

# NODE.JS & FRAMEWORK EXPRESS

# Plan

2

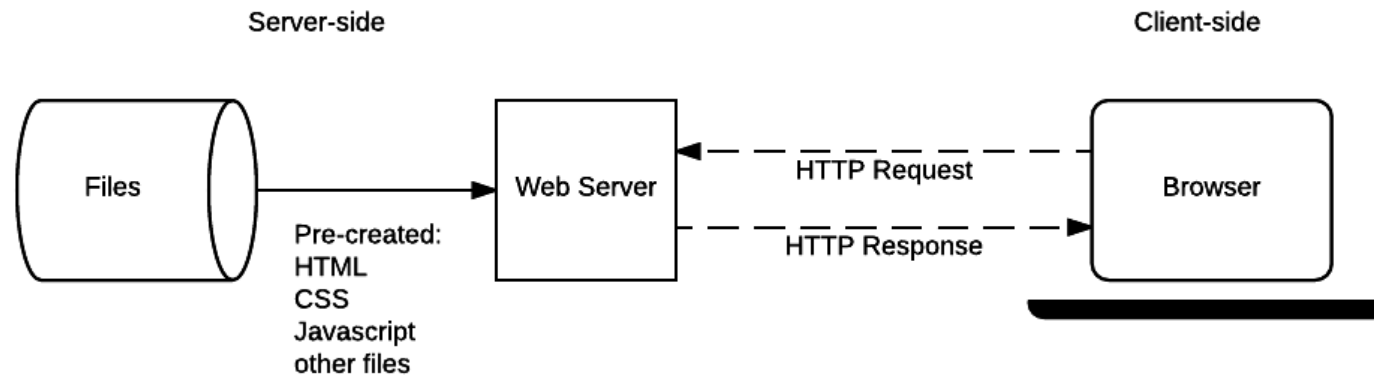
- Node.js : les bases
- Node.js : base de données
- Express
- MVC : un exemple d'une application

3

# Node.js : les bases

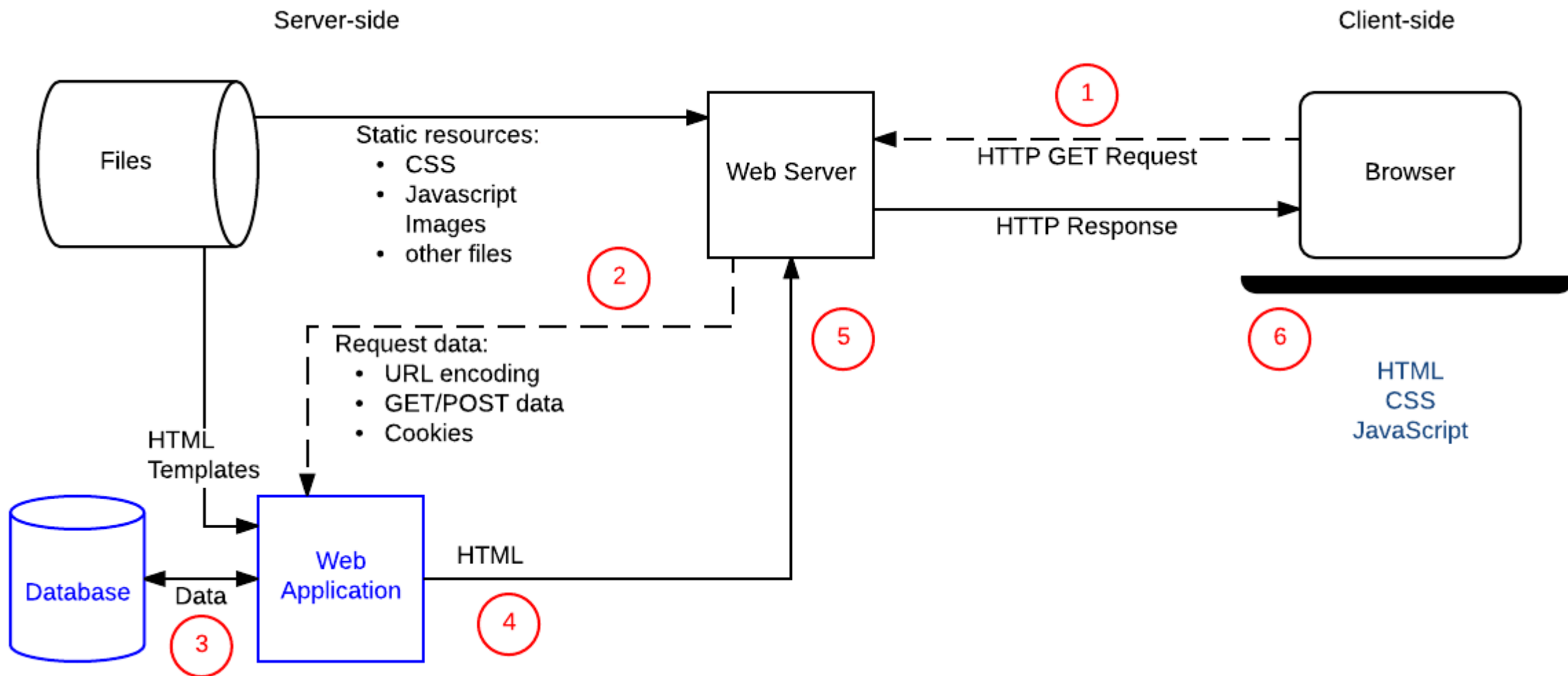
# Site Statique

4



# Site dynamique

5



# Node.js : Introduction

6

- ❑ Node est un serveur web open source, gratuit et multiplateforme
- ❑ Node Utilise JavaScript côté serveur pour :
  - ▣ Traiter les requêtes http
  - ▣ Avoir accès aux fonctionnalités de l'OS du serveur via librairie
- ❑ Les APIs web coté client du navigateur ne sont pas accessibles/utilisables
- ❑ Node.js permet la programmation asynchrone

# Node.js : avantages

7

- ❑ Bonnes performances => applications temps-réel
- ❑ Langage Javascript côté client et serveur => gain de temps
- ❑ Plusieurs librairies js faciles à intégrer (npm)
- ❑ Portable et multi-plateforme
- ❑ Une grande communauté dynamique de développeur

# Node.js : tester

8

- Installer node.js
  - ▣ Selon votre OS suivez la procédure d'installation sur le lien suivant :
  - ▣ <https://nodejs.org/fr/download>
  
- Lancer un programme sur node.js : > *node test.js*



# Exemple d'un programme exécuté sur node.js

9

```
// charger le module HTTP
const http = require("http");

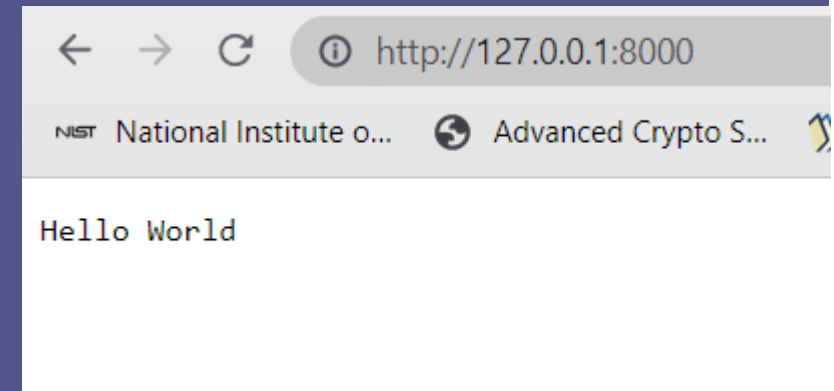
const hostname = "127.0.0.1";
const port = 8000;

// Créer un serveur HTTP
const server = http.createServer(function (req, res) {
  // Ajouter l'entête de la réponse avec HTTP status et type de contenu
  res.writeHead(200, { "Content-Type": "text/plain" });

  // Envoyer la réponse body "Hello World"
  res.end("Hello World\n");
});

// Affiche un log une fois le serveur commence l'écoute des requête
server.listen(port, hostname, function () {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

```
PS F:\Mes document AT16\Contenu 2023\TDs\projet-web> node .\test.js
Server running at http://127.0.0.1:8000/
```



# Modules

10

- ❑ Inclure un module :
  - ▣ `var un_module=require("nom_module");`
- ❑ Modules intégrés (built-in)

<a href="#"><u>assert</u></a>	Provides a set of assertion tests
<a href="#"><u>buffer</u></a>	To handle binary data
<a href="#"><u>child_process</u></a>	To run a child process
<a href="#"><u>cluster</u></a>	To split a single Node process into multiple processes
<a href="#"><u>crypto</u></a>	To handle OpenSSL cryptographic functions
<a href="#"><u>dgram</u></a>	Provides implementation of UDP datagram sockets
<a href="#"><u>dns</u></a>	To do DNS lookups and name resolution functions
<a href="#"><u>domain</u></a>	Deprecated. To handle unhandled errors
<a href="#"><u>events</u></a>	To handle events
<a href="#"><u>fs</u></a>	To handle the file system
<a href="#"><u>http</u></a>	To make Node.js act as an HTTP server
<a href="#"><u>https</u></a>	To make Node.js act as an HTTPS server.
<a href="#"><u>net</u></a>	To create servers and clients
<a href="#"><u>os</u></a>	Provides information about the operation system

<a href="#"><u>path</u></a>	To handle file paths
<a href="#"><u>punycode</u></a>	Deprecated. A character encoding scheme
<a href="#"><u>querystring</u></a>	To handle URL query strings
<a href="#"><u>readline</u></a>	To handle readable streams one line at the time
<a href="#"><u>stream</u></a>	To handle streaming data
<a href="#"><u>string_decoder</u></a>	To decode buffer objects into strings
<a href="#"><u>timers</u></a>	To execute a function after a given number of milliseconds
<a href="#"><u>tls</u></a>	To implement TLS and SSL protocols
<a href="#"><u>tty</u></a>	Provides classes used by a text terminal
<a href="#"><u>url</u></a>	To parse URL strings
<a href="#"><u>util</u></a>	To access utility functions
<a href="#"><u>v8</u></a>	To access information about V8 (the JavaScript engine)
<a href="#"><u>vm</u></a>	To compile JavaScript code in a virtual machine
<a href="#"><u>zlib</u></a>	To compress or decompress files

# Création d'un module

11

- Créer un fichier “mon\_module.js”

```
exports.maDateTime = function () {  
    return Date();  
};
```

- ▣ le mot-clé « **exports** » rend les propriétés et les méthodes disponibles en dehors du fichier de module

- Importer le module

- ▣ `var dt = require('./mon_module');`
- ▣ `Console.log(dt.maDateTime());`

# Module.exports

12

```
//db.js
var mysql = require("mysql");

var pool = mysql.createPool({
  connectionLimit : 10,
  host: "tuxa.sme.utc", //ou localhost
  user: "ai16p0*0",
  password: "*****",
  database: "ai16p0*0"
});
Module.exports=pool;
```

```
//user.js
var db = require('./db.js');
module.exports = {
  read: function (email, callback) {
    .....
  },
  readall: function (callback) {
    .....
  }
}
```

13

# Node.js : base de données

# Accès à une base de données

14

- ❑ Installer mysql Driver
  - ▣ npm install mysql
  
- ❑ Importer le module
  - ▣ `var mysql = require('mysql');`
  
- ❑ Créer une connexion
  - ▣ `var con = mysql.createConnection({`
    - `host: "tuxa.sme.utc", //ou localhost`
    - `user: "ai16p000",`
    - `password: "*****"`  - ▣ `});`

- Lancer la connexion et faire une requête

```
.....  
var sql="votre sql ici";  
con.connect(function(err) {  
  if (err) throw err;  
  console.log("Connected!");  
  con.query(sql, function (err, result) {  
    if (err) throw err;  
    console.log("Result: " + result);  
  });  
});
```

# Requêtes SQL

16

- Requêtes SQL :
  - ▣ Création d'une base de donnée
    - var sql= "CREATE DATABASE mydb"
  - ▣ Création d'une table
    - var sql= "CREATE TABLE customers (name VARCHAR(255), address VARCHAR(255))"
  - ▣ Supprimer une table
    - var sql= "DROP TABLE customers"
  - ▣ Insérer une ligne dans une table
    - var sql= "INSERT INTO customers (name, address) VALUES ('Company Inc', 'Highway 37')"
  - ▣ Supprimer une ou plusieurs lignes
    - var sql= "DELETE FROM customers WHERE address = 'Mountain 21'"
  - ▣ Modifier une ligne ou plusieurs
    - var sql= "UPDATE customers SET address = 'Canyon 123' WHERE address = 'Valley 345'"



# L'objet « result »

17

- Cet objet est retourné lors de l'exécution de la requête

- ```
{  
  fieldCount: 0,  
  affectedRows: 14,  
  insertId: 0,  
  serverStatus: 2,  
  warningCount: 0,  
  message: '\Records:14 Duplicated: 0 Warnings: 0',  
  protocol41: true,  
  changedRows: 0  
}
```

- Pour afficher un champ :

- ▣ `console.log(result.affectedRows)`
- ▣ `console.log(result.insertId) // récupérer l'id généré automatiquement`

```
.....  
var sql="votre sql ici";  
con.connect(function(err) {  
  if (err) throw err;  
  console.log("Connected!");  
  con.query(sql, function (err, result) {  
    if (err) throw err;  
    console.log("Result: " + result);  
  });  
});
```

# La requête « Select from »

18

- le paramètre « result » pour récupérer les résultats

```
con.connect(function(err) {  
  if (err) throw err;  
  con.query("SELECT * FROM customers", function (err, result, fields) {  
    if (err) throw err;  
    console.log(result);  
  });  
});
```

▣ *result[0].name; result[0].address.*

```
[  
  { name: 'John', address: 'Highway 71'},  
  { name: 'Peter', address: 'Lowstreet 4'},  
]
```

- le paramètre « field » pour récupérer des informations (tableau) sur chaque colonne

# La requête « select from where »

19

```
con.connect(function(err) {  
  if (err) throw err;  
  con.query("SELECT * FROM customers WHERE address = 'Park Lane 38'", function (err, result) {  
    if (err) throw err;  
    console.log(result);  
  });  
});
```

## □ Utiliser Like

▣ « % » : zéro ou plusieurs caractères

```
con.connect(function(err) {  
  if (err) throw err;  
  con.query("SELECT * FROM customers WHERE address LIKE 'S'", function (err, result) {  
    if (err) throw err;  
    console.log(result);  
  });  
});
```

# La requête « select from where »

## Eviter les injections SQL

20

- ❑ `mysql.escape()`  
var adr = 'Mountain 21';  
var sql = 'SELECT \* FROM customers WHERE address = ' + mysql.escape(adr);
  
- ❑ Espace réservé « ? »  
var name = 'Amy';  
var adr = 'Mountain 21';  
var sql = 'SELECT \* FROM customers WHERE name = ? OR address = ?';  
con.query(sql, [name, adr], function (err, result) {  
 if (err) throw err;  
 console.log(result);  
});

# Mutualiser les connexions (pooling)

21

```
var mysql = require("mysql");

var pool = mysql.createPool({
  connectionLimit : 10,
  host: "tuxa.sme.utc", //ou localhost
  user: "ai16p0*0",
  password: "*****",
  database: "ai16p0*0"
});

pool.query("select * from Utilisateur", function (err, results) {
  if (err) throw err;
  console.log(results[0].nom);
});
```

22

Express

# Express (node js/ javascript)

23

- ❑ Réaliser une application web nécessitent généralement de traiter toutes les requêtes http, générer les réponses, retourner des pages html, ....
- ❑ Utiliser uniquement node.js pour le faire impliquera coder tout vous-même
- ❑ Framework express permet de réutiliser des composants afin de développer rapidement une application web

- Un framewok très populaire qui permet de :
  - ▣ Écrire des contrôleurs pour les requêtes HTTP différents verbes (get, post, delete,...) sur différents chemins d'URL (routes).
  - ▣ Intégrer un moteur de rendu « vue » pour générer des des réponses en intégrant des données dans des templates
  - ▣ Définissez les paramètres d'application Web courants, tels que le port à utiliser pour la connexion et l'emplacement des modèles utilisés pour le rendu de la réponse.
  - ▣ Ajoutez un "middleware" de traitement des requêtes supplémentaire à tout moment dans le pipeline de traitement des requêtes.



# Middlewares express

25

- Express est minimaliste, c-à-d il a des fonctionnalités de base mais extensible via des middleware pour gérer :
  - ▣ Les cookies, les sessions, user logins, les paramètres d'URL, les données de POST, les entêtes de security,...
  - ▣ Liste des middleware express :  
<https://expressjs.com/en/resources/middleware.html>

# Première application express

26

```
const express = require("express");  
const app = express();  
const port = 3000;
```

```
app.get("/", function (req, res) {  
  res.send("Hello World!");  
});
```

```
app.listen(port, function () {  
  console.log(`Example app listening on port ${port}!`);  
});
```

Importer le module express

Get spécifie une fonction de rappel qui sera invoquée chaque fois qu'il y a une requête HTTP GET avec un chemin ('/')

- `app.get("/", function (req, res) {  
 res.send("Hello World!");  
});`
- Il existe des méthodes pour tous les verbes http : get, post, put, delete, options, .....
- `app.all()` : elle est appelée à n'importe quelle méthode http

```
app.all("/secret", function (req, res, next) {  
    console.log("Accessing the secret section...");  
    next(); // pass control to the next handler  
});
```

# express.Router()

28

- Regrouper la gestion de plusieurs routes d'une partie d'un site avec le même préfix (« /wiki ») dans le même fichier :

```
const express = require("express");
const router = express.Router();

// Home page route
router.get("/", function (req, res) {
  res.send("Wiki home page");
});

// About page route
router.get("/about", function (req, res) {
  res.send("About this wiki");
});
```

# express.Router()

29

## □ Pour les chemins

```
const wiki = require("./wiki.js");  
// ...  
app.use("/wiki", wiki);
```

# Ajouter des fichier statiques

30

- Utiliser le middleware « `express.static` »
  - ▣ `app.use(express.static("public"));`
  - ▣ *retourner les fichiers statiques stockés dans le dossier “public”*
  
- *Pour accéder :*
  - ▣ <http://localhost:3000/index.html>
  - ▣ <http://localhost:3000/images/logo.jpg>
  - ▣ <http://localhost:3000/style.css>
  - ▣ *Etc.*

# Ajouter des fichier statiques

31

- Ajouter plusieurs dossiers :
  - ▣ `app.use(express.static("public"));`
  - ▣ `app.use(express.static("media"))`
  - ▣ Les fichiers seront servis selon l'ordre des déclarations
  
- Ajouter un prefix
  - ▣ `app.use("/media", express.static("public"));`
  - ▣ <http://localhost:3000/media/images/logo.jpg>

# Création du rendu (vue)

32

## □ Configuration

```
const express = require("express");
const path = require("path");
const app = express();

// Set directory to contain the templates ('views')
app.set("views", path.join(__dirname, "views"));

// Set view engine to use, in this case 'some_template_engine_name'
app.set("view engine", "some_template_engine_name");
```

## □ Fabriquer le rendu avec res.render ("page",data\_js)

```
app.get("/", function (req, res) {
  res.render("index", { title: "About dogs", message: "Dogs rock!" });
});
```



# Création du rendu (vue)

## Utilisation de EJS

33

### □ Étape d'initialisation

```
// view engine setup
app.set('views', path.join(__dirname,
'views'));
app.set('view engine', 'ejs');
```

### □ Retourner une vue (suite à une requête)

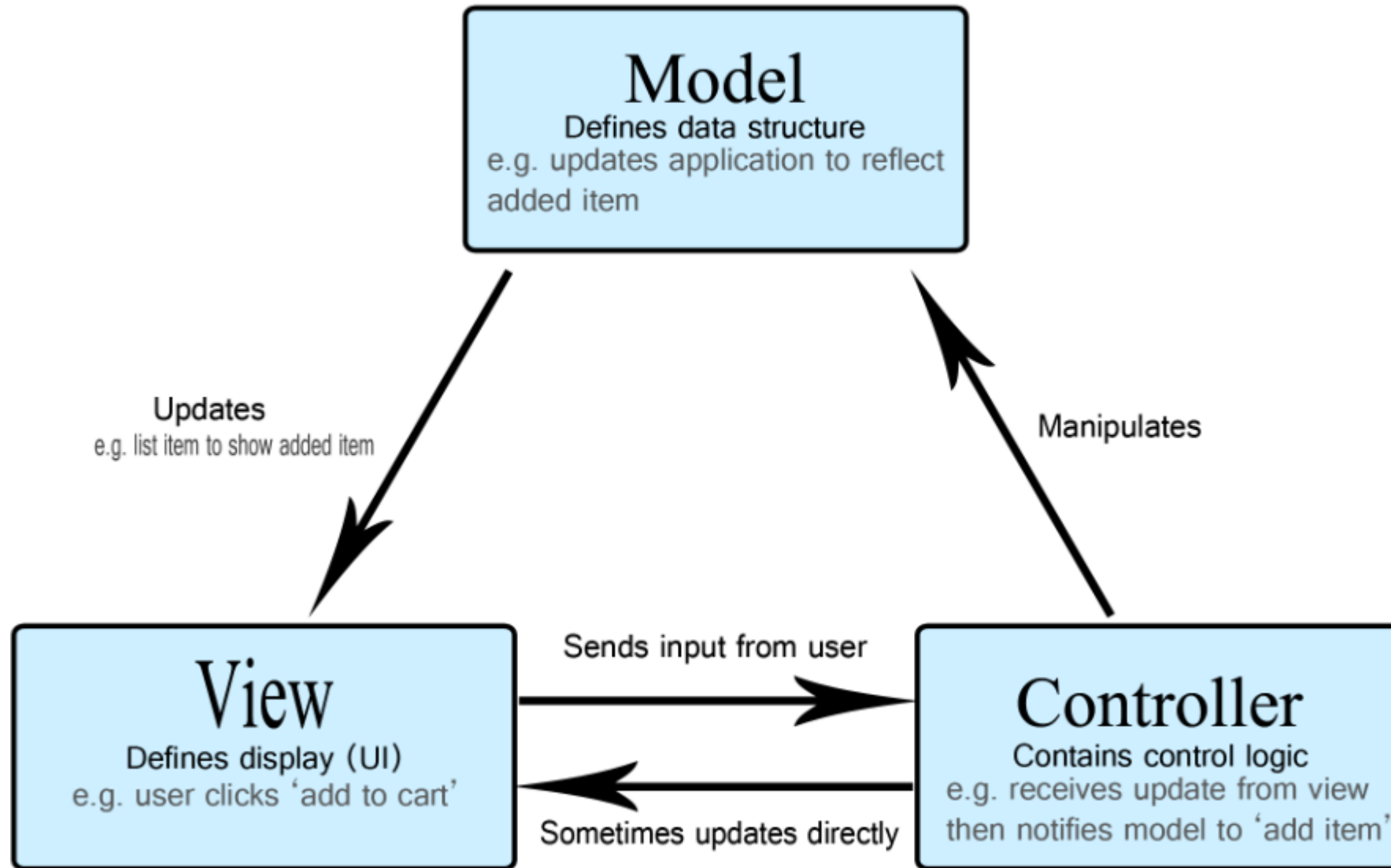
```
res.render('usersList', { title: 'List des
utilisateurs', users: result });
```

```
<h1>
  <%= title %>
</h1>
<table border>
  <thead>
    <tr>
      <th>Firstname</th>
    </tr>
  </thead>
  <tbody>
    <% users.forEach((user)=> { %>
      <tr>
        <td>
          <%= user.prenom %>
        </td>
      </tr>
    <% }) %>
  </tbody>
</table>
```

# MVC : un exemple d'une application

# Architecture MVC

35



# express-generator

36

- npm install -g express-generator
  - ▣ express -h
- Créer un projet express (choisir comme moteur de template ejs):
  - ▣ express --view=ejs myapp

```
create : myapp\  
create : myapp\public\  
create : myapp\public\javascripts\  
create : myapp\public\images\  
create : myapp\public\stylesheets\  
create : myapp\public\stylesheets\style.css  
create : myapp\routes\  
create : myapp\routes\index.js  
create : myapp\routes\users.js  
create : myapp\views\  
create : myapp\views\error.ejs  
create : myapp\views\index.ejs  
create : myapp\app.js  
create : myapp\package.json  
create : myapp\bin\  
create : myapp\bin\www
```

# Tester l'application

37

- Les commandes pour tester cette application :
  - ▣ Aller dans le dossier de l'application :
    - > cd myapp
  - ▣ Installer les dépendances :
    - > npm install
  - ▣ Lancer l'application (windows):
    - > SET DEBUG=myapp:\* & npm start

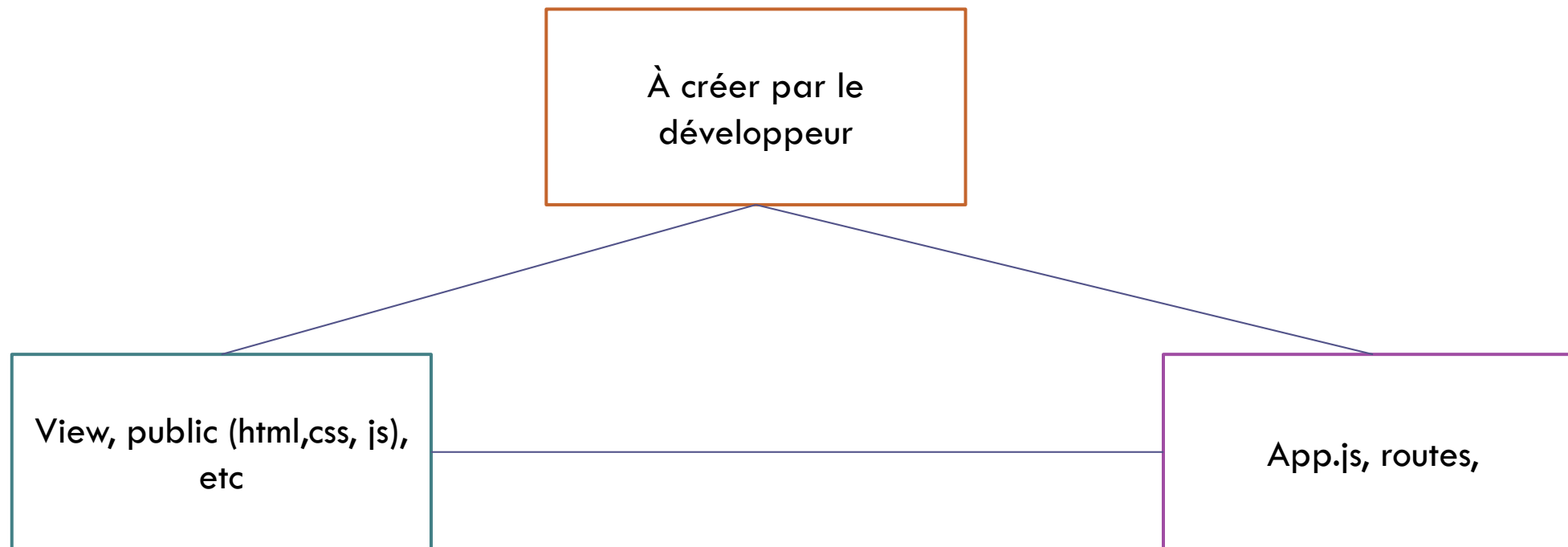
# Structure d'un projet express

38

Dossier/fichier	Description
<b>public</b>	Les ressources statiques (html, css, javascript, images)
<b>routes</b>	Contient les scripts js qui traites les requêtes http
<b>routes\index.js</b>	Traiter les requêtes de l'url home (/)
<b>routes\users.js</b>	Traiter les requêtes de l'url (/users)
<b>view\</b>	Contients les templates ejs
<b>view\error.ejs</b>	Template de la page d'erreur
<b>view\index.ejs</b>	Template de la page accueil
<b>app.js</b>	Le point d'entrée d'une application express => choisir le moteur de template, déclarer les routes, etc.
<b>package.js</b>	Les dépendances
<b>bin\www</b>	Configurer et lancer le serveur http

# Structure d'un projet express et MVC

39



# Créer votre modèle

40

- ❑ Créer un sous-dossier model dans le dossier de votre application
- ❑ Créer un fichier de config de la base de données
- ❑ Pour chaque entité créer un fichier js pour chaque entité
- ❑ Exporter tous les fichiers
- ❑ Utiliser votre model dans les contrôleurs (routes)



# Fichier model/db.js

41

```
//db.js
var mysql = require("mysql");

var pool = mysql.createPool({
  connectionLimit : 10,
  host: "tuxa.sme.utc", //ou localhost
  user: "ai16p0*0",
  password: "*****",
  database: "ai16p0*0"
});
Module.exports=pool;
```

# Le fichier model/user.js

42

```
var db = require('./db.js');
module.exports = {
  read: function (email, callback) {
    db.query("select * from Utilisateur
where email= ?",email, function (err, results)
{
    if (err) throw err;
    callback(results);
  });
},
  readall: function (callback) {
    db.query("select * from Utilisateur",
function (err, results) {
    if (err) throw err;
    callback(results);
  });
},
}
```

## □ La suite

```
  areValid: function (email, password,
callback) {
    sql = "SELECT pwd FROM USERS WHERE
email = ?";
    rows = db.query(sql, email, function
(err, results) {
    if (err) throw err;
    if (rows.length == 1 && rows[0].pwd
=== password) {
      callback(true)
    } else {
      callback(false);
    }
  });
},
}
```

# Utiliser mon model

## routes/users.js

43

```
var express = require('express');
var router = express.Router();
var userModel = require('../model/user.js')

/* GET users listing. */
router.get('/', function (req, res, next) {

    res.send('respond with a resource');

});
router.get('/userslist', function (req, res, next) {

    result=userModel.readall(function(result){

        res.render('usersList', { title: 'List des utilisateurs', users: result });

    });
});

module.exports = router;
```

# App.js

44

```
...  
...  
var indexRouter = require('./routes/index');  
var usersRouter = require('./routes/users');  
  
var app = express();  
  
// view engine setup  
app.set('views', path.join(__dirname, 'views'));  
app.set('view engine', 'ejs');  
  
app.use(logger('dev'));  
app.use(express.json());  
app.use(express.urlencoded({ extended: false }));  
app.use(cookieParser());  
app.use(express.static(path.join(__dirname, 'public')));  
  
app.use('/', indexRouter);  
app.use('/users', usersRouter);  
...
```

```
...  
...  
...  
/**  
 * Create HTTP server.  
 */  
  
var server = http.createServer(app);  
  
/**  
 * Listen on provided port, on all network  
 interfaces.  
 */  
  
server.listen(port);  
server.on('error', onError);  
server.on('listening', onListening);  
  
...  
...  
...
```

# Package.json

46

```
{  
  "name": "myapp",  
  "version": "0.0.0",  
  "private": true,  
  "scripts": {  
    "start": "node ./bin/www"  
  },  
  "dependencies": {  
    "cookie-parser": "~1.4.4",  
    "debug": "~2.6.9",  
    "ejs": "^3.1.9",  
    "express": "^4.18.2",  
    "http-errors": "~1.6.3",  
    "morgan": "~1.9.1"  
  }  
}
```

# Références

47

- ❑ [https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express\\_Nodejs](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs)
- ❑ <https://github.com/mysqljs/mysql#pooling-connections>
- ❑ <https://ejs.co/>
- ❑ <https://www.w3schools.com/>