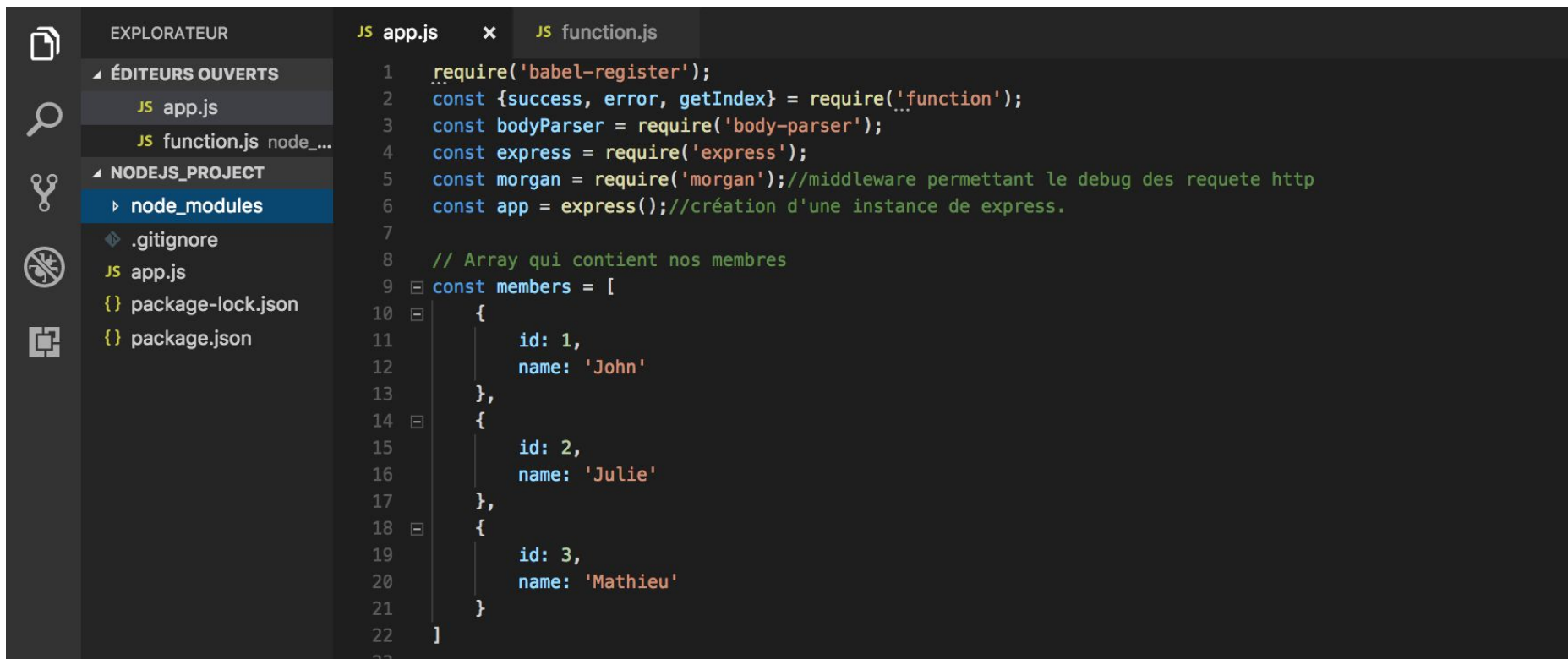


app.js partie 1



The image shows a screenshot of the Visual Studio Code editor interface. On the left, the Explorer sidebar is open, showing the project structure. The 'node_modules' directory is expanded, and the 'app.js' file is selected. The main editor area displays the code for 'app.js'.

```
1  require('babel-register');
2  const {success, error, getIndex} = require('function');
3  const bodyParser = require('body-parser');
4  const express = require('express');
5  const morgan = require('morgan');//middleware permettant le debug des requete http
6  const app = express();//création d'une instance de express.
7
8  // Array qui contient nos membres
9  const members = [
10     {
11         id: 1,
12         name: 'John'
13     },
14     {
15         id: 2,
16         name: 'Julie'
17     },
18     {
19         id: 3,
20         name: 'Mathieu'
21     }
22 ]
```

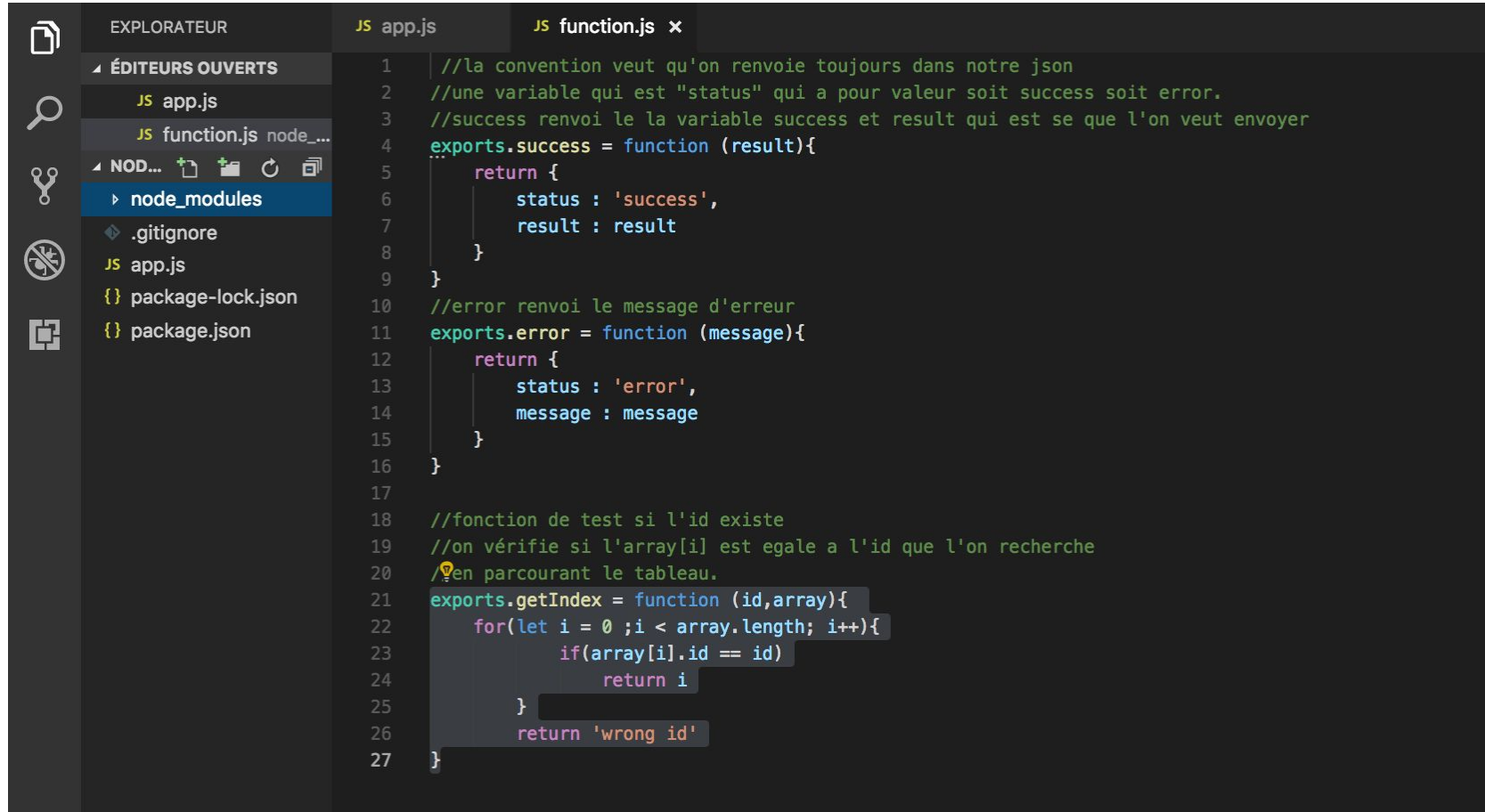
app.js partie 2

```
23
24 //création d'un router MembersRouter
25 let MembersRouter = express.Router();
26
27 //middleware permettant de debug les requete http pendant la phase de développement
28 app.use(morgan('dev'));
29
30
31 app.use(bodyParser.json()); // permet de parser le json
32 app.use(bodyParser.urlencoded({ extended: true })); // pour parser l'application/x-www-form-urlencoded
33
34 //on inclus nos fonction dans notre routeur
35 MembersRouter.route('/:id')
36   .put((req, res) => {
37     //fonction qui permet de faire une modification sur un membre
38     let index = getIndex(req.params.id, members);
39     if(typeof(index) == 'string'){
40       res.json(error(index));
41     }else{
42       let sameName = false;
43       for(let i = 0; i < members.length; i++){ // on verifie que le name n'est pas déjà pris par un autre membre
44         if(req.body.name == members[i].name && req.params.id != members[i].id){ // mais on autorise un membre a se rappeler de la meme façon
45           sameName = true;
46           break;
47         }
48       }
49       if(sameName){
50         res.json(error('name already taken'));
51       }else{
52         members[index].name = req.body.name;
53         res.json(success(true));
54       }
55     }
56   })
57   //fonction qui nous permet de récupérer les membre en fonction de leur id
58   .get((req, res) => {
59     let index = getIndex(req.params.id, members);
60
61     if(typeof(index) == 'string'){
62       res.json(error(index));
63     }else{
64       res.json(success(members[index]));
65     }
66   })
67
68 //]
69 //fonction de suppression de membres
70 .delete((req, res) => {
71   let index = getIndex(req.params.id, members);
72   if(typeof(index) == 'string'){
73     res.json(error(index));
74   }else{
75     members.splice(index, 1) // splice() permet de supprimer un élément d'un array
76     res.json(success(members));
77   }
78 })
```

app.js partie 3

```
79
80 MembersRouter.route('/')
81 //fonction avec req.query qui nous permet de récupérer au choix le max de membres que l'on veut.
82 .get((req, res)=>{
83     if(req.query.max != undefined && req.query.max > 0){
84         res.json(success(members.slice(0, req.query.max)));
85     }else if(req.query.max != undefined){ // gestion de l'erreur
86         res.json(error('Wrong max value'));
87     }else{
88         res.json(success(members)); //requête permettant de récupérer tous les membres
89     }
90 })
91
92 //quand on arrive avec la méthode post sur cet url cela déclenchera la fonction qui permet de rajoute un membre avec sa propriété name
93 .post((req, res)=>{
94     if(req.body.name){
95         let sameName= false;
96         //boucle de test a savoir si le prénom du membres que l'on veut rajouter n'est pas déjà pris
97         for(let i = 0; i < members.length; i++){
98             if(members[i].name == req.body.name){
99                 sameName = true;
100                 break;
101             }
102         }
103         if(sameName){
104             res.json(error('name already taken'));
105         }else{
106             let member = {
107                 id : createID(),
108                 name : req.body.name
109             };
110             members.push(member);
111             res.json(success(member))
112         }
113     }else{
114         res.json(error('no name value'));
115     }
116 })
117
118 // on précise sur quoi agit notre routeur MembersRouter
119 app.use('/api/v1/members', MembersRouter);
120
121 //on donne un port a notre application avec une fonction de call back pour confirmer la conection
122 app.listen(8080,()=> console.log('Started on port 8080'));
123
124
125 function createID(){
126     return members[members.length-1].id + 1;
127 }
128
```

function.js



EXPLORATEUR

ÉDITEURS OUVERTS

- JS app.js
- JS function.js node_...

NOD...

- node_modules
- .gitignore
- JS app.js
- package-lock.json
- package.json

```
1 //la convention veut qu'on renvoie toujours dans notre json
2 //une variable qui est "status" qui a pour valeur soit success soit error.
3 //success renvoi le la variable success et result qui est se que l'on veut envoyer
4 exports.success = function (result){
5     return {
6         status : 'success',
7         result : result
8     }
9 }
10 //error renvoi le message d'erreur
11 exports.error = function (message){
12     return {
13         status : 'error',
14         message : message
15     }
16 }
17
18 //fonction de test si l'id existe
19 //on vérifie si l'array[i] est egale a l'id que l'on recherche
20 //en parcourant le tableau.
21 exports.getIndex = function (id,array){
22     for(let i = 0 ;i < array.length; i++){
23         if(array[i].id == id)
24             return i
25     }
26     return 'wrong id'
27 }
```