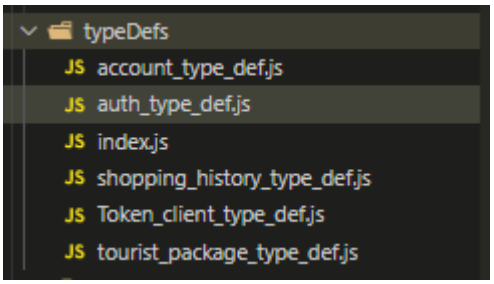
API GATEWAY

Repositorio: https://github.com/DavidCastro88/4a-apigateway.git
Repositorios con los docs: https://github.com/DavidCastro88/4a-docs

Context de autenticación:

```
const { ApolloError } = require('apollo-server');
const serverConfig = require('../server');
const fetch = require('node-fetch');
const authentication = async ({ req }) => {
    const token = req.headers.authorization || '';
    if (token == '')
        return { userIdToken: null }
    else {
        try {
            let requestOptions = {
            method: 'POST', headers: { "Content-Type": "application/json" },
            body: JSON.stringify({ token }), redirect: 'follow'
            };
            let response = await fetch(
            `${serverConfig.auth_shopping_url}/verifyToken/`,
            requestOptions)
            if (response.status != 200) {
            console.log(response)
            throw new ApolloError(`SESION INACTIVA - ${401}` + response.status, 401)
            }
            return { userIdToken: (await response.json()).UserId };
    }
        catch (error) {
            throw new ApolloError(`TOKEN ERROR: ${500}: ${error}`, 500);
    }
    }
}
module.exports = authentication;
```

Type Defs:



```
const { gql } = require('apollo-server');
const accountTypeDefs = gql `
    type Account {
        username: String!
        lastChange: String!
    }
    extend type Query {
        accountByUsername(username: String!): Account
    }
`;
module.exports = accountTypeDefs;
```

```
const { gql } = require('apollo-server');
const authTypeDefs = gql `
    type Tokens {
        refresh: String!
        access: String!
    }
    type Access {
        access: String!
    }
    input CredentialsInput {
        username: String!
        password: String!
    }
    input SignUpInput {
        username: String!
        password: String!
        name: String!
        email: String!
        cellphone: String!
    }
    type UserDetail {
        id: Int!
        username: String!
        password: String!
        name: String!
        email: String!
        cellphone: String!
    }
    type Mutation {
        signUpUser(userInput :SignUpInput): Tokens!
        logIn(credentials: CredentialsInput!): Tokens!
        refreshToken(refresh: String!): Access!
    }
    type Query {
        userDetailById(userId: Int!): UserDetail!
    }

`;
module.exports = authTypeDefs;
```

```
const { gql } = require('apollo-server');
const shoppinghistoryTypeDefs = gql `
    type Shoppin_history{
        Id_Purchase: Int!
        Id_user: Int!
        Id_tourist_package: Int!
        Amount_package: String!
        Total_value: String!
        Status: String!
        Shopping_Date: String!
    }
    input Shoppinghistory_input{
        Id_user: Int!
        Id_tourist_package: Int!
        Amount_package: String!
        Total_value: String!
        Status: String!
    }
    extend type Mutation{
        createShoppingRegister(Shoppingregister: Shoppinghistory_input!): Shopping_history
    }
    type Query{
        shoppinghistorybyId_Purchase(Id_Purchase:Int!):Shoppin_history!
    }
    type Query{
        shoppinghistoryUserbyID(userId: Int!): Shoppin_history!
    }
`;
module.exports = shoppinghistoryTypeDefs;
```

```
const { gql } = require('apollo-server');
const touristPackageTypeDefs = gql `
    type  Tourist_package{
        Id_tourist_package: Int!
        Destination_place: String!
        Hotel: String!
        Days_stay: Int!
        Nights_stay: Int!
        Feeding: String!
        Guides: String!
        Price_Person: Int!
    }
    input Tourist_package_input{
        Destination_place: String!
        Hotel: String!
        Days_stay: Int!
        Nights_stay: Int!
        Feeding: String!
        Guides: String!
        Price_Person: Int!
    }
    extend type Mutation{
        createTourist_package(TouristPackageregister: Tourist_package_input!): Tourist_package
    }
    type Query{
        Tourist_packagebyId(Id_tourist_package: Int!):Tourist_package!
    }

`;
module.exports = touristPackageTypeDefs;
```

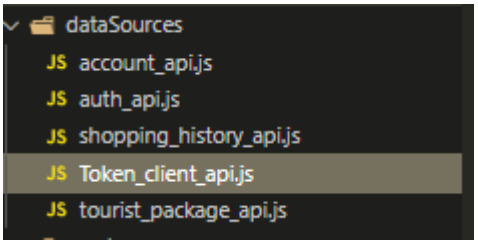
```graphql
const { gql } = require('apollo-server');
const tokenClientTypeDefs = gql `
    type TokenClient{
        _id: Int!
        nombre: String!
        telefono: String!
        email: String!
        inquietudes: String!
        comentario: String!
        sol_activa: String!
    }
    input TokenClient_input{
        nombre: String!
        telefono: String!
        email: String!
        inquietudes: String!
        comentario: String!
        sol_activa: String!
    }
    extend type Mutation{
        createTokenRegister(Tokenregister: TokenClient_input!): TokenClient
    }
    type Query{
        tokenClientbyId(_id: Int!):TokenClient!
    }
    type Query{
        tokenClientbyEmail(email: String!): TokenClient!
    }
`;
module.exports = tokenClientTypeDefs;
```

Index de los typedefs:

src > typeDefs > JS index.js > ...

```graphql
//Se llama al typedef (esquema) de cada submodulo
const accountTypeDefs = require('./account_type_defs');
const authTypeDefs = require('./auth_type_defs');
const shoppinghistoryTypeDefs = require('./shopping_history_type_defs');
const tokenClientTypeDefs= require('./token_client_type_defs');
const touristPackageTypeDefs= require('./tourist_package_type_defs')
//Se unen
const schemasArrays = [authTypeDefs,accountTypeDefs, shoppinghistoryTypeDefs,tokenClientTypeDefs,touristPackageTypeDefs];
//Se exportan
module.exports = schemasArrays;
```

Data_sources:



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

        constructor() {
            super();
            this.baseURL = serverConfig.auth_shopping_api_url;
    }
        async getShoppingHistorybyUser(userId) {
            return await this.get(`/shoppinghistory/user/${userId}/`);
    }
        async getShoppingHistorybyId_Purchase(Id_Purchase) {
            return await this.get(`/shoppinghistory/${Id_Purchase}/`);
    }
        async createshoppingRegister(Shoppingregister) {
            Shoppingregister = new Object(JSON.parse(JSON.stringify(Purchase)));
            return await this.post(`/shoppinghistory/`, Purchase);

}}
module.exports = ShoppingHistoryAPI;
```

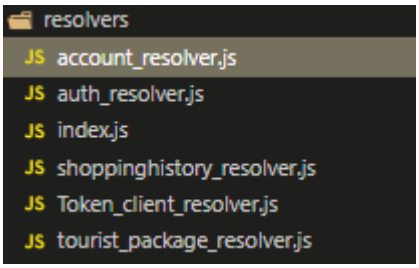
```
const { RESTDataSource } = require('apollo-datasource-rest');
const serverConfig = require('../server');
class AccountAPI extends RESTDataSource {
    constructor() {
        super();
        this.baseURL = serverConfig.account_api_url;
    }
    async createAccount(account) {
        account = new Object(JSON.parse(JSON.stringify(account)));
        return await this.post('/accounts', account);
    }
    async accountByUsername(username) {
        return await this.get(`/accounts/${username}`);
    }
}
module.exports = AccountAPI;
```

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

        constructor() {
            super();
            this.baseURL = serverConfig.auth_shopping_api_url;
    }
        async createUser(user) {
            user = new Object(JSON.parse(JSON.stringify(user)));
            return await this.post(`/user/`, user);
    }
        async getUser(userId) {
            return await this.get(`/user/${userId}/`);
    }
        async authRequest(credentials) {
            credentials = new Object(JSON.parse(JSON.stringify(credentials)));
            return await this.post(`/login/`, credentials);
    }
        async refreshToken(token) {
            token = new Object(JSON.parse(JSON.stringify({ refresh: token })));
            return await this.post(`/refresh/`, token);
    }
}
module.exports = AuthAPI;
```

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

        constructor() {
            super();
            this.baseURL = serverConfig.auth_shopping_api_url;
    }
        async getTouristPackgebyId(Id_tourist_package) {
            return await this.get(`/tourist_package/${Id_tourist_package}/`);
    }

        async createTouristPackage(Tourist_Package) {
            Tourist_Package = new Object(JSON.parse(JSON.stringify(Tourist_Package)));
            return await this.post(`/tourist_package/`, Tourist_Package);

}}
module.exports = TouristPackageAPI;
```

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

        constructor() {
            super();
            this.baseURL = serverConfig.Tokenclient_api_url;
    }
        async getTokenClientbyEmail(Email) {
            return await this.get(`/contacto/email/${Email}/`);
    }
        async getTokenClientbyId(_id) {
            return await this.get(`/contacto/${_id}/`);
    }
        async createTokenClient(Token_inf) {
            Token_inf = new Object(JSON.parse(JSON.stringify(Token_inf)));
            return await this.post(`/contacto/`, Token_inf);

}}
module.exports = TokenClientAPI;
```

Resolvers:




```
const accountResolver = {
    Query: {
        accountByUsername: async(_, { username }, { dataSources, userIdToken }) => {
            usernameToken = (await dataSources.authAPI.getUser(userIdToken)).username
            if (username == usernameToken)
                return await dataSources.accountAPI.accountByUsername(username)
            else
                return null
        },
    },
    Mutation: {}
};
module.exports = accountResolver;
```

```
const usersResolver = {
    Query: {
        userDetailById: (_, { userId }, { dataSources, userIdToken }) => {
        if (userId == userIdToken)
            return dataSources.authAPI.getUser(userId)
        else
            return null
        },
    },
    Mutation: {
        signUpUser: async(_, { userInput }, { dataSources }) => {
            const accountInput = {
                username: userInput.username,
                lastChange: (new Date()).toISOString()
    }
            await dataSources.accountAPI.createAccount(accountInput);
            const authInput = {
                username: userInput.username,
                password: userInput.password,
                name: userInput.name,
                email: userInput.email,
                cellphone: userInput.cellphone,
            }
            return await dataSources.authAPI.createUser(authInput);
    },
        logIn: (_, { credentials }, { dataSources }) =>
            dataSources.authAPI.authRequest(credentials),
        refreshToken: (_, { refresh }, { dataSources }) =>
            dataSources.authAPI.refreshToken(refresh),
    }
};
module.exports = usersResolver;
```

```
const shoppinghistoryResolver = {
    Query: {
        shoppinghistorybyId_Purchase: async(_, { Id_Purchase }, { dataSources, ShoppingId }) => {
            if (Id_Purchase == ShoppingId)
                return await dataSources.ShoppingHistoryAPI.getShoppingHistorybyId_Purchase(Id_Purchase)
            else
                return null
        },
        shoppinghistoryUserbyID: async(_, { userId }, { dataSources, ShoppingId }) => {
            if (userId == ShoppingId)
                return await dataSources.ShoppingHistoryAPI.getShoppingHistorybyUser(userId)
            else
                return null
        },
    },
    Mutation: {
        createShoppingRegister: async(_, { Shoppingregister }, { dataSources }) => {
            return await dataSources.ShoppingHistoryAPI.createshoppingRegister(Shoppingregister)
    }}
};
module.exports = shoppinghistoryResolver;
```

```
const TokenclientResolver = {
    Query: {
        tokenClientbyId: async(_, { _id }, { dataSources}) => {
            return await dataSources.TokenClientAPI.getTokenClientbyId(_id)
        },
        tokenClientbyEmail: async(_, { email }, { dataSources}) => {
            return await dataSources.TokenClientAPI.getTokenClientbyEmail(email)
        },
    },
    Mutation: {
        createTokenRegister: async(_, { Tokenregister }, { dataSources }) => {
            return await dataSources.TokenClientAPI.createTokenClient(Tokenregister)
    }}
};
module.exports = TokenclientResolver;
```

```
const touristpackageResolver = {
    Query: {
        Tourist_packagebyId: async(_, { Id_tourist_package }, { dataSources}) => {
            return await dataSources.TouristPackageAPI.getTouristPackgebyId(Id_tourist_package)
        },
    },
    Mutation: {
        createTourist_package: async(_, { TouristPackageregister }, { dataSources }) => {
            return await dataSources.TouristPackageAPI.createTouristPackage(TouristPackageregister)
    }}
};
module.exports = touristpackageResolver;
```

Index de los resolvers:

```
const authResolver = require('./auth_resolver');
const accountResolver = require('./account_resolver');
const shoppinghistoryResolver = require('./shoppinghistory_resolver');
const TokenclientResolver = require('./Token_client_resolver');
const touristpackageResolver = require('./tourist_package_resolver');
const lodash = require('lodash');
const resolvers = lodash.merge(authResolver,accountResolver,shoppinghistoryResolver,TokenclientResolver,touristpackageResolver);
module.exports = resolvers;
```

Trabajo en Jira: