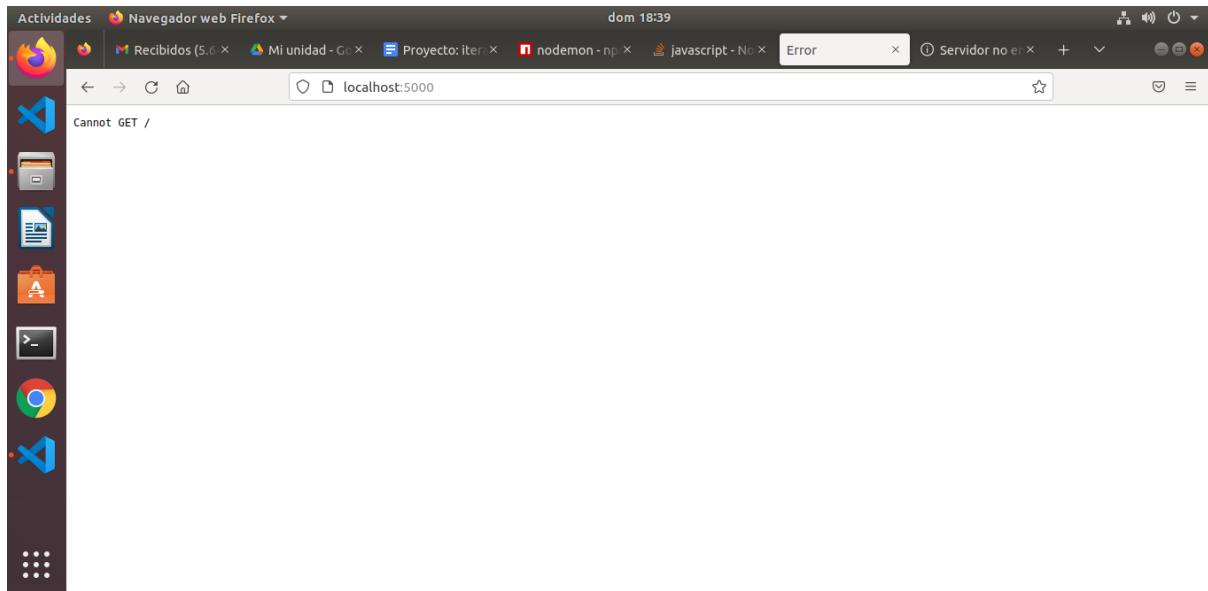
Y el index.js quedaría así, también hemos ejecutado el npm run dev para ver si funciona:

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure: `FULLSTACK_VUE_EXPRESS` (with `node_modules` and `server` folders), `index.js` (selected), and `package-lock.json`.
- Editor:** Displays the `index.js` file content:const express = require('express');
const bodyParser = require('body-parser');
const cors = require('cors');
const app = express();
// middleware
app.use(bodyParser.json());
app.use(cors());
const port = process.env.PORT || 5000;
app.listen(port, () => console.log(`Server sarted on port \${port}`));
- Terminal:** Shows the command `npm run dev` being executed and its output:

```
> fullstack_vue_express@1.0.0 dev /home/daniel/Escritorio/prácticas/falso 4º/STW/fullstack_vue_express  
> nodemon server/index.js  
[nodemon] 2.0.20  
[nodemon] to restart at any time, enter `rs`  
[nodemon] watching path(s): *.  
[nodemon] watching extensions: js,mjs,json  
[nodemon] starting `node server/index.js`  
Server sarted on port 5000
```

Y si funciona no hay nada en la ruta:



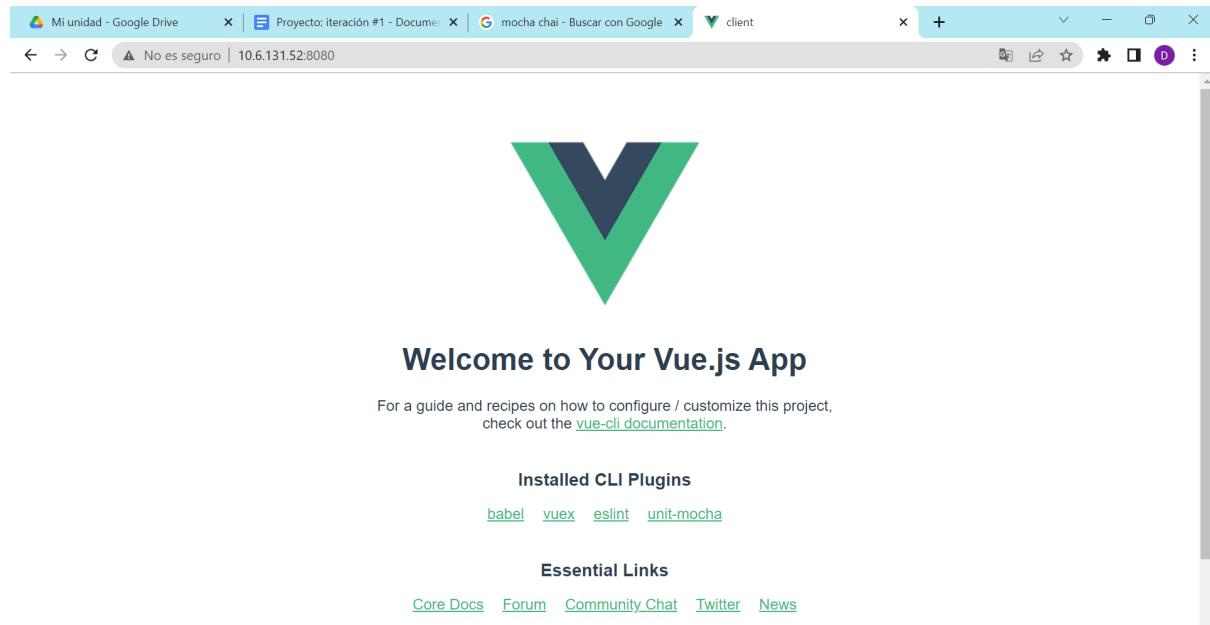
Ahora quedaría crear la estructura del front end que lo haremos con el comando “`vue create client`”, para esto debemos instalar vue con anterioridad. Y lo creamos eligiendo la opción de vuex, ya que lo vamos a utilizar y unit testing.

```
Vue CLI v5.0.8
? Please pick a preset: Manually select features
? Check the features needed for your project: (Press <space> to select, <a> to toggle all
  <i> to invert selection, and <enter> to proceed)
  • Babel
  o TypeScript
  o Progressive Web App (PWA) Support
  o Router
  • Vuex
  o CSS Pre-processors
  • Linter / Formatter
  >• Unit Testing
  o E2E Testing
```

Que más adelante utilizaremos con mocha y chai.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Vue CLI v5.0.8
? Please pick a preset: Manually select features
? Check the features needed for your project: Babel, Vuex, Linter, Unit
? Choose a version of Vue.js that you want to start the project with 3.x
? Pick a linter / formatter config: Basic
? Pick additional lint features: Lint on save
? Pick a unit testing solution:
  Jest
  > Mocha + Chai
```

A continuación entramos en cliente y utilizamos el comando `npm run serve` y entonces la app está desplegada (podemos ver el frontend de vue):



3.6. Sign-In y Sign-Up.

-Backend:

En el backend, primero tendremos que instalar mongoose, para conectarnos a la base de datos, mediante las siguientes líneas:

```
mongoose.set("strictQuery", false);
mongoose.connect('mongodb+srv://abc123:Z7ZhKVI@vueexpress.bvdaajx.mongodb.net/?retryWrites=true&w=majority')
```

Luego creamos una carpeta models donde pondremos los modelos de datos que subiremos, en este caso el de user que contendrá los datos del usuario que se está registrando para subirlo a la base de datos. Esto lo importamos en el index que es donde está el código del backend.

```

> tests
  E .browserslistrc
  ⓧ .eslintrc.js
  ⓧ .gitignore
  ⓧ babel.config.js
  ⓘ jsonconfig.json
  ⓘ package-lock.json
  ⓘ package.json
  ⓘ README.md
  ⓘ vue.config.js
  > node_modules
  < server
    < models
      JS Film.js
      JS User.js
      > routes
      JS index.js
      > uploads
      ⓧ .gitignore

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
[nodemon] starting `node server/index.js`
Server started on port 5000
[nodemon] restarting due to changes...
[nodemon] starting `node server/index.js`
```

+ ^ >

npm npm



Una vez hecho esto creamos una ruta a la que llegará la petición post para registrarse:

```
app.post('/signup', (req, res, next) => {
  const newUser = new User({
    gmail: req.body.gmail,
    usuario: req.body.usuario,
    contrasena: req.body.contrasena
  })
  newUser.save(err => {
    if (err) {
      return res.status(400).json({
        title: 'error',
        error: 'email in use'
      })
    }
    return res.status(200).json({
      title: 'signup success'
    })
  })
})
```



Creamos una instancia del model y lo subimos.

-Front end:

En el front tendremos un formulario para que el usuario introduzca su información, y al pulsar el botón de registro se llamará a la función createUser.

```
<template style="background-color: black">
<div>
  <p><H1 id="title" style ="color: black;"><b>
  | Registro
  </b></H1></p>
  <hr>
  <label>Email</label><br>
  <input id="email" name="email" type="text" placeholder="Email del usuario" required v-model="email">
  <label>Usuario</label><br>
  <input id="usuario" name="usuario" type="text" placeholder="Nombre del usuario" required v-model="usuario">
  <label>Contraseña</label><br>
  <input id="contraseña" name="contraseña" type="password" placeholder="Contraseña del usuario" v-model="contraseña">
  <br><br>
  <button id="btn-registro" v-on:click="createUser" style="width:150px; height:45px; background-color: black; color: white; border: none; font-weight: bold; font-size: 1em; padding: 5px; margin-top: 10px">
    {{ error }}
  </button>
</div>
</template>
```



La función createUser hace un post al backend al código que explicamos anteriormente, con la información para que este lo suba a la base de datos. Si es satisfactorio, nos redirige al login para que iniciemos sesión.



```
},
methods: {
  async createUser() {
    let newUser = {
      usuario: this.usuario,
      gmail: this.gmail,
      contrasena: this.contrasena
    }
    axios.post('http://localhost:5000/signup', newUser)
      .then( res => {
        this.error = res;
        this.$router.push('/login');
        this.error = '';
      }, err => {
        console.log(err.response)
        this.error = err.response.data.error
      })
  }
}
```

3.7. Session Management.

-Backend:

Como podemos ver si se hace un post la login del backend con la información del usuario, primero se busca al usuario por el gmail, ya que es único para cada usuario registrado, si todo va bien, no hay error en el servidor y la contraseña es correcta, se crea el jwt token y se envía al front para que se pueda autenticar al hacer peticiones.

```
app.post('/login', (req, res, next) => {

  User.findOne({ gmail: req.body.gmail }, (err, user) => {
    if (err) return res.status(500).json({
      title: 'server error',
      error: err
    })
    if (!user) {
      return res.status(401).json({
        title: 'user not found',
        error: 'invalid credentials'
      })
    }
    //incorrect password
    if (req.body.contrasena != user.contrasena) {
      return res.status(401).json({
        title: 'login failed',
        error: 'invalid credentials'
      })
    }
  })
})
```



```
        }
        //incorrect password
        if (req.body.contrasena != user.contrasena) {
            return res.status(401).json({
                title: 'login failed',
                error: 'invalid credentials'
            })
        }
        //IF ALL IS GOOD create a token and send to frontend
        let token = jwt.sign({userId: user._id}, 'secretkey');
        return res.status(200).json({
            title: 'login succes',
            token: token
        })
    })
}
```

Para comprobar la autenticación mediante el token, en la parte del backend donde el usuario haga peticiones, hay que poner el siguiente código de ejemplo para recuperar usuario mediante el token:

```
app.get('/user', (req, res, next) => {
    let token = req.headers.token; //token
    jwt.verify(token, 'secretkey', (err, decoded) => {
        if (err) return res.status(401).json({
            title: 'unauthorized'
        })
        //token is valid
        User.findOne({ _id: decoded.userId }, (err, user) => {
            if (err) return console.log(err)
            return res.status(200).json({
                title: 'user grabbed',
                user: {
                    email: user.gmail,
                    name: user.usuario
                }
            })
        })
    })
})
```

-Front end

En primer se crea un formulario y al rellenarlo se pulsa el botón que llama a la función loginUser



```
<template style="background-color: black">
<div>
  <p><H1 id="title" style ="color: black;"><b>
    | Login
  </b></H1></p>
  <hr>
  <label>gmail</label><br>
  <input id="gmail" name="gmail" type="text" placeholder="Gmail del usuario" required v-mod
  <label>Contraseña</label><br>
  <input id="contraseña" name="contraseña" type="password" placeholder="Contraseña del usua
  <br><br>
  <hr>

  <button id="btn-login" v-on:click="loginUser" style="width:150px; height:45px; background
  {{ error }}
</div>
</template>

<script>
  import CreateAccountService from './CreateAccountService'.
```

En esta función se hace un post y si es satisfactorio, devolverá el token generado, que se guardará en el local storage.

```
async loginUser() {
let user = {
  gmail: this.gmail,
  contraseña: this.contrasena
}

axios.post('http://localhost:5000/login', user)
  .then(res => {
    //if successfull
    if (res.status === 200) {
      localStorage.setItem('token', res.data.token);
      this.$router.push('/peliculas');
    }
  }, err => {
    console.log(err.response);
    this.error = err.response.data.error
  })
this.usuario = '';
this.gmail = '';
this.contrasena = '';
```

Para terminar, cuando se loguea se accede a la funcionalidad de la página web que contiene una cabecera en verde con los enlaces para acceder a la cesta, las películas, la about page, logout y que da la bienvenida al usuario registrado, este es un componente que hace que cuando se elimine el jwt token de las cookies se haga el logout y que solo se pueda acceder a estas páginas si se está logueado, también cuando se pincha en logout, se elimina este token y se redirige al home.



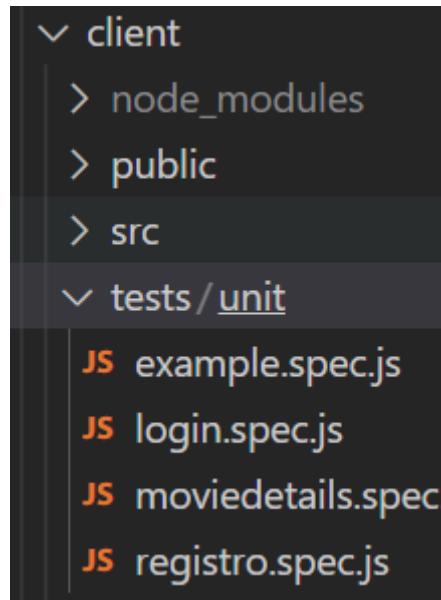
```
<template>
<div class="test1">
<nav>
| <router-link to="/peliculas">Peliculas</router-link>
</nav>
<nav>
| <router-link to="/cesta">Cesta</router-link>
</nav>
<nav>
| <router-link to="/about">About</router-link>
</nav>
<nav>
| <router-link to="/" @click="logout">Logout</router-link>
</nav>
<nav>
| <router-link to="">Bienvenido {{ usuario }}</router-link>
</nav>
</div>
</template>

created:function(){
  this.logeado();
},
methods: {
  logeado(){
    //user is not authorized
    if (localStorage.getItem('token') === null) {
      this.$router.push('/');
    }

    axios.get('http://localhost:5000/user', { headers: { token: localStorage.getItem('token') } })
      .then(res => {
        console.log(res.data);
        this.usuario = res.data.user.name;
      })
    },
  logout() {
    localStorage.clear();
    this.$router.push('/login');
  }
}
```

3.8. Tests.

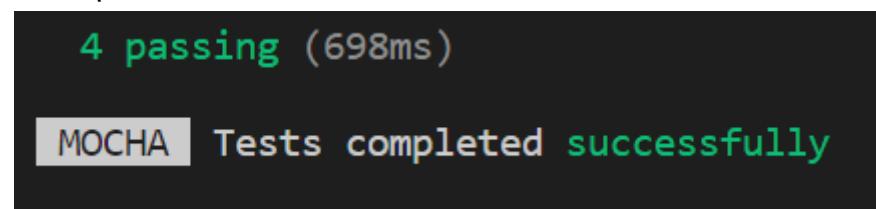
Hemos implementado tests con Mocha.js. Mocha es un framework de pruebas de JavaScript que se ejecuta en Node.js. Nos da la posibilidad de crear tanto tests síncronos como asíncronos de una forma muy sencilla. Nos proporciona muchas utilidades para la ejecución y el reporte de los tests. Hemos creado una carpeta tests dentro de la carpeta client y una subcarpeta unit donde hemos metido todos los tests realizados.



Estos tests realizan una búsqueda de distintos elementos y nos aseguramos de que muestren los textos que nosotros queremos que salgan, por ejemplo, dentro de la información de una película debe aparecer un apartado para título, descripción, año, país, duración y un botón que contenga “Añadir a la cesta”. Esto lo hemos realizado con las siguientes líneas de código:

```
expect(wrapper.find('#title').exists()).to.be.equal(true);
expect(wrapper.find('#description').exists()).to.be.equal(true);
expect(wrapper.find('#year').exists()).to.be.equal(true);
expect(wrapper.find('#country').exists()).to.be.equal(true);
expect(wrapper.find('#duration').exists()).to.be.equal(true);
expect(wrapper.find('#btn-cesta').exists()).to.be.equal(true);
expect(wrapper.find('#btn-cesta').text()).to.equal('Añadir a la cesta');
```

Hemos hecho esto para las páginas de inicio, login, registro y detalles de la película. Al ejecutar el comando *npm run test:unit* corre todos los tests y nos dice cuántos han fallado / pasado y donde están los errores. Si los tests no funcionaban debíamos corregir la página donde estuviera el error para asegurarnos de que todas las casillas / botones necesarios estuvieran ahí.





Al realizar los tests hemos tenido unos warnings con poca relevancia pero que hemos intentado solucionar para que funcione correctamente pero no hemos encontrado una solución por ninguna parte (no encontramos nada por foros de internet ni obtuvimos una respuesta de parte de los profesores).

```
[Vue warn]: Failed to resolve component: H1
If this is a native custom element, make sure to exclude it from component resolution via compilerOptions.isCustomElement.
  at <RegistroUser ref="VTU_COMPONENT" >
  at <VTUROOT>
```

3.9. Subir películas a la base de datos

-Backend:

Utilizaremos multer para pasar la imagen del frontend al backend.

```
// Subir película
// storage
const Storage = multer.diskStorage({
  destination: 'uploads',
  filename:(req,file, cb)=> {
    cb(null, file.originalname);
  },
})

const upload = multer({
  storage: Storage
}).single('file')
```

Después creamos un model con la información de la película que queremos tener y en imagen, que será la portada, pondremos un buffer y el tipo de la imagen.

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const filmSchema = new Schema({
  titulo:{
    type: String,
    unique: true
  },
  image:{
    data: Buffer,
    contentType: String
  },
  descripcion: String,
  ano: Number,
  duracion: Number,
  genero: String,
  director: String,
  precio: Number
})

const Film = mongoose.model('Film', filmSchema);
module.exports = Film;
```



Ahora en la ruta a la que llega la petición, creamos un nuevo objeto del tipo Film y lo rellenamos, en el data de la imagen pondremos el contenido de la imagen en binario.

```
app.post('/upload', (req, res) => {
  upload(req,res,(err)=>{
    if(err){
      console.log(err)
    }
    else{
      const newFilm = new Film({
        titulo: req.body.titulo,
        image:{
          data:fs.readFileSync(__dirname + '/../uploads/' + req.file.originalname),
          contentType:'image/jpeg'
        },
        descripcion: req.body.descripcion,
        ano: req.body.ano,
        duracion: req.body.duracion,
        genero: req.body.genero,
        director: req.body.director,
        precio: req.body.precio
      })
    }
  })
})
```

-Front End

En el frontend, crearemos un formulario con la información de la peli y una entrada del tipo file, con una referencia y el campo @change que llama a la función onFileSelected.

```
<label>Portada</label><br>
<input id="fileInput" name="portada" type="file" ref="file" @change="onFileSelected" ><br><br>
```

En esta función creamos un objeto del tipo FormData y hacemos un append de la información de la peli. Este objeto es el que enviaremos al backend para que lo suba a la base de datos.

```
async uploadFilm() {
  let film = new FormData();

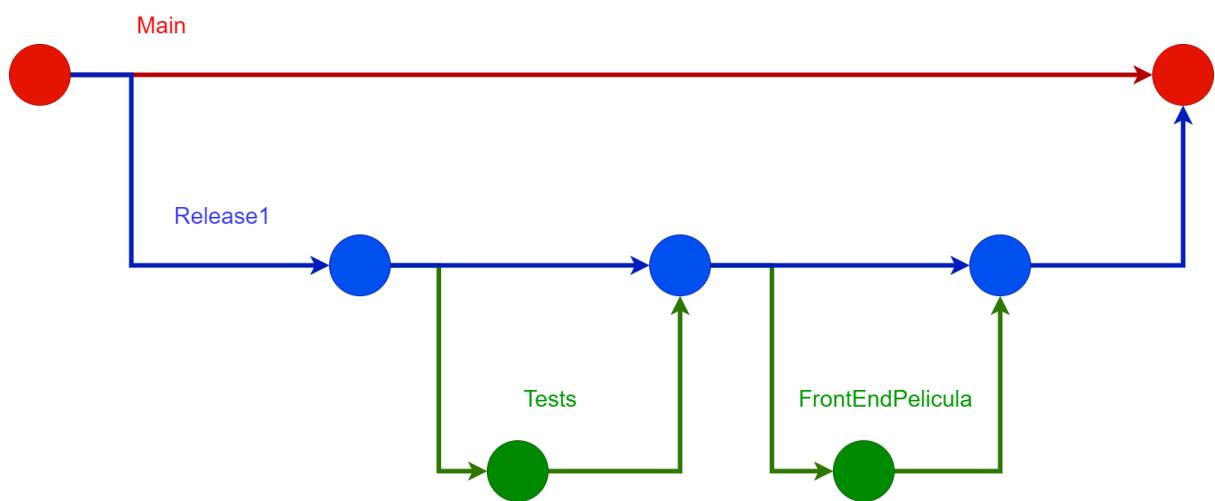
  film.append('titulo', this.titulo);
  film.append('descripcion', this.descripcion);
  film.append('ano', this.ano);
  film.append('duracion', this.duracion);
  film.append('genero', this.genero);
  film.append('director', this.director);
  film.append('precio', this.precio);

  film.append('file', this.portada);
  console.log(film);
  axios.post('http://localhost:5000/upload', film)
    .then(res => {
      //if successfull
      if (res.status === 200) {
        confirm("Película subida");
      }
    }, err => {
      console.log(err.response);
      this.error = err.response.data.error
    })
  this.error = '';
}
```



3.10. Github.

Hemos utilizado la herramienta Github para ir subiendo nuestro código, poder trabajar por separado cada individuo del equipo y tenerlo actualizado en todo momento. Para que cada uno pudiera ir cambiando partes del código sin que hubiera problemas a la hora de unirlo utilizamos varias ramas. Principalmente tenemos la rama **main** que fué la última rama unida y la entregada a los profesores de SyTW. Para asegurarnos de que lo entregado funcionara correctamente, creamos una subrama llamada **Release1** y fuimos generando subramas para cada tarea que completamos y uniéndola a la rama **Release1**. Finalmente, esta subrama la unimos a la rama principal. Este sería un esquema de ejemplo utilizado:



Y este sería un ejemplo de cómo íbamos haciendo este procedimiento correctamente por tarea:

Primero debíamos asegurarnos en qué rama estábamos con *git branch*. Para crear una nueva rama hay que poner *git branch nombre* y para cambiar a esa rama, *git checkout nombre*. Ya en la rama adecuada, ejecutamos un *git add* . procedido de un *git commit -m "comentario"* y finalmente *git push origin nombre*. Ahora dentro de github nos aparecía un pull request:



The screenshot shows the GitHub Pull Requests page. At the top, there are navigation links: Code, Issues, Pull requests (which is underlined), Actions, Projects, Security, Insights, and Settings. A yellow banner at the top indicates that 'FrontEnd' had recent pushes less than a minute ago. Below the banner, there are search filters: 'Filters' (with 'is:pr is:open'), a search bar ('Q. is:pr is:open'), and buttons for 'Labels 9', 'Milestones 0', and 'New pull request'. Below the filters, there are status filters: '0 Open' and '1 Closed'. To the right are dropdown menus for 'Author', 'Label', 'Projects', 'Milestones', 'Reviews', 'Assignee', and 'Sort'. A green button at the top right says 'Compare & pull request'.

Al clicar, veríamos la página para el pull request que debemos especificar qué ramas debemos unir:

The screenshot shows the 'Comparing changes' page. It asks to choose two branches to see what's changed or to start a new pull request. It notes that if needed, one can also compare across forks. Below this, it shows the 'base: Release1' and 'compare: registroBackend' branches. A green checkmark indicates that 'Able to merge. These branches can be automatically merged.'

Se compara y se hace el pull request revisando los cambios hechos en los ficheros implicados:

The screenshot shows the GitHub Pull Request review interface. It displays a list of files changed, with 'client/package-lock.json' expanded to show a diff. Below it, 'client/src/App.vue' is shown with a detailed diff. The diff highlights changes made by comparing the 'base' branch ('Release1') with the 'compare' branch ('registroBackend'). The code changes include the addition of a logo image and its styling.

Finalmente aceptamos, le damos a merge y borramos la rama utilizada.

The screenshot shows the GitHub Pull Request history. It lists comments, commits, and actions. A comment from 'alu0101601830' is shown, followed by a commit from 'Creando el FrontEnd de la página principal, de Login y del Registro' (Scd1816). An action shows a review request from 'alu0101601830' to 'Daniel-Arbelo'. Finally, a commit merges the pull request into the 'Release1' branch. A message at the bottom states: 'Pull request successfully merged and closed. You're all set—the FrontEnd branch can be safely deleted.' A 'Delete branch' button is visible.

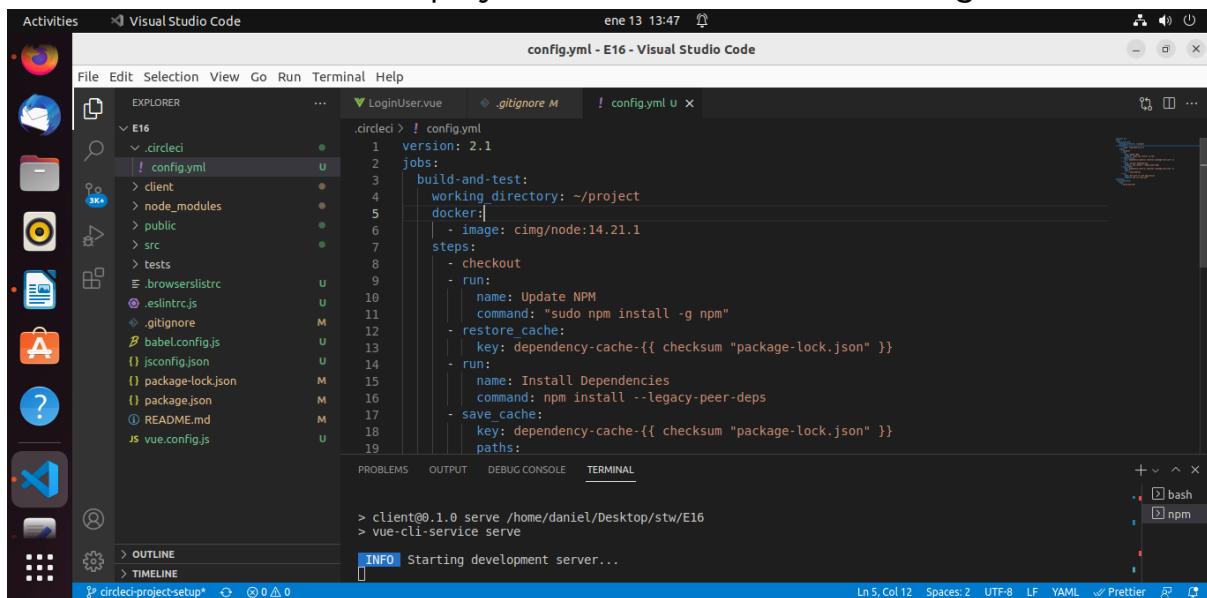


Fuimos haciendo eso para cada tarea y finalmente hicimos lo mismo con la rama Release1 a la rama main.

3.11. Integración continua.

En primer lugar tendremos que hacer una copia del repositorio para ponerlo público, para poder utilizar la parte gratis de travis ci, también tendremos que poner el código del front, que es donde se ejecutan los test en el dir raíz, y el código del servidor en una carpeta, ya que en nuestro proyecto lo tenemos al revés.

Haremos un remote add del proyecto al nuevo directorio con origin2.



The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure with a folder named "E16" containing a ".circleci" directory which contains a "config.yml" file.
- Code Editor:** Displays the content of the "config.yml" file:

```
version: 2.1
jobs:
  build-and-test:
    working_directory: ~/project
    docker:
      - image: cimg/node:14.21.1
    steps:
      - checkout
      - run:
          name: Update NPM
          command: "sudo npm install -g npm"
      - restore_cache:
          key: dependency-cache-{{ checksum "package-lock.json" }}
      - run:
          name: Install Dependencies
          command: npm install --legacy-peer-deps
      - save_cache:
          key: dependency-cache-{{ checksum "package-lock.json" }}
    paths:
```
- Terminal:** Shows the command being run: "client@0.1.0 serve /home/daniel/Desktop/stw/E16" and "vue-cli-service serve". It also displays an INFO message: "Starting development server...".

Se crea un documento config.yml dentro del dir .circleci. Después de esto se asocia el directorio en circle ci, y cada vez que se haga un push, se ejecutarán las pruebas como tenemos definido en el doc config.yml.

4. Resultados finales.

Estas son las páginas de nuestra página web



4.1. Página de entrada.



¡Bienvenido a nuestro servicio de películas online!

Encontrarás un amplio catálogo de películas para elegir
Por favor selecciona o continua para Registrarte o Iniciar Sesión

Register

Login

Este es el home de nuestra página web, contiene una introducción a esta misma, así como dos botones, que redirigen al registro y al login, si no se está logueado, no se puede acceder al resto de páginas.

4.2. Login y Registro.

[Home](#)

Página de Registro

Registro

Email

Usuario

Contraseña

Register

Como podemos ver el registro tiene un formulario que una vez rellenado, al pulsar el botón de register se subirá la información a la base de datos.



[Home](#)

Página para iniciar sesión

Login

gmail
Gmail del usuario

Contraseña
Contraseña del usuario

[Login](#)

El login contiene un formulario, pero más corto que hace una petición al backend, el cual si es satisfactorio devuelve un token, cuando este se recibe se redirige a la página con el catálogo de películas.

4.3. Catálogo de películas y página para subirlas

Una vez logueados nos redirige a la página con el catálogo de las películas sacadas del servidor.

Películas Disponibles

[Pulp Fiction](#)

[El padrino](#)

[Reservoir Dogs](#)

[Snatch](#)

[Resacón](#)

Para subirlas tiene que estar logueado como un admin y podrá acceder a la página de upload, que contiene un formulario que luego enviará la información a la base de datos para subirla.



Página para subir películas al servidor

Rellena la siguiente información

Titulo

Portada

Descripción

Año

Duración

Género

Director

Precio

Subir

4.4. Detalles de una película.

Cuando en el catálogo de películas, se pincha sobre una aparece la información de esta como un botón para añadirla a la cesta.

Películas Cesta About Logout Bienvenido daniel

Detalles de la Película



Pulp Fiction

Historias de dos matones, un boxeador y una pareja de atracadores de poca monta envueltos en una violencia espectacular e irónica.

| | | | |
|--------|------|----------|-------------------|
| Year | 1994 | Duración | 154 |
| Precio | 10 € | Director | Quentin Tarantino |

[Añadir a la cesta](#)

4.5. Cesta del usuario.

En la cesta aparecen las películas añadidas con su precio y el precio total de las pelis que tenemos, así como un botón para proceder a la compra introduciendo la dirección así como un método de pago.



Peliculas Cesta About Logout Bienvenido daniel

Página de la cesta de la compra


[Pulp Fiction](#)
10 €


[El padrino](#)
9 €

Precio del pedido total
19 €
[Comprar](#)

5. Conclusión.

5.1. Conclusión Formal.

La aplicación ha acabado con lo básico para que funcione correctamente con los objetivos que nos habíamos marcado, aunque si hubiéramos tenido más tiempo, alomejor podríamos haber implementado más cosas, como compra de series también, o libros, y aplicar filtros a las búsquedas, para que se hicieran búsquedas más personalizadas, así como introducir un catálogo más amplio de películas, una página de perfil de usuario, incluso mejorar un poco el estilo de la página.

5.2. Conclusión Personal.

En conclusión, esta asignatura nos ha servido para aprender a utilizar la arquitectura MEVN, así como interconectar front, backend y con una base de datos e introduciéndonos más en el desarrollo de páginas web. Y así si en un futuro tenemos que hacer un desarrollo web con cualquier otra tecnología, o arquitectura diferente a la que hemos utilizado, por lo menos sabremos cómo enfrentarnos al proyecto, ya que nos hemos tenido que buscar bastante la vida con una pequeña guía por parte de los profesores, cosa que en otras asignaturas no pasa ya que te lo dan todo masticado y alomejor no se asimilan tanto los contenidos como en esta, en la que hemos tenido que batallar un poco más.

6. Despliegue de Aplicación.

No hemos podido desplegar nuestra aplicación con ninguna plataforma entonces para probarla se debe hacer dentro de la carpeta client y ejecutar el



comando `npm run serve` y comprobar la funcionalidad con un sistema de ordenadores montado como hicimos en las prácticas individuales.

7. Referencias.

1. [https://www.educative.io/answers/what-is-meavn-stack](https://www.educative.io/answers/what-is-mevn-stack)
2. <https://carlosazaustre.es/como-funciona-flux>
3. <https://www.paradigmadigital.com/dev/testeando-javascript-mocha-chai/>