NODE.JS & FRAMEWORK EXPRESS

SR10

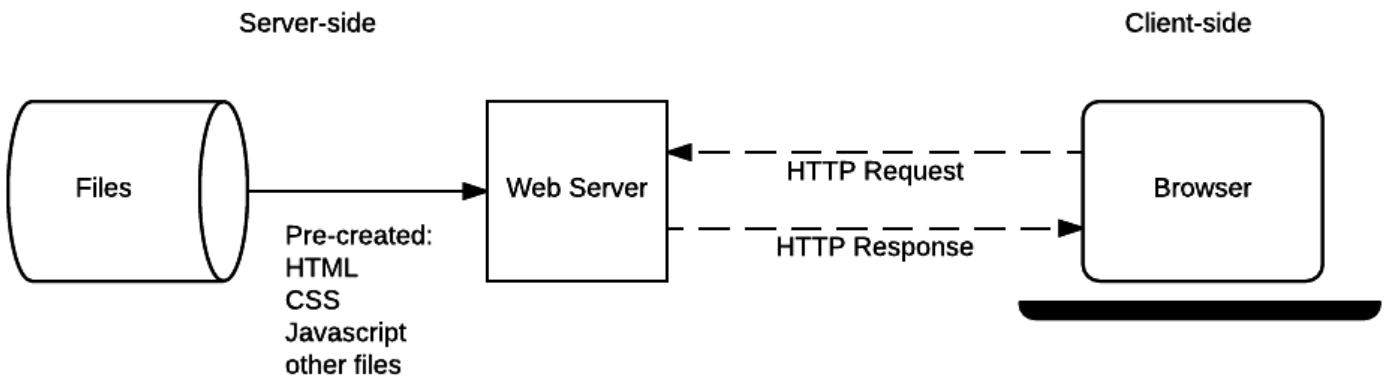
Ahmed.lounis@hds.utc.fr

Plan

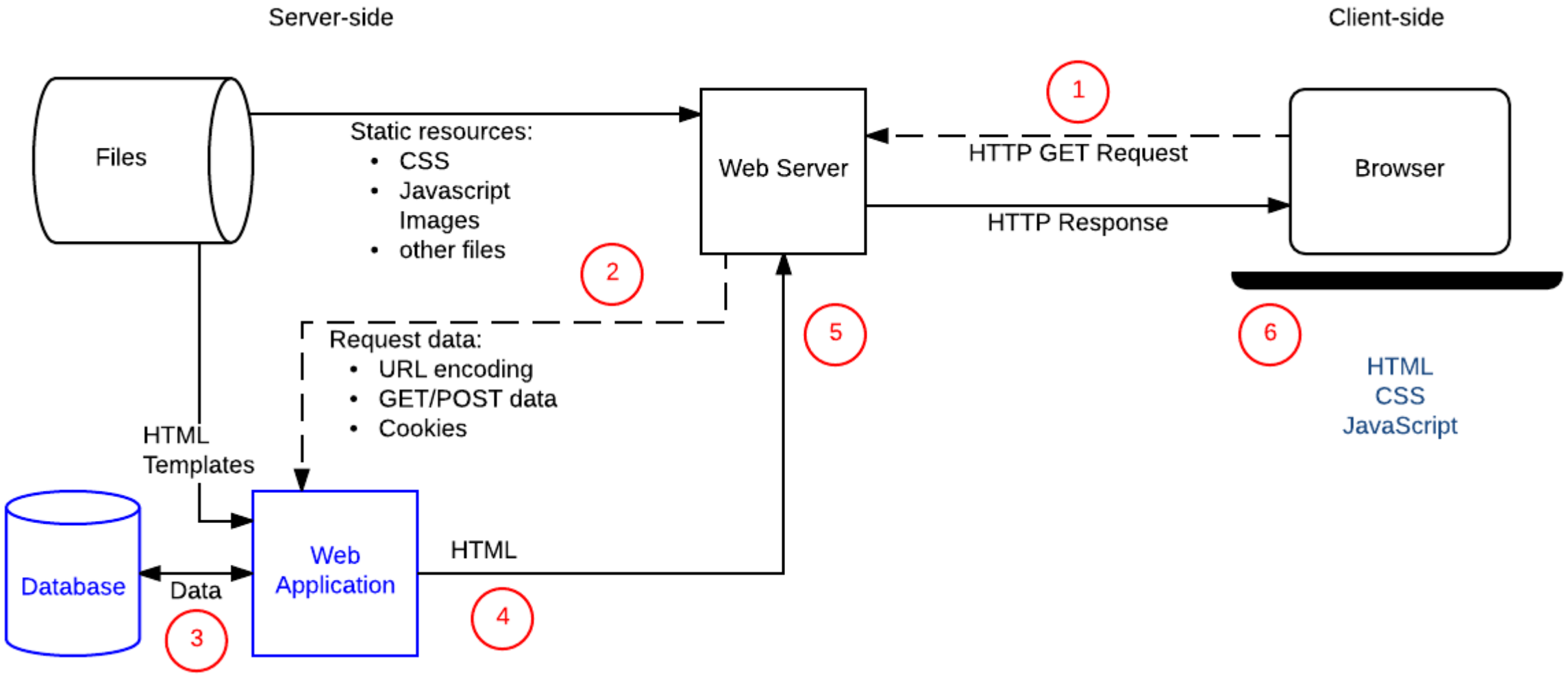
- Node.js : les bases
- Node.js : base de données
- Express
- MVC : un exemple d’une application

Node.js : les bases

Site Statique



Site dynamique



Node.js : Introduction

- 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

- 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

- 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

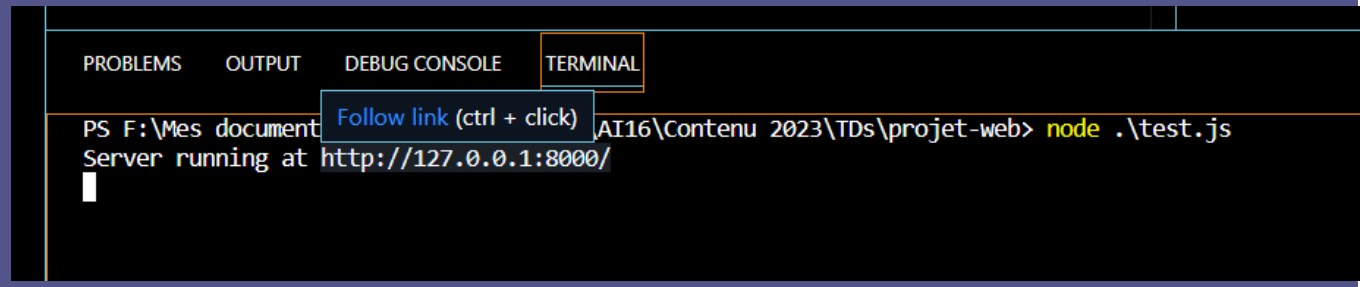

```
// charger le module HTTP
const http = require("http");

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");
});

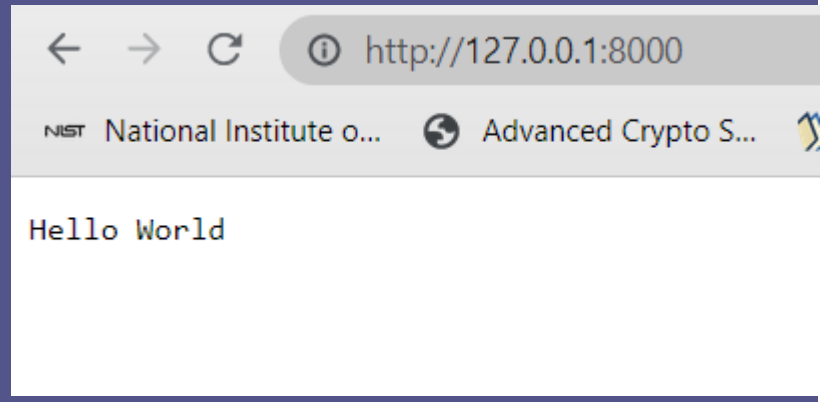
// Affiche un log une fois le serveur commence l'écoute des requête
server.listen(port, hostname, function () {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```



requête / réponse

fonction de rappel (callback) exécutée à chaque requête reçue par le serveur

définit l'entête de la réponse :
- statut : HTTP 200 (OK)
- type de contenu : texte brut



fonction de rappel exécutée une fois que le serveur commence à écouter les requêtes

Modules

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

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

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

Création d'un module

- 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


```
//db.js
var mysql = require("mysql");
```
import the mysql module (allows to interact with a MySQL database)
```
var pool = mysql.createPool({
    connectionLimit : 10,
    host: "tuxa.sme.utc", //ou localhost
    user: "ai16p0*0",
    password: "******",
    database: "ai16p0*0"
});
Module.exports=pool;
```
export the pool object (allows other modules to use it)

```
//user.js
var db = require('./db.js');
```
import the pool object from db.js
```
module.exports = {
```
initialize an object that hold the functions we want to export
```
    read: function (email, callback) {
…….  * cf code ci-dessous
    },
    readall: function (callback) {
…….
  }
}
```

```
read: function(email, callback) { // retrieve a user's information based on their email
  var sql = 'SELECT * FROM users WHERE email = ?';
  db.query(sql, [email], function(error, results, fields) {
    if (error) {
      return callback(error);
    }
    callback(null, results[0]);
  });
},
```

Node.js : base de données

Accès à une base de données

- 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: "********"
  - });

Connexion à une base de données

- 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

- 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 »

- 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
  }
  ```

```
…..
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);
  });
});
```

- Pour afficher un champ :
  - *console.log(result.affectedRows)*
  - *console.log(result. insertId) // récupérer l'id généré automatiquement*
```

La requête « Select from »

- 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);
  });
});
```

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

]
```

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

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

La requête « select from where »


```
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

- 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)


```
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);
});
```

pool de connexions à la base de données MySQL avec 10 connexions simultanées max

Express

Express (node js/ javascript)

- 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

Express

- 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

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

Première application express


```
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

app : instance of the Express application and can be used to define routes and middleware

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

defines a route handler for HTTP GET requests to the root URL ("/")

Start the server and listen on the defined port

set up a basic Express.js server that listens on port 3000 and respond with "Hello World!" when the root URL ("/") is accessed

Création des gestionnaire des routes

- app.get("/", function (req, res) {

  res.send("Hello World!"); envoi de réponse au client

  });
- Il exite des méthodes pour tous les verbes http : get, post, put, delete, options, …..
- app.all() : elle est appelée à n'importe méthode http

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

express.Router()

- 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()

- Pour les chemins

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

Ajouter des fichier statiques

- 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

- 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)

- 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!" });
});
```

fichier EJS passage des variables title et message

Création du rendu (vue)
Utilisation de EJS

- 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)

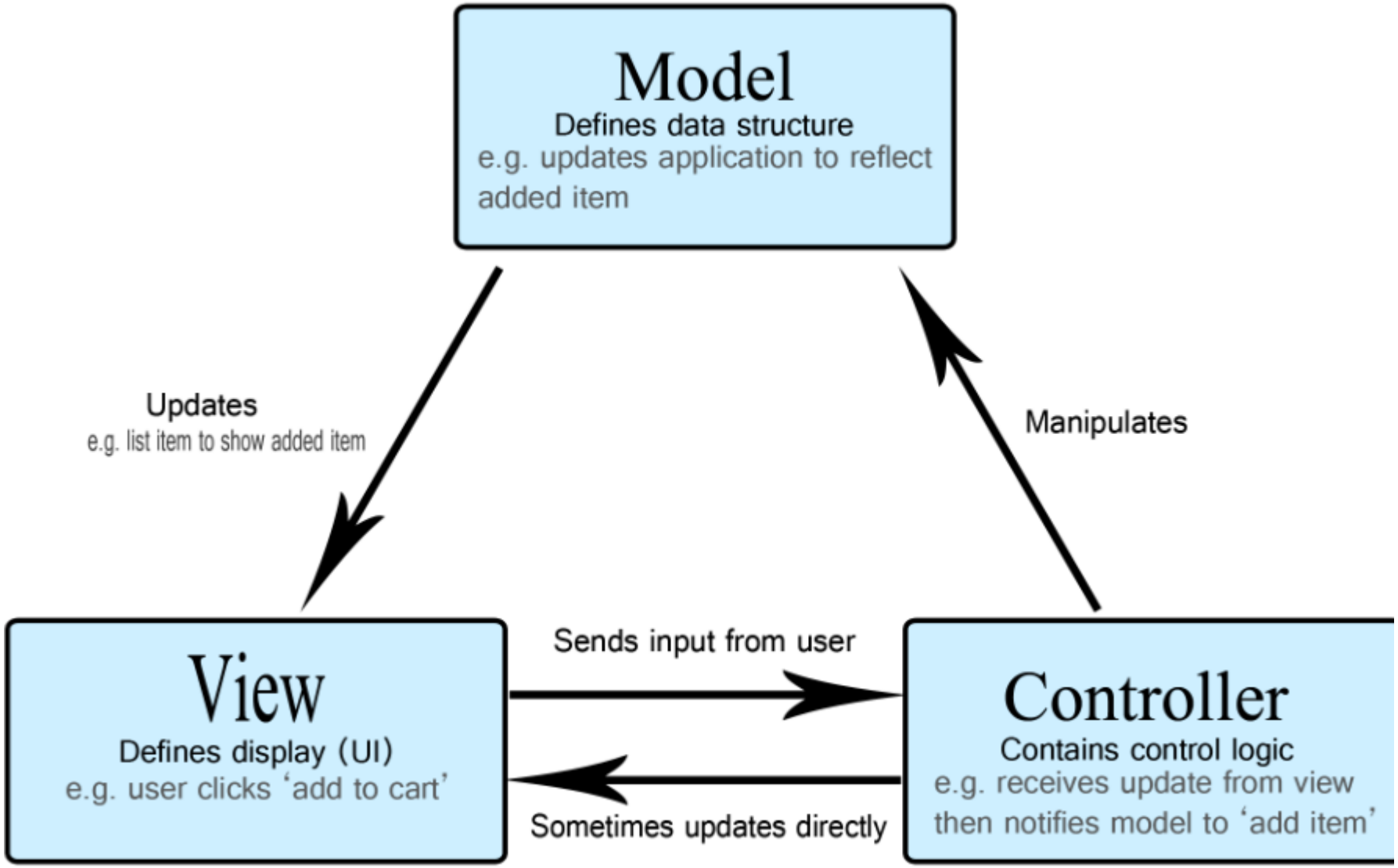
```
res.render('usersList', { title: 'List des
utilisateurs', users: result });
```

```
<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



express-generator

- 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
```

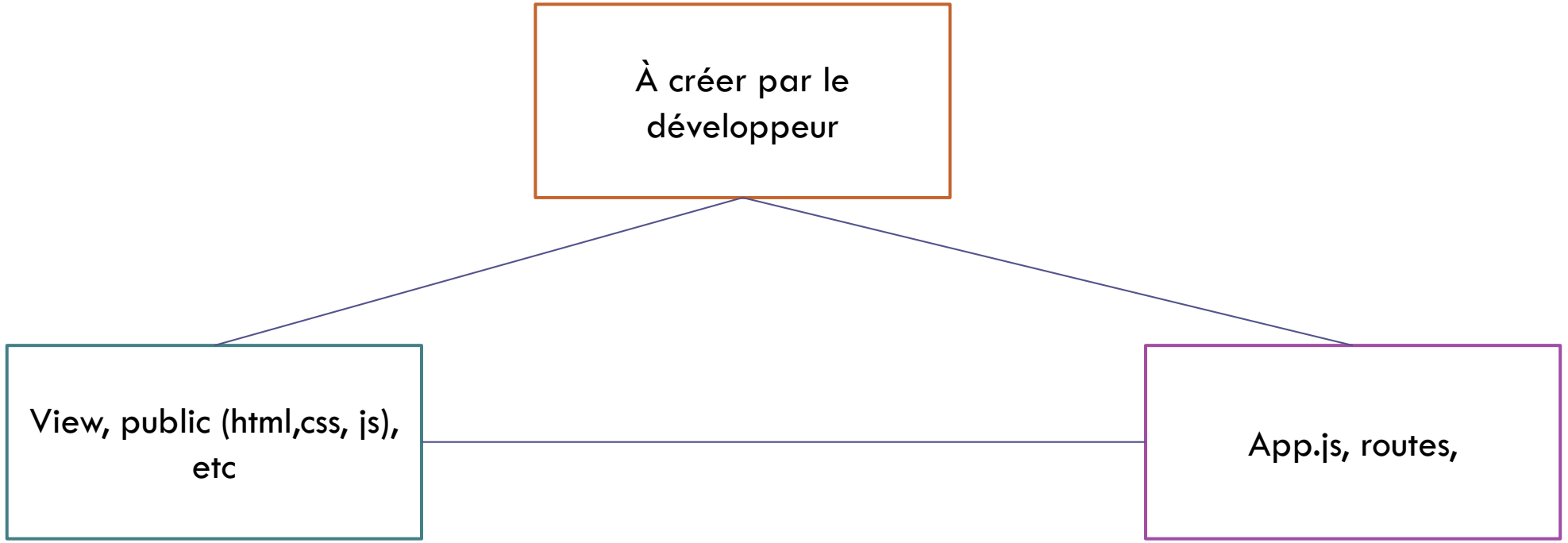
Tester l'application

- 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

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

Structure d'un projet express et MVC



Créer votre modèle

/models: fichiers relatifs aux modèles de données
/controllers: fichiers contrôleurs = logique de gestion des requêtes
/routes: fichiers de définition des routes où les chemins et leurs handlers sont définis

fichiers relatifs aux modèles de l'application

/model
- Créer un sous-dossier model dans le dossier de votre application
- Créer un fichier de config de la base de données — model/db.js : connexion à BdD
- Pour chaque entité (= table) créer un fichier js pour chaque entité — opérations CRUD : Create, Read, Update, Delete
- Exporter tous les fichiers — pour pouvoir être utilisé dans les contrôleurs (module.exports = nom_fichier)

/routes
- Utiliser votre model dans les contrôleurs (routes) — importer les fichiers de modèle dans les fichiers de routes pour effectuer des opérations sur la base de données en réponse aux requêtes HTTP

```
// controllers/userController.js
const express = require("express");
const router = express.Router();
const User = require('../models/User');

// Route pour obtenir tous les utilisateurs
router.get('/users', function(req, res) {
  User.getAll(function(err, users) {
    if (err) return res.status(500).send(err);
    res.json(users);
  });
});
```

```
//suite code
// Route pour obtenir un utilisateur par email
router.get('/user/:email', function(req, res) {
  const email = req.params.email;
  User.getByEmail(email, function(err, user) {
    if (err) return res.status(500).send(err);
    res.json(user);
  });
});

module.exports = router;
```

Fichier model/db.js


```
//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


```
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

fonction de rappel : effectuée une fois que la fonction qui la prend en argument est effectuée


```
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;
```

exécutée une fois que route est accédée

exécutée une fois que les données utilisateur ont été lues

App.js

```
…
…
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);
…
```

./bin/www


```
...
...
...
/**
 * 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


```
{
  "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

- 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/