Tests sur notre API

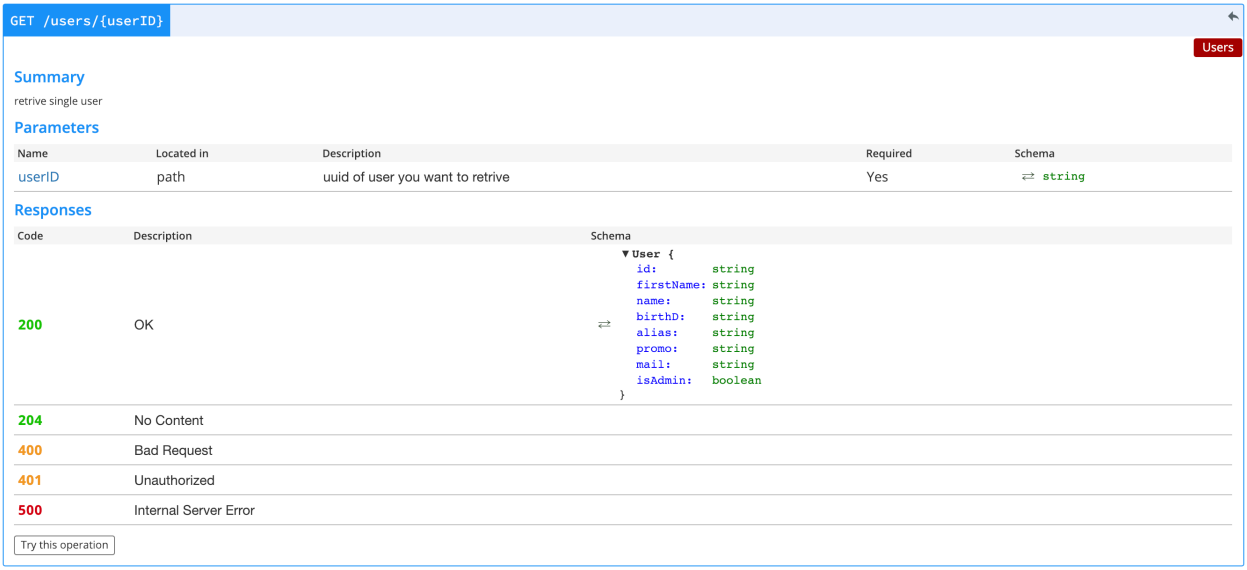
Anaël Bourgneuf et Timothé Iskander

Qu'est ce qu'elle fait ?

Le principe est de gérer les principaux outils utiles à la classe.
Pour cela nous aurons en base des utilisateurs, des promos, des "idées", des "reviews" (pour acceder aux formulaires de retours sur les modules de cours) et des événements.

Les premiers tests :

Les premiers chemins créés permettaient de récupérer les utilisateurs et autres objets stockés en base.
Voici par exemple la documentation du chemin permetant de récuperer un utilisateur par son uuid.



GET /users/{userID}

Users

Summary

retrive single user

Parameters

Name	Located in	Description	Required	Schema
userID	path	uuid of user you want to retrive	Yes	string

Responses

Code	Description	Schema
200	OK	User { id: string, firstName: string, name: string, birthD: string, alias: string, promo: string, mail: string, isAdmin: boolean }
204	No Content	
400	Bad Request	
401	Unauthorized	
500	Internal Server Error	

Try this operation

Voici la route empruntée sur le serveur.

main.js

```
// et tous les chemins principaux de l'api
app.use( fn: '/users', usersRoutes)
app.use( fn: '/promos', promosRoutes)
app.use( fn: '/reviews', reviewsRoutes)
app.use( fn: '/ideas', ideasRoutes)
app.use( fn: '/events', eventsRoutes)
```

usersRoutes.js

```
//Décalaration de routes pour /Users
app.route('/')
    .get(getUsersList)
    .post(addUser);

app.route('/:id')
    .get(getUserById)
    .put(updateUser)
    .delete(deleteUser);

app.route('/connect/:id')
    .get(connect)
```

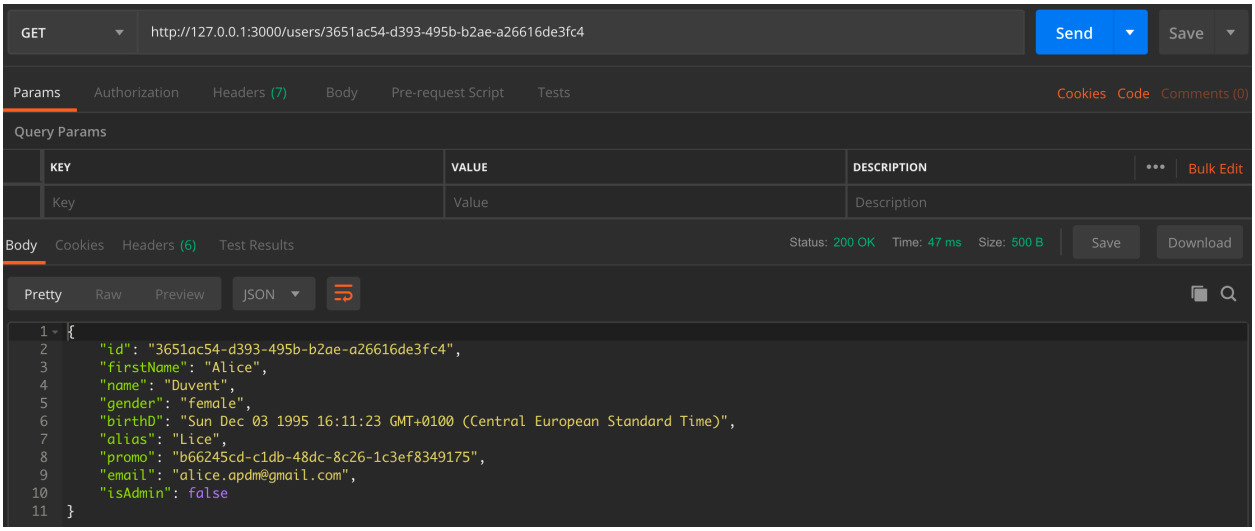
usersControler.js

```
// pour recuperer un utilisateur specifique par son identifiant
exports.getUserById = (req, res) => {
    console.log(req.params)
    const user = users.getUserById(req.params.id)
    if (user) {
        res.status(200).send(user)
    } else {
        res.status(404).send('user not found')
    }
    res.end()
}
```

usersRepository.js

```
getUserById (id) {
    //console.log(id)
    const i = this.items.findIndex(el => el.id === id)
    if (i !== -1) {
        return this.items.find(el => el.id === id)
    } else {
        return null
    }
}
```

Et maintenant la réponse du serveur.



Connaissant les données qui nous seront renvoyés lors de cette requete, on peut créer un test afin de verifier qu'elle reste stable au fur et a mesure du développement.
Le test est le suivant :

```
var expectedUserSchema = {type: 'object', properties: {id: {type: 'string'}, firstName: {type: 'string'}, name:{type: 'string'},
 birthD:{type: 'string'}, alias:{type: 'string'}, promo: {type: 'string'},mail:{type: 'string'},isAdmin:{type: 'boolean'}}}
```

```
//          Retrive one User
//-------------------------------
describe("HTTP assertions : Retrive one User", function () {
  it("It should return HTTP_200 : user ", function () {
    var response = chakram.get(url+"/users/3651ac54-d393-495b-b2ae-a26616de3fc4");
    expect(response).to.have.status(200);
    //expect(response).to.have.header("content-type", "application/json");
    expect(response).to.have.schema(expectedUserSchema);
    expect(response).to.have.json(UserData);
    return chakram.wait();
  });
});
```

Et après avoir lancé le test avec mocha, on voit que tout s'est bien passé.

```
HTTP assertions : Retrive one User
  ✓ It should return HTTP_200 : user
```

L’api ne pouvant pas se contenter de ces fonctions primaires, on y a aussi ajouté des fonctions permettant par exemple d’inscrire un utilisateur à un événement.

eventsRoutes.js

```
// Déclaration de sroutes pour /Events
app.route('/')
    .get(getEventsList)
    .post(addEvent)

app.route('/:id')
    .get(getEventById)
    .put(updateEvent)
    .delete(deleteEvent);

app.route('/users/:id')
    .put(addUser)
    .delete(deleteUser)
```

eventsControler.js

```
// pour inscrire un utilisateur a un evenement
exports.addUser = (req, res) => {
    const event = events.getEventById(req.params.id)
    const user = users.getUserById(req.body.userId)
    if (isset(event)){
        if (isset(user)){
            res.status(202).send(events.addUser(req.params.id, req.body.userId))
        } else {
            res.status(404).send('user not found')
        }
    } else {
        res.status(404).send('event not found')
    }
}
```

eventsRepository.js

```
addUser(id, userId){
    const infos = {
        user : userId,
        joinDateTime : new Date().toString()
    }
    const i = this.items.findIndex( predicate: el => el.id === id)
    if (i !== -1) {
        const j = this.items[i].users.findIndex( predicate: el => el.user === userId)
        if (j === -1) {
            this.items[i].users.push(infos)
        }
    }
    return this.getEventById(id)
}
```

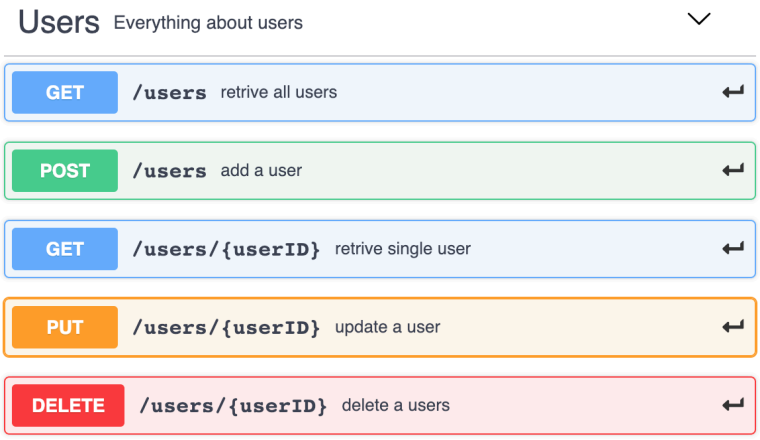
testApi.js

```
var userToAdd = { userId: "b66245cd-c1db-48dc-8c26-1c3ef8349175" }
//          Add one user to an Event
//-------------------------------------
describe("HTTP assertions : Add one user to an Event", function () {
  it("It should return HTTP_202", function () {
    var response = chakram.put(url+"/events/users/3651ac54-d393-495b-b2ae-a26616de3fc4", userToAdd);
    expect(response).to.have.status(202);
    expect(response).to.have.schema(expectedEventSchema);
    //expect(response).to.have.header("content-type", "application/json");
    return chakram.wait();
  });
});
```

Cette fois encore le test passe sans soucis.

```
HTTP assertions : Add one user to an Event
  ✓ It should return HTTP_202
```

Afin d'avoir un code coverage à 100%, nous avons effectué sur chacun des objets cités en début de document au moins les 5 tests suivants :



Mais nous avons aussi testé pour les idées la possibilité qu'il y a d'y mettre un "like" ou un "dislike" et pour les evenement, de s'y inscrire ou de se désinscrire.
Comme montré précédemment avec l'exemple sur l'inscription à un evennement.

Ce qui nous fait un total de 29 tests, tous passants.

```
HTTP assertions : Retrive all Events
  ✓ It should return HTTP_200 : list with events

HTTP assertions : Add one Event
  ✓ It should return HTTP_201 : event

HTTP assertions : Retrive one Event
  ✓ It should return HTTP_200 : event

HTTP assertions : Update one Event
  ✓ It should return HTTP_202 : event

HTTP assertions : Add one user to an Event
  ✓ It should return HTTP_202

HTTP assertions : Delete one user from an Event
  ✓ It should return HTTP_202

HTTP assertions : Delete one Event
  ✓ It should return HTTP_200


29 passing (312ms)
```

Voyez-vous ?