FUTMATCH

Informe Sprint #4

Objeto: En este sprint se desarrolla el API Gateway de la aplicación, dónde viene converger la comunicación de los microservicios. Para ello, se detalla en este documento el proceso SCRUM seguido en el software Jira de Atlasias, se muestra el código desarrollado, las pruebas realizadas y se evidencia el manejo de control de versiones en el repositorio online.

Jira

En el software de gestión JIRA, vinculamos todas las incidencias de la creación del API-GATEWAY, de acuerdo a las guías de la actividad práctica.

La primera instancia que creamos fue la de la creación del proyecto en VS-CODE, con las respectivas carpetas y archivos.

Después adaptamos el código de la práctica del banco, y de acuerdo a los modelos creados en los microservicios de autenticación, jugador y convocatoria, y creamos los archivos para los TypeDef, los DataSource y los Resolvers.

Desplegamos a través de la consola el código a Heroku, para posteriormente realizar las respectivas pruebas con Apollo-Server

CIF-30 Realizar pruebas de verificación
Ciclo IV - FutMatch

CIF-29 Despliegue del API Gateway
Ciclo IV - FutMatch

CIF-28 Adaptar el código de los Resolvers a la aplicación
Ciclo IV - FutMatch

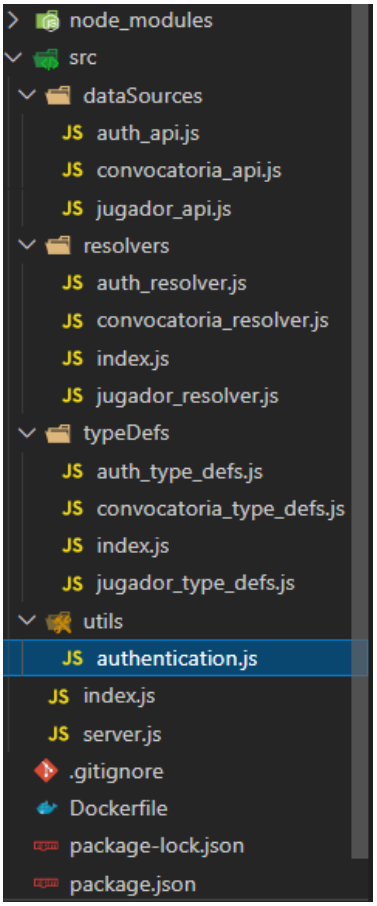
CIF-27 Adaptar el código de los DataSource a la aplicación
Ciclo IV - FutMatch

CIF-26 Adaptar el código de los TypeDef a la aplicación
Ciclo IV - FutMatch

CIF-25 Crear la estructura de carpetas en VSCODE
Ciclo IV - FutMatch

Código

Se realizó el orquestador en Visual Studio siguiendo los lineamientos de clase, de esa manera la estructura de este quedó de la siguiente manera:

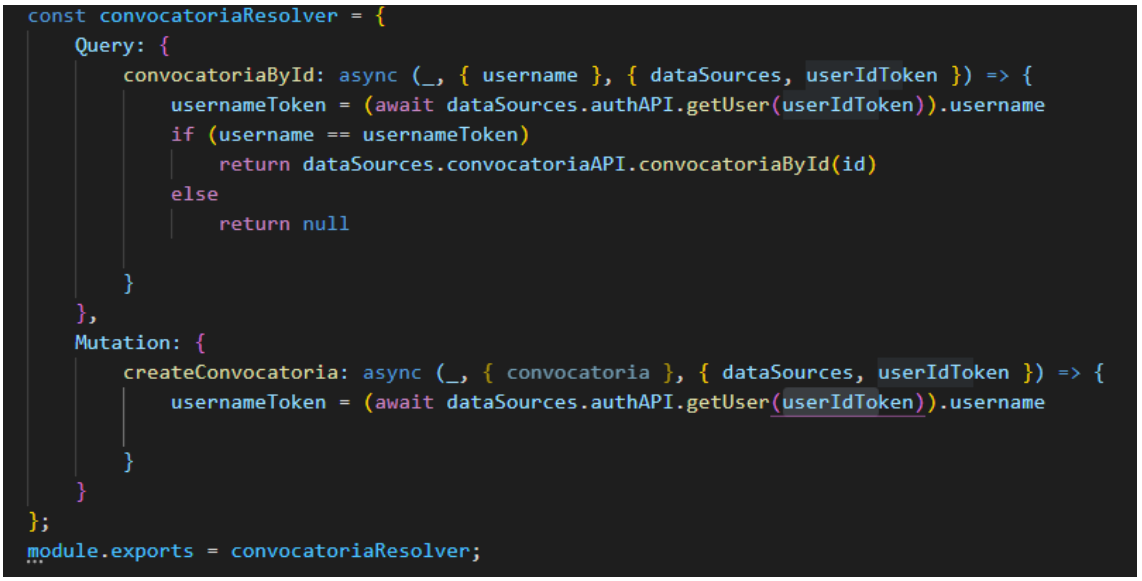


En los typeDefs se definen los tipos de los atributos, se ilustran solamente los de convocatoria y jugador ya que los de autenticación e index siguen los mismos lineamientos del trabajo en clase.

```
const { gql } = require('apollo-server');
const convocatoriaTypeDefs = gql `
    type Convocatoria {
        id: String!
        jugEquip1: [Int]
        jugEquip2: [Int]
        fecha: String!
        hora: String!
        localidad : String!
        posiciones: String
    }
    input ConvocatoriaInput {
        jugEquip1: [Int]
        jugEquip2: [Int]
        fecha: String!
        hora: String!
        localidad : String!
        posiciones: String
    }
    extend type Query {
        convocatoriaById(id: String!): Convocatoria
    }
    extend type Mutation {
        createConvocatoria(convocatoria: ConvocatoriaInput!): Convocatoria
    }
`;
module.exports = convocatoriaTypeDefs;
```

```
const jugadorTypeDefs = gql `
    type Jugador {
        nombre: String!
        fechaNacimiento: String!
        convJugadas: Int
    }
    extend type Query {
        jugadorByNombre(username: String!): Jugador!
    }
`;
module.exports = jugadorTypeDefs;
```

En Resolvers que es dónde se codifican las operaciones, se muestra el código de convocatoria y jugador ya que autenticación e index siguen los mismos lineamientos del trabajo en clase.



```
const convocatoriaResolver = {
    Query: {
        convocatoriaById: async (_, { username }, { dataSources, userIdToken }) => {
            usernameToken = (await dataSources.authAPI.getUser(userIdToken)).username
            if (username == usernameToken)
                return dataSources.convocatoriaAPI.convocatoriaById(id)
            else
                return null

        }
    },
    Mutation: {
        createConvocatoria: async (_, { convocatoria }, { dataSources, userIdToken }) => {
            usernameToken = (await dataSources.authAPI.getUser(userIdToken)).username

        }
    }
};
module.exports = convocatoriaResolver;
```

```
const jugadorResolver = {
    Query: {
        jugadorByNombre: async (_, { username }, { dataSources, userIdToken }) => {
            usernameToken = (await dataSources.authAPI.getUser(userIdToken)).username
            if (username == usernameToken)
                return await dataSources.jugadorAPI.jugadorByNombre(username)
            else
                return null

        },
    },
    Mutation: {}
};
module.exports = jugadorResolver;
```

Puede evidenciarse que la creación del jugador no se encuentra en la parte de jugador, ya que esto se realiza en la parte de autenticación (al crear el usuario, también se crea el perfil). Además se puede filtrar convocatoria por Id, perfil de jugador por el nombre y la creación de la convocatoria.

A continuación se ilustra el código de convocatoria y jugador de los dataSources que permiten conectarse con los MS.

```
const { RESTDataSource } = require('apollo-datasource-rest');
const serverConfig = require('../server');

class ConvocatoriaAPI extends RESTDataSource {
    constructor() {
        super();
        this.baseURL = serverConfig.convocatoria_api_url;
    }
    async createConvocatoria(convocatoria) {
        convocatoria = new Object(JSON.parse(JSON.stringify(convocatoria)));
        return await this.post('/convocatorias', convocatoria);
    }
    async convocatoriaById(id) {
        return await this.get(`/convocatorias/${id}`);
    }
}
module.exports = ConvocatoriaAPI;
```

```
const { RESTDataSource } = require('apollo-datasource-rest');
const serverConfig = require('../server');

class JugadorAPI extends RESTDataSource {
    constructor() {
        super();
        this.baseURL = serverConfig.jugador_api_url;
    }
    async createJugador(jugador) {
        jugador = new Object(JSON.parse(JSON.stringify(jugador)));
        return await this.post('/jugador', jugador);
    }
    async jugadorByNombre(username) {
        return await this.get(`/jugador/${username}/`);
    }
}
module.exports = JugadorAPI;
```

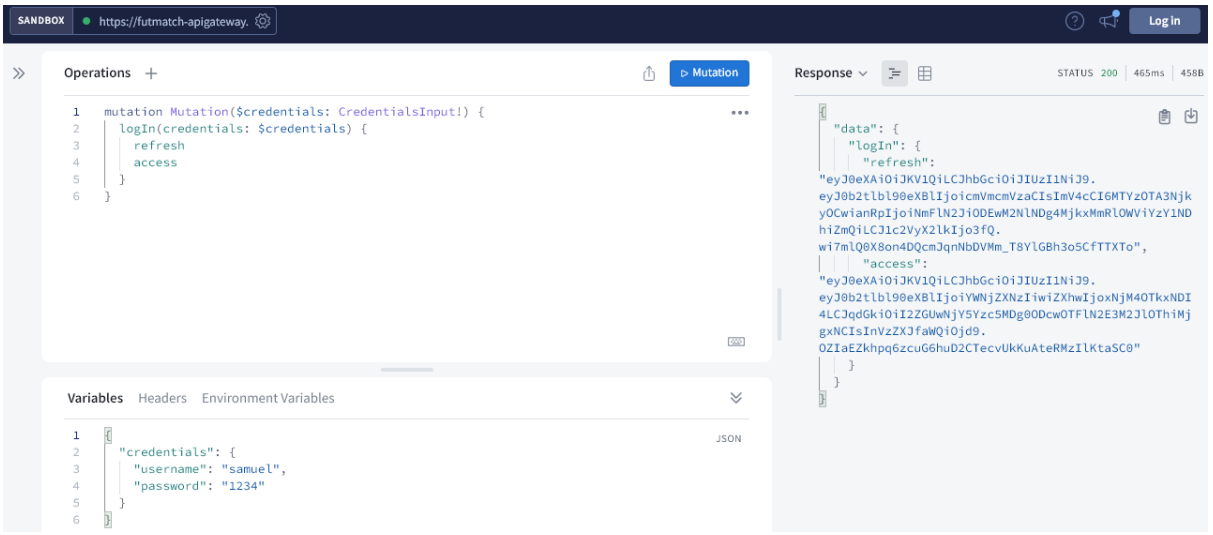
Se puede ver cómo estos códigos permiten la creación y el filtro de jugador y convocatoria.

Los demás archivos del documento siguen el mismo código que el ejemplo desarrollado en clase. Sin embargo, quisiéramos mostrar los links del server.js dónde se explicita la conexión con los microservicios desarrollados anteriormente.

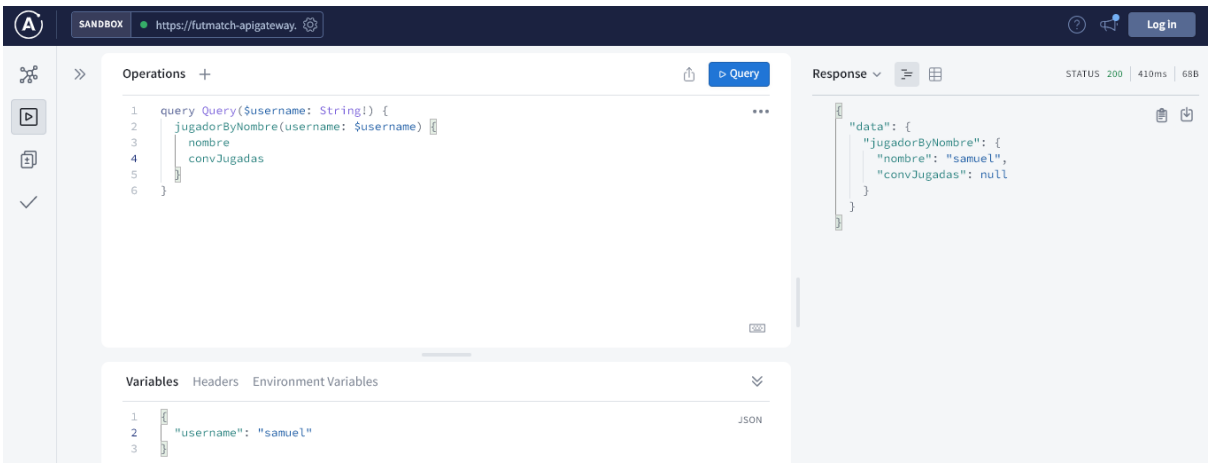
```
module.exports = {
    auth_api_url: 'https://futmatch-authms.herokuapp.com',
    jugador_api_url: 'https://futmatch-jugador-ms.herokuapp.com',
    convocatoria_api_url: 'https://futmatch-convocatoria-ms.herokuapp.com',
    };
```

Pruebas

Para realizar las pruebas sobre las peticiones correspondientes al API Gateway, ingresamos al servidor GraphQL dispuesto por Apollo, construyendo las respectivas peticiones. En este caso, la Mutation logIn para realizar una autenticación de usuario:

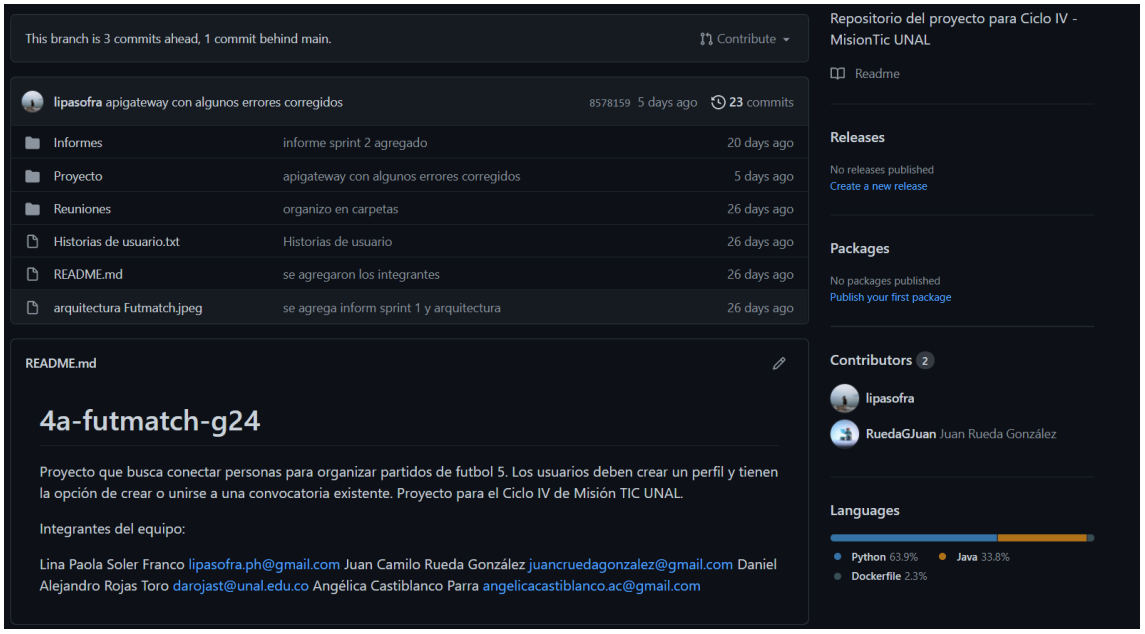


Una vez se obtienen los tokens de autenticación, realizamos la construcción de la Query jugadorByNombre, teniendo en cuenta que es una petición que se puede realizar una vez el usuario está autenticado. Para ello, añadimos el access token a la petición en la pestaña Headers y obtenemos la siguiente salida requerida:

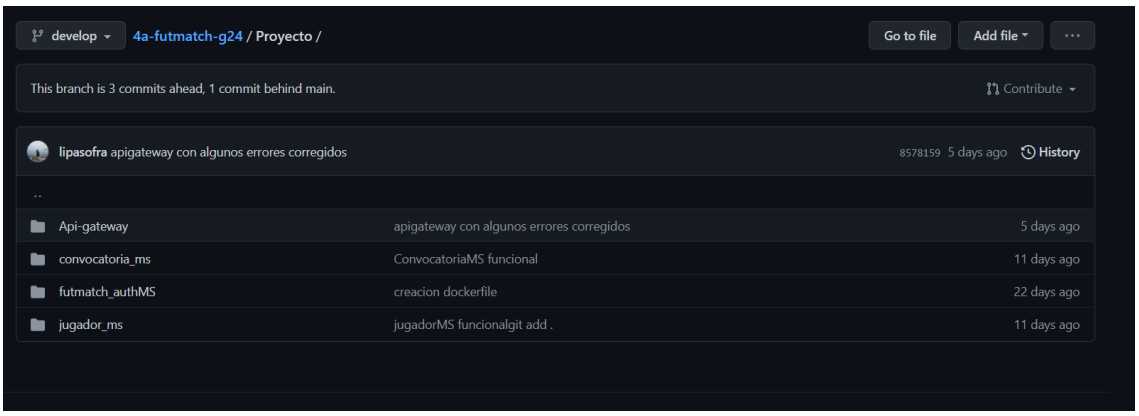


Repositorio

Dado que el repositorio que hemos estado utilizando esta alojado en Github, sincronizamos todo a la carpeta de desarrollo develop, aca vemos la carpeta general del proyecto:

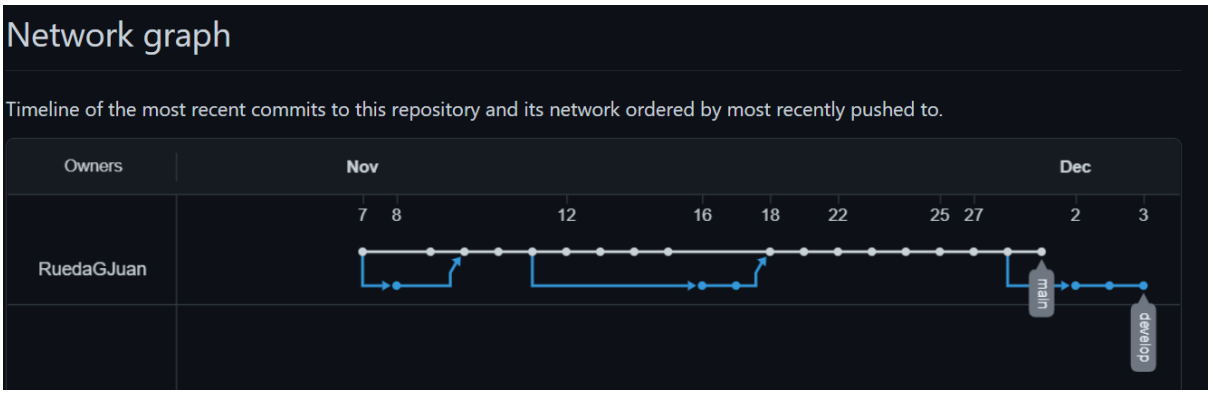


Dejando todavía la carpeta en main intacta, dado que tuvimos percances en las pruebas del orquestador no hemos querido todavía generar el push, hasta que el código pase en su totalidad todas pruebas. Aca podemos ver cómo se estructura la carpeta Proyecto:



En la carpeta del orquestador Api-gateway encontramos el formato según como se definió en la guías:

Se muestran los mensajes de modificación, mostrando que son una "segunda versión", dado que tuvimos ciertos errores que nos hizo caer en cuenta las pruebas, que fueron corregidos después en la segunda versión. Al final el historial del repositorio va de la siguiente manera:



Siguiendo los siguientes Commits: