Application Web

Interaction avec une base de données PostgreSQL

1

Objectifs d'apprentissage

- Comprendre comment organiser le code d'une application Web
- Se connecter à une BD PostgreSQL à partir d'un serveur NodeJS
- Appliquer les concepts appris avec la BD Hotel

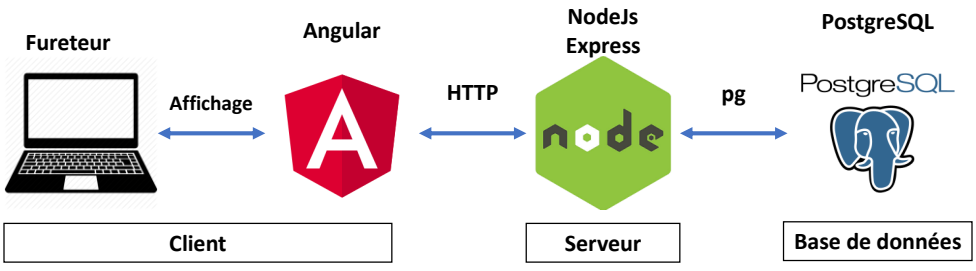
2

Logiciels nécessaires

- NodeJS
  - https://nodejs.org/fr/download/
- PostgreSQL
  - https://www.postgresql.org/download/
- PgAdmin (pas obligatoire)
  - https://www.pgadmin.org/download/

3

Architecture générale



4

5

CLIENT

Angular 6+

Nikolay Radoev

5

Angular

- Cadriciel permettant le développement Web frontal (Front-end) d'une application
- Crée par Google et basé sur TypeScript, un superset d'ECMAScript6
- Vise à découpler la logique d'affichage et la logique de l'application
- **NB:** Même si votre ordinateur agit comme un serveur pour votre projet Angular, aucune fonctionnalité **serveur** ne doit se retrouver du côté Angular

6

Angular - Modules

- Angular est composé de plusieurs **NgModules** qui offrent un contexte de compilation pour les différents **Components** et **Services** d'un projet.
- Un contenant pour le code dédié à une tâche spécifique
- Documentation : **https://angular.io/guide/architecture-modules**

```
import { NgModule } from '@angular/core';
import { BrowserModule } from
'@angular/platform-browser';

@NgModule({
        imports: [ BrowserModule ],
        providers: [ Logger ],
        declarations: [ AppComponent ],
        exports: [ AppComponent ],
        bootstrap: [ AppComponent ]
        })
export class AppModule { }
```

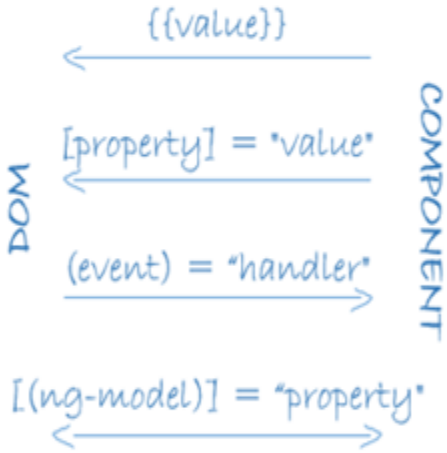

Angular - Components

- Angular utilise des **Components** pour contrôler l'affichage des données.
- Un **Component** ne doit contenir que la logique d'affichage. Toute autre logique est contenue dans les **Services**
- Les fichiers **.html**, **.css** et **.ts** sont séparés et c'est l'entête du **Component** qui fait le lien entre les 3
- Documentation : **https://angular.io/guide/architecture-components**

```
@Component({
        selector: 'mon-component-selector',
        templateUrl: './htmlFile.html',
        styleUrls: './cssFile.css',
        providers: [ Service1,Service2 ]
})
export class MonComponent {
        constructor(private service1: Service1,
                private service2: Service){}
 }
```

Angular – Data binding

- On ne veut pas manipuler le DOM directement, mais plutôt le data qui sera affiché
- **Interpolation** ({{valeur}}) : la valeur d'une variable du component est affiché
- **Property binding** ([property] = "value") : permet de modifier la valeur d'une propriété d'un élément HTML
- **Event binding** ((event) ="handler") : l'evénement event du DOM est géré par une fonction handler du component
- **Two way binding** ([(ngModel)] = "property") : permet de lier un élément HTML et une variable d'un component de manière à ce que modifier un élément modifie l'autre et vice-versa.



Angular - *ngFor

ngFor permet d'itérer à travers une liste en TypeScript et d'afficher ses composants.

```
export class AppComponent {
    heroes = ['Windstorm', 'Bombasto', 'Magneta', 'Tornado'];
}
```

Pour afficher la listes des héros dans le HTML:

```
<li *ngFor="let hero of heroes"> {{ hero }} </li>
```

Angular - *ngIf

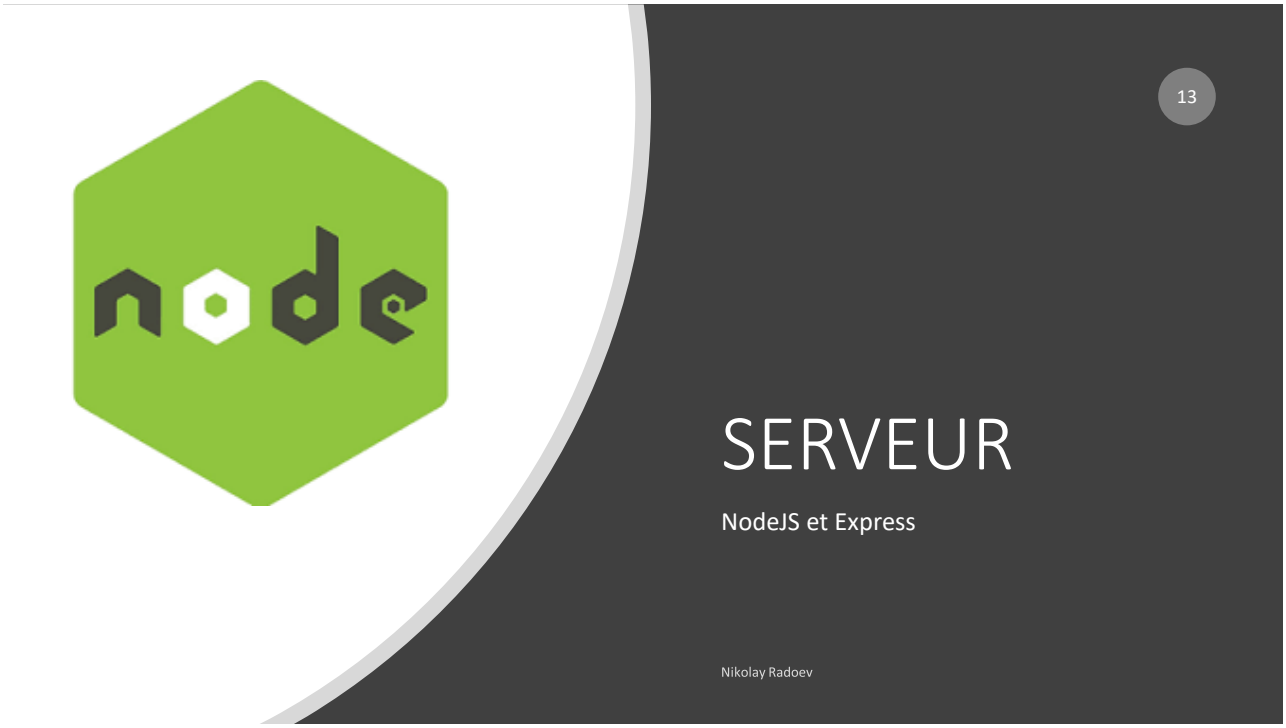
```
export class AppComponent {
    heroes = ['Windstorm', 'Bombasto', 'Magneta', 'Tornado'];
}
```

Ceci sera affiché seulement s'il y a plus qu'un héro:

```
<p *ngIf="heroes.length > 1">Il y a plusieurs héros!</p>
```


Angular – liens utiles

- Documentation officielle d'Angular : **https://angular.io/docs**
- Tutoriel **Tour of Heroes**: **https://angular.io/tutorial**
- AngularCLI : **https://cli.angular.io/**
- CSS : **https://www.w3schools.com/css/**



13

NodeJS

- Environnement d'exécution complet qui permet de rouler JavaScript à l'extérieur d'un fureteur web.
- Bâti sur le **V8 Engine** de Chrome et est facilement utilisable sur plusieurs plateformes (Windows, Linux, OS X).
- Système à un seul fil d'exécution avec une exécution asynchrone et un système d'événements.
- Tutoriel: https://polymtl-web.github.io/tutoriels/node/node

14

NodeJS - Installation

- Pour télécharger Node : https://nodejs.org/fr/download/
- Pour vérifier la version de Node (dans une console) : **node -v**
- La dernière version stable (LTS) : 12.15.0
- Viens avec **npm**, un gestionnaire de paquets pour des projets Web

15

NodeJS – Serveur minimal fonctionnel

```
http.createServer( function(request,response){
    // Mettre les entêtes HTTP
    response.writeHead(200,{'Content-Type':'text/plain'});
    //Envoyer notre message
    response.end('Exemple de serveur Node\n');
}).listen(3000);
```


16

Express

- Cadriciel bâti sur NodeJS pour faciliter la création d'applications web
- Offre un système plus puissant et plus simplifié de **Routing** pour un serveur

- Exemple d'appel d'Express:

```
app.get('/', function(req,res){
   res.send("Exemple d'appel GET!");
})
```

- Tutoriel : https://polymtl-web.github.io/tutoriels/express/express

Express – Serveur avec plusieurs routes

```
var express = require('express');
var app = express();
app.get('/', function(req,res){
    res.send("Exemple de serveur Node avec Express!");
})
app.get('/about',function(req,res){
    res.send("Une page sur nous");
})
app.get('/express',function(req,res){
    res.send("Une page sur ExpressJS");
})
app.listen(3000);
```

Méthodes HTTP

- **GET**
  - La méthode **GET** demande une ou plusieurs données. Les requêtes **GET** devraient préférablement être utilisées afin de récupérer des données.
- **POST**
  - La méthode **POST** sert à envoyer de l'information vers une ressource spécifiée. Ceci cause un changement d'état ou des modifications des données du côté du serveur. En général, l'information se trouve à l'intérieur du **body** de la requête.
- **PUT**
  - La méthode **PUT** permet de remplacer ou modifier une ou plusieurs données. La différence entre **POST** et **PUT** est parfois ambigüe, mais **PUT** est à privilégier lorsqu'on veut modifier de l'information existante et **POST** lorsqu'on veut créer plus de données ou envoyer de l'information.
- **DELETE**
  - La méthode **DELETE** sert à supprimer une ou plusieurs ressources spécifiées par la requête.

19

Create Read Update Delete (CRUD)

Opération	SQL	HTTP
Create	INSERT	POST
Read	SELECT	GET
Update	UPDATE	PUT/POST/PATCH
Delete	DELETE	DELETE

20

21

Accès à distance à PostgreSQL
(déjà installé sur les VMs dans les postes du laboratoire)

Pour autoriser une connexion à distance à PostgreSQL (à partir de votre application), vous devez tout d'abord modifier les fichiers de configuration:

Modification du pg_hba.conf

1- Se positionner dans la VM

2- Accéder au dossier */var/lib/pgsql/9.6/data/*

3- Ouvrir le fichier *pg_hba.conf avec la commande vi*

4- En mode insertion, ajouter la ligne suivante à la fin du fichier:
host all all 0.0.0.0/0 md5

5- Enregistrer le fichier *pg_hba.conf* avec la commande :wq

22

Accès à distance à PostgreSQL
(déjà installé sur les VMs)

Modification du postgresql.conf

6- Toujours dans le même dossier, ouvrez le fichier postgresql.*conf avec la commande vi*

7- En mode insertion, ajouter la ligne suivante à la fin du fichier:

Listen_addresses= '*'

8- Enregistrez le fichier *postgresql.conf* avec la commande :wq

9- Redémarrez le service avec la commande:

service postgresql-9.6.service restart

Accès à distance à PostgreSQL
(déjà installé sur les VMs)

Maintenant au niveau de votre application, dans les paramètres de connexion à PostgreSQL:

→ Il suffit d'ajouter l'adresse IP de votre VM dans Host, ci-dessous un exemple:

user: "user-test",

database: "TP5",

password: "XXXX",

port: 5432,

host: "@ de la VM"

PG (node-postgres)

- Ensemble de modules NodeJs qui permet de communiquer avec une base de données PostgreSQL
- Peut être installé en faisant : **npm install --save pg**
- Site officiel : https://node-postgres.com/
- Vient avec des types pour Typescript : **npm install --save @types/pg**

PG - Connexion

- Avant de se connecter, assurez-vous d'avoir une base de donnée PostgreSQL existante et un utilisateur assigné (on utilise **sysadmin** comme exemple)
- Pour créer un utilisateur dans pgAdmin
  - Login/Group Roles -> bouton droit -> Create
- Pour assigner un utilisateur à une BD
  - maBD -> bouton droit -> Properties -> Security -> choisir l'utilisateur -> Privileges(ALL)

Exemple de connexion (JavaScript)

```
const pg = require('pg');
const pool = new pg.Pool({
    host: '127.0.0.1',
    database: 'pg_exemple',
    port: '5432',
    user: 'sysadmin',
    password: '1234',
    });

pool.query(`CREATE TABLE users(
        id SERIAL PRIMARY KEY,
        firstname VARCHAR(40) NOT NULL,
        lastName VARCHAR(40) NOT NULL)`,   (err, res) =>
{
                console.log(err, res);
});
```

PG – Connexion avec le cadriciel

```
import * as pg from "pg";
public connectionConfig: pg.ConnectionConfig = {
     database: "pg_exemple",
     host: "127.0.0.1",
     port: 5432,
     user: "sysadmin",
     password: "1234",
     keepAlive: true
};
private pool: pg.Pool = new pg.Pool(this.connectionConfig);
```

PG – Connexion avec le cadriciel

```
public constructor() {
        this.pool.connect();
}
```

Note: Il est important d'appeler pool.connect() qu'une seule fois dans le constructeur pour assurer la stabilité du serveur.

PG - Envoi de requêtes

- Pour faire un appel à la base de données, la méthode **query** permet d'envoyer une requête SQL.

- Par exemple, on peut seulement aller chercher la date et l'heure actuelles avec la fonction **NOW()** de PostgreSQL

```
pool.query("SELECT NOW()", (err, res) => {

    console.log(err, res);
});
```

PG – Requêtes sur une table

```
public getHotels(): Promise<pg.QueryResult> {
      this.pool.connect();
      return this.pool.query('SELECT * FROM HOTELDB.Hotel;');
} // Seulement la table Hotel

public getAllFromTable(tableName: string): Promise<pg.QueryResult> {
      this.pool.connect();
      return this.pool.query(`SELECT * FROM HOTELDB.${tableName};`);
} // Le nom de la table est passé en paramètres
```


PG – Requête paramétrée sur une table

```
public getWithParams(tableName: string, params:
object): Promise<pg.QueryResult> {
    this.pool.connect();

    let query: string = `SELECT * FROM
        \'${tableName}'\ \n`;
    const keys: string[] = Object.keys(params);
    if (keys.length > 0) {
    query = query.concat(`WHERE ${keys[0]}
        =\'${params[keys[0]]}\'`);
}
```

```
// On enleve le premier element
keys.shift();

for (const param in keys) {
    const value: string = keys[param];
    query = query.concat(`AND ${value} =
        \'${params[value]}\'`);
}

return this.pool.query(query);

}
```

PG – Insérer des élements

- La méthode **query** peut prendre 2 éléments pour une méthode **INSERT**
  - Le texte de la requête
  - Un tableau de string (string[]) avec les différentes valeurs

- Les éléments du tableau sont référés par $X dans le texte de la requête avec X = le numéro de la valeur.
- **NB:** les valeurs commencent à 1 et non 0

PG – Exemple d'insertion

```
public createRoom(room: Room): Promise<pg.QueryResult> {
	this.pool.connect();

	const values: string[] = [
		room.roomno,
		room.hotelno,
		room.typeroom,
		room.price.toString()
	];
	const queryText: string = `INSERT INTO HOTELDB.ROOM VALUES($1,$2,$3,$4);`;

	return this.pool.query(queryText, values);
}
```

PG – Update et Delete

- Le **UPDATE** et **DELETE** sont similaires à la fonction **INSERT**
- Les deux peuvent utiliser la méthode **query(queryText, values[])**

Exercice

- Ecrire la fonction pour supprimer un hôtel en fonction de son **hotelNo** dans le code du cadriciel

Solution

```
public deleteHotel(hotelNo: string): Promise<pg.QueryResult> {
    this.pool.connect();

    const values: string[] = [hotelNo];
    const queryText: string =`DELETE FROM HOTELDB.HOTEL WHERE hotelNo = $1`;

    return this.pool.query(queryText, values);
}
```


37

38

CADRICIEL

Angular

NodeJS + Express

postgreSQL

Nikolay Radoev

38

Structure

- Le cadriciel qui vous est fourni contient un **Serveur**, un **Client** et permet la connexion à une base de données PostgreSQL
- Le contexte du cadriciel est celui de l'exercice 1 du TP2 : Hotel
- Le code est donné à titre indicatif, vous ne devez pas garder tout le code pour la remise finale

Installer le projet

- Vérifiez que vous avez NodeJs installé avec **node –v**
  - Si vous ne l'avez pas, téléchargez le de https://nodejs.org/en/download/
- Allez dans le dossier **client** et lancez **npm install**
- Allez dans le dossier **server** et lancez **npm install**

Lancer le projet

- **Avant de lancer le projet**
  - Assurez-vous que Postgres roule sur la machine
  - Créez une base de données et lui ajouter un utilisateur
  - Allez dans /server/app/controllers/database.service.ts et modifiez **connectionConfig** avec les bons paramètres de votre BD

- **Lancer le projet**
  - Allez dans /server et faites **npm start**
  - Allez dans /client et faites **npm start**
    - Une fenêtre de votre fureteur doit s'ouvrir, sinon allez à **localhost:4200**

SERVEUR

- La majorité du code est concentré dans :
  - **controllers/database.controller.ts** : la définition de vos routes
  - **services/database.service.ts** : le service qui communique avec la base de données

- Le serveur expose un API partiel qui permet d'interagir avec la base de données. L'API se concentre sur les tables HOTEL et ROOM
- Pour accéder à l'API, vous pouvez appeler **localhost:3000/database/**

SERVER - API

- **/hotel** : Permet d'obtenir tout les hotels
- **/hotel/hotelNo** : Permet d'obtenir les PKs de Hotel
- **/hotel/insert** : Permet d'insérer un hotel dans la BD

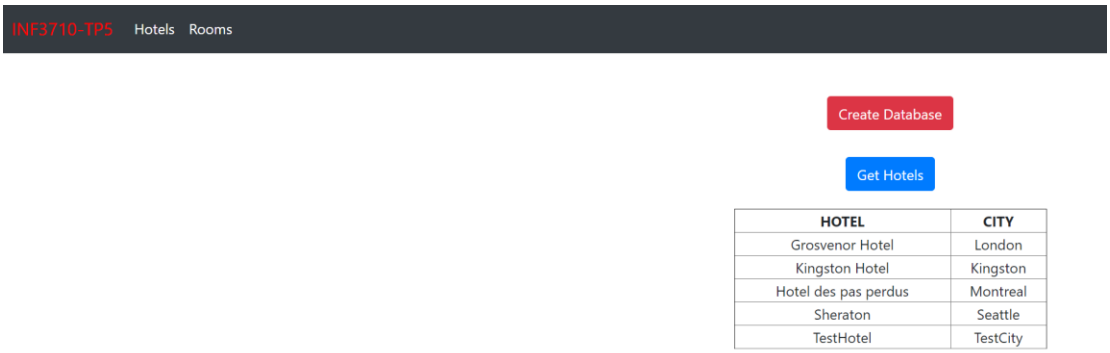
- **/rooms** : Permet de chercher des rooms. La requête peut prendre des paramètres dans la **query**. Tout les paramètres sont optionels
  - Ex: localhost:3000/database/rooms?hotelNo=H111&typeroom=S&price=100
    - Permet de trouver tout les chambres de l'hotel 'H11' de type 'S' à un prix de 100
- **/rooms/insert** : Permet d'insérer une room dans la BD

SERVER – API (autres)

- **/createSchema**: Insère le schema d'Hotel dans la DB. Ceci est une fonction de test pour vous aider à créer votre DB
- **/populateDb**: Insère des valeurs dans la DB. Ceci est une fonction de test pour aider à tester le cadriciel.

- **/tables/:tableName** : Permet d'obtenir tous les valeurs d'une table en fonction du nom de la table.

CLIENT

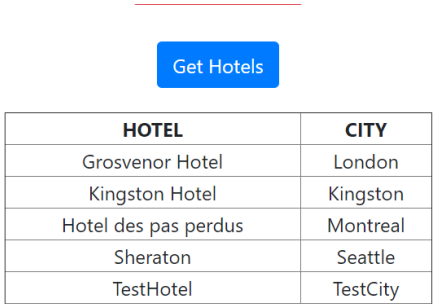


CLIENT

- L'affichage des données se fait dans **AppComponent**.
- **ATTENTION:** Aucune communication avec le serveur ne doit être faite dans un **component**
- La communication avec le serveur se fait dans **CommunicationService.**
- **ATTENTION :** Aucun affichage ne doit être fait dans un **service**

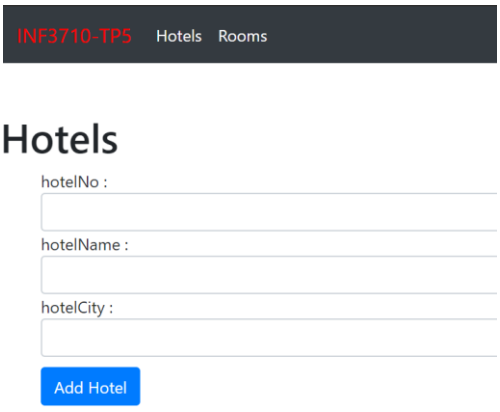
CLIENT – Obtenir des données

- Le bouton **Get Hotels** permet d'obtenir tous les hotels de la base de données et les afficher dans une table
- La fonction **getHotels** dans **CommunicationService** fait l'appel au serveur pour récupérer les données



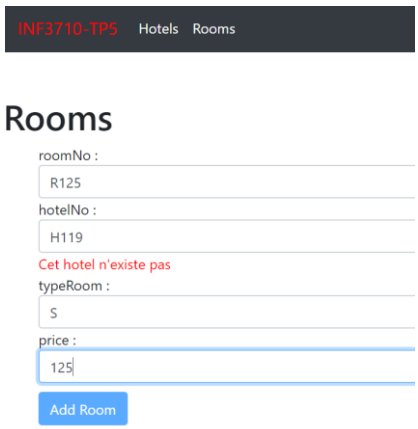
CLIENT – Insérer des données

- Le bouton **Add Hotel** rajoute un hotel avec les éléments dans les 3 champs présents
- Si l'insertion a réussi, la liste des hotels se met à jour automatiquement
- Si l'insertion échoue à cause d'une clé primaire déjà existante, **une erreur est affichée**



CLIENT – Insérer des données

- Le bouton **Add Room** rajoute un hotel avec les éléments dans les 4 champs présents
- L'insertion d'une ROOM est possible **seulement** si la valeur de hotelNo est celle d'un Hotel existant
- **Attention: pour une telle contrainte, vous devez donnez les valeurs possibles à l'utilisateur au lieu d'un input libre**



Conseils

- Toute donnée provenant de la BD doit être chargée à partir de la BD et non pas inscrite à la main ou « encodée » dans un formulaire Web ou dans le code de l'application
- Exemple : On veut modifier la succursale de l'employé E1 de B002 à B0010 dans un formulaire de l'application Web.
  - On doit donc avoir une liste déroulante qui montre toutes les succursales possibles pour que l'usager puisse sélectionner B0010 sans avoir à le taper!
- Règle générale : minimiser le nombre d'éléments qu'un utilisateur doit entrer à la main.

Conseils

- L'expérience utilisateur est aussi importante que le bon fonctionnement de la base de données
  - Les erreurs doivent être bien gérées et clairement expliquées à l'utilisateur.
    Exemple: un échec d'insertion doit être accompagné d'une explication (valeur existante, contrainte non respectée, mauvaises valeurs, etc.)
  - L'interface utilisateur doit être assez claire pour être utilisée par quelqu'un qui ne connait pas la structure de votre base de données
  - Planifiez votre interface en fonction des requis du projet (lisez attentivement l'énoncé)

Références

- https://nodejs.org
- Documentation d'Angular : **https://angular.io/docs**
- AngularCLI : **https://cli.angular.io/**
- CSS : **https://www.w3schools.com/css/**