**1. Gestion de projet (HPR, méthode agile, sprint, trello, scrum, intégration continue, ...) Random : Math.floor(Math.random() \* max + min);**

**Agile**

|  |  |
| --- | --- |
| Définition | Les méthodes agiles sont des méthodologies essentiellement dédiées à la gestion de projets informatiques. Elles reposent sur des cycles de développement itératifs et adaptatifs en fonction des besoins évolutifs du client. Elles permettent notamment d'impliquer l'ensemble des collaborateurs ainsi que le client dans le développement du projet.Ces méthodes permettent généralement de mieux répondre aux attentes du client en un temps limité (en partie grâce à l'implication de celui-ci) tout en faisant monter les collaborateurs en compétences. Ces méthodes constituent donc un gain en productivité ainsi qu'un avantage compétitif tant du côté client que du côté du fournisseur. |
| Les valeurs communes de ces méthodes | L'équipe et la communication avant les outils et processus : dans la vision agile, l'équipe est bien plus importante que les outils ou les procédures de fonctionnement **/** L'application avant la documentation : il est primordial que le projet fonctionne, c'est la priorité avant toute chose. **/** La collaboration avant la négociation : le client doit être impliqué dans le développement. **/** L'acceptation du changement et  la flexibilité avant la planification : la planification initiale et la structure du projet doivent être flexibles afin de permettre les évolutions attendues par le client. |

**Scrum**

|  |  |
| --- | --- |
| Définition | Elle s'appuie sur le découpage des projets en itérations encore nommées « sprints ». Un sprint peut avoir une durée qui varie généralement entre deux semaines et un mois. Avant chaque sprint, les tâches sont estimées en temps et en complexité à l'aide de certaines pratiques comme planning poker. La méthode Scrum est aussi caractérisée par une « mêlée » quotidienne, encore appelée « morning » ou « stand-up » |
| Les rôles de la méthode Scrum | Le product owner : il s'agit du représentant officiel du client au sein d'un projet Scrum. Il est l'interlocuteur principal du Scrum Master et des membres de l'équipe. Il définit les besoins du produit et rédige les spécifications. Il peut se faire aider de responsables fonctionnels pour la rédaction des spécifications. Il est également chargé de définir et prioriser les users stories pour chaque sprint. **/** Le scrum master : il s'agit d'une personne chargée de veiller à la mise en application de la méthode et au respect de ses objectifs. Il ne s'agit pas d'un chef de projet, mais d'une personne chargée de lever les obstacles éventuels qui empêcherait l'avancement de l'équipe et du projet pendant les différents sprints. **/** L'équipe (« team members ») : ce sont les personnes chargées de la réalisation du sprint et d'un produit utilisable en fin de sprint. Il peut s'agir de développeurs, architectes, personnes chargées de faire des tests fonctionnels… |

**2. Côté serveur et BD (vous devez écrire du code, comprendre du code et/ou compléter du code)**

**Toutes ces requêtes prennent un ou des objets en paramètre**

|  |  |
| --- | --- |
| db.collection.insertOne(dataObject) | Inserts a document into a collection. |
| db.collection.insertMany(dataArray) | Inserts multiple documents into a collection. |
| db.collection.find(filter) | Selects documents in a collection based on the filter and returns a cursor to the selected documents. |
| db.collection.updateOne(filter, update) | Updates a single document within the collection based on the filter. |
| db.collection.updateMany(filter, update) | Updates all documents within the collection that match the filter. |
| db.collection.replaceOne(filter, replacement) | Replaces a single document within the collection based on the filter. |
| db.collection.deleteOne(filter) | Removes a single document from a collection based on the filter. |
| db.collection.deleteMany(filter) | Removes all documents that match the filter from a collection. |

**Filtering operators (plug it in the filter field):**

|  |  |
| --- | --- |
| {a: 10, b: "hello"} | Docs where a is 10 and b is "hello" |
| {a: {$gt: 10}} | Docs where a is greater than 10. |
| {a: {$in: [10, "hello"]}} | Docs where a is either 10 or "hello" |
| {a: {$all: [10, "hello"]}} | Docs where a is an array containing both 10 and "hello" |
| {"a.b": 10} | Docs where a is an object with b == 10 |
| {a: {$elemMatch: {b: 1, c: 2}}} | Docs where a is an array that contains an object with attributes b == 1 && c == 2 |
| {$or: [{a: 1}, {b: 2}]} | Docs where a is 1 or b is 2 |
| {a: {$regex: "^m"}}  {a: /^m/} | Docs where a begins with the letter m |
| {a: {$mod: [10, 1]}} | Docs where a % 10 == 1 |
| {a: {$type: "string"}} | Docs where a is a string |
| {$text: {$search: "hello"}} | Docs that contain "hello" on a full text search |
| {a: {$nin: [10, "hello"]}} | Docs where a is different from 10 or "hello" |
| {a: {$size: 3}} | Docs where a in array with 3 elements |
| {a: {$exists: true}} | Docs containing a field named a |
| $lt, $lte, $gt, $gte, $eq, $ne | Lower than, lower than or equal to, greater than, greater than or equal to, equal to, not equal to |

**Field update operator (plug it in the update field)**

|  |  |
| --- | --- |
| {$inc: {a: 2}} | Increment a by 2 |
| {$set: {a: 5}} | Set a to 5 |
| {$unset: {a: 1}} | Deletes a |
| {$max: {a: 10}} | Set a to the greater value between a and 10 |
| {$min: {a: -10}} | Set a to the lower value between |
| {$mul: {a: 2}} | Set a to a \* 2 |
| {$rename: {a: "b"}} | Rename a to b |
| {$bit: {a: {and: 7}}} | Perform bitwise and between a and 7 |
| {$currentDate: {a: {$type: "date"}}} | Set a to the current date |
| {$push: {a: 1}} | Append 1 to the array a |
| {$push: {a: {$each: [1, 2]}}} | Append 1 and 2 to the array a |
| {$push: {a: {$each: [10, 20, 30], $slice: -5}}} | Append 10, 20, 30 to the array a then trims the array to only contain 5 elements |
| {$push: {a: {$each: [50, 60, 70], $position: 0}}} | Insert 50, 60, 70 at position 0 of the array a |
| {$addToSet: {a: 1}} | Append to the value 1 to the array a if it's not there |
| {$addToSet: {a: {$each: [1, 2]}}} | Append to the value 1 and 2 to the array a if they're not there |
| {$pop: {a: 1}} | Remove the last element from the array a |
| {$pop: {a: -1}} | Remove the first element from the array a |
| {$pull: {a: ($gt: 5}}} | Removes all values greater than 5 from a |
| {$pullAll: {a: [5, 6]}} | Removes multiples occurences of 5 or 6 from array a |

**Cursor methods (ex: db.collection.find({name: “yoan”}).limit(42) ) :**

|  |  |
| --- | --- |
| forEach() | Applies a function to each document in a cursor |
| hasNext() | Returns true if the cursor has documents and can be iterated. |
| limit() | Constrains the size of a cursor’s result set. |
| toArray() | Returns an array that contains all documents returned by the cursor. |
| sort() | Returns results ordered according to a sort specification. (ex: {name: 1, age: -1} va sort selon le name ascending et selon age descending |
| next() | Returns the next document in a cursor. |
| size() | Returns a count of the documents in the cursor after applying skip() and limit() methods. |
| skip() | Returns a cursor that begins returning results only after passing or skipping a number of documents. |

**Socket IO const socket = io('http://www.exemple.org/user'); const io = require('socket.io')(http);**

|  |  |
| --- | --- |
| Pour envoyer un message au client | socket.emit(<nom événement>, <message\_1>,...,<message\_m>); |
| Si on veut envoyer un message à tous les sockets | io.emit(<nom événement>, <message\_1>,...,<message\_m>); |
| Lorsque le serveur reçoit une nouvelle connexion, il peut ajouter le nouveau socket à une "room" : | socket.join(room,callback);  Le callback est optionnel. Il est appelé en cas d'erreur. |
| Si on veut envoyer un message à une room | socket.to('others').emit('an event', { some: 'data' });  socket.to('room1').to('room2').emit('hello'); |
| Pour quitter une room | socket.leave(room,callback); |

**REST**

|  |  |  |  |
| --- | --- | --- | --- |
| HTTP Verb | CRUD | Entire Collection (e.g. /customers) | Specific Item (e.g. /customers/{id} |
| POST | Create | 201 (Created), 'Location' header with link to /customers/{id} containing new ID. | 404 (Not Found), 409 (Conflict) if resource already exists..  POST http://www.example.com/customers/12345/orders |
| GET | Read | 200 (OK), list of customers. Use pagination, sorting and filtering to navigate big lists. | 200 (OK), single customer. 404 (Not Found), if ID not found or invalid. GET http://www.example.com/buckets/sample |
| PUT | Update/Replace | 405 (Method Not Allowed), unless you want to update/replace every resource in the entire collection. | 200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid. PUT http://www.example.com/buckets/secret\_stuff |
| PATCH | Update/Modify | 405 (Method Not Allowed), unless you want to modify the collection itself. | 200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid. PATCH http://www.example.com/buckets/secret\_stuff |
| DELETE | Delete | 405 (Method Not Allowed), unless you want to delete the whole collection—not often desirable. | 200 (OK). 404 (Not Found), if ID not found or invalid.  DELETE http://www.example.com/bucket/sample |

**3. Côté client (vous devez écrire du code, comprendre du code et/ou compléter du code)**

**Angular6 - Template syntax**

|  |  |
| --- | --- |
| <input [value]="firstName"> | Binds property value to the result of expression firstName. |
| <div [attr.role]="myAriaRole"> | Binds attribute role to the result of expression myAriaRole. |
| <div [class.extra-sparkle]="isDelightful"> | Binds the presence of the CSS class extra-sparkle on the element to the truthiness of the expression isDelightful. |
| <div [style.width.px]="mySize"> | Binds style property width to the result of expression mySize in pixels. Units are optional. |
| <button (click)="readRainbow($event)"> | Calls method readRainbow when a click event is triggered on this button element (or its children) and passes in the event object. |
| <div title="Hello {{ponyName}}"> | Binds a property to an interpolated string, for example, "Hello Seabiscuit". Equivalent to: <div [title]="'Hello ' + ponyName"> |
| <p>Hello {{ponyName}}</p> | Binds text content to an interpolated string, for example, "Hello Seabiscuit". |
| <my-cmp [(title)]="name"> | Sets up two-way data binding. Equivalent to: <my-cmp [title]="name" (titleChange)="name=$event"> |
| <video #movieplayer ...>  <button (click)="movieplayer.play()">  </video> | Creates a local variable movieplayer that provides access to the video element instance in data-binding and event-binding expressions in the current template. |
| <p \*myUnless="myExpression">...</p> | The \* symbol turns the current element into an embedded template. Equivalent to: <ng-template [myUnless]="myExpression"><p>...</p></ng-template> |
| <p>Card No.: {{cardNumber | myCardNumberFormatter}}</p> | Transforms the current value of expression cardNumber via the pipe called myCardNumberFormatter. |
| <p>Employer: {{employer?.companyName}}</p> | The safe navigation operator (?) means that the employer field is optional and if undefined, the rest of the expression should be ignored. |
| <svg:rect x="0" y="0" width="100" height="100"/> | An SVG snippet template needs an svg: prefix on its root element to disambiguate the SVG element from an HTML component. |
| <svg> <rect x="0" y="0" width="100" height="100"/> </svg> | An <svg> root element is detected as an SVG element automatically, without the prefix. |

**Angular6 - Built-in directives import { CommonModule } from '@angular/common'; @component: selector: “app-root”, templateUrl:”path.html”, styleUrls: [“path.css”]**

|  |  |
| --- | --- |
| <section \*ngIf="showSection"> | Removes or recreates a portion of the DOM tree based on the showSection expression. |
| <li \*ngFor="let item of list"> | Turns the li element and its contents into a template, and uses that to instantiate a view for each item in list. |
| <div [ngSwitch]="conditionExpression">  <ng-template [ngSwitchCase]="case1Exp">...</ng-template>  <ng-template ngSwitchCase="case2LiteralString">...</ng-template>  <ng-template ngSwitchDefault>...</ng-template>  </div> | Conditionally swaps the contents of the div by selecting one of the embedded templates based on the current value of conditionExpression. |
| <div [ngClass]="{'active': isActive, 'disabled': isDisabled}"> | Binds the presence of CSS classes on the element to the truthiness of the associated map values. The right-hand expression should return {class-name: true/false} map. |
| <div [ngStyle]="{'property': 'value'}">  <div [ngStyle]="dynamicStyles()"> | Allows you to assign styles to an HTML element using CSS. You can use CSS directly, as in the first example, or you can call a method from the component. |

**Angular6 -Forms import { FormsModule } from '@angular/forms';**

|  |  |
| --- | --- |
| <input [(ngModel)]="userName"> | Provides two-way data-binding, parsing, and validation for form controls. |

**Angular6 -Class field decorators for directives and components import { Input, ... } from '@angular/core';**

|  |  |
| --- | --- |
| @Input() myProperty; | Declares an input property that you can update via property binding (example: <my-cmp [myProperty]="someExpression">). |
| @Output() myEvent = new EventEmitter(); | Declares an output property that fires events that you can subscribe to with an event binding (example: <my-cmp (myEvent)="doSomething()">). |
| @ViewChild(myPredicate) myChildComponent; | Binds the first result of the component view query (myPredicate) to a property (myChildComponent) of the class. Not available for directives. |

**Angular6 -Class decorators import { Directive, ... } from '@angular/core';**

|  |  |
| --- | --- |
| @Component({...}) class MyComponent() {} | Declares that a class is a component and provides metadata about the component. |
| @Directive({...}) class MyDirective() {} | Declares that a class is a directive and provides metadata about the directive. |
| @Pipe({...}) class MyPipe() {} | Declares that a class is a pipe and provides metadata about the pipe. |
| @Injectable() class MyService() {} | Declares that a class has dependencies that should be injected into the constructor when the dependency injector is creating an instance of this class. |

**Angular6 -Directive and component change detection and lifecycle hooks (implemented as class methods)**

|  |  |
| --- | --- |
| constructor(myService: MyService, ...) { ... } | Called before any other lifecycle hook. Use it to inject dependencies, but avoid any serious work here. |
| ngOnChanges(changeRecord) { ... } | Called after every change to input properties and before processing content or child views. |
| ngOnInit() { ... } | Called after the constructor, initializing input properties, and the first call to ngOnChanges. |
| ngDoCheck() { ... } | Called every time that the input properties of a component or a directive are checked. Use it to extend change detection by performing a custom check. |
| ngAfterContentChecked() { ... } | Called after every check of the component's or directive's content. |
| ngAfterViewInit() { ... } | Called after ngAfterContentInit when the component's views and child views / the view that a directive is in has been initialized. |
| ngAfterViewChecked() { ... } | Called after every check of the component's views and child views / the view that a directive is in. |
| ngOnDestroy() { ... } | Called once, before the instance is destroyed. |
| ngAfterContentInit() { ... } | Called after ngOnInit when the component's or directive's content has been initialized. |

**Angular6 -Routing and navigation import { Routes, RouterModule, ... } from '@angular/router';**

|  |  |
| --- | --- |
| const routes: Routes = [  { path: '', component: HomeComponent },  { path: 'path/:routeParam', component: MyComponent },  { path: 'staticPath', component: ... },  { path: '\*\*', component: ... },  { path: 'oldPath', redirectTo: '/staticPath' },  { path: ..., component: ..., data: { message: 'Custom' } }  ]); | cotnst routing = RouterModule.forRoot(routes);  Configures routes for the application. Supports static, parameterized, redirect, and wildcard routes. Also supports custom route data and resolve. |
| <router-outlet></router-outlet>  <router-outlet name="aux"></router-outlet> | Marks the location to load the component of the active route. |
| <a routerLink="/path">  <a [routerLink]="[ '/path', routeParam ]">  <a [routerLink]="[ '/path', { matrixParam: 'value' } ]">  <a [routerLink]="[ '/path' ]" [queryParams]="{ page: 1 }">  <a [routerLink]="[ '/path' ]" fragment="anchor"> | Creates a link to a different view based on a route instruction consisting of a route path, required and optional parameters, query parameters, and a fragment. To navigate to a root route, use the / prefix; for a child route, use the ./prefix; for a sibling or parent, use the ../ prefix. |
| <a [routerLink]="[ '/path' ]" routerLinkActive="active"> | The provided classes are added to the element when the routerLink becomes the current active route. |
| class CanActivateGuard implements CanActivate {  canActivate(  route: ActivatedRouteSnapshot,  state: RouterStateSnapshot  ): Observable<boolean>|Promise<boolean>|boolean { ... }  } | { path: ..., canActivate: [CanActivateGuard] }  An interface for defining a class that the router should call first to determine if it should activate this component. Should return a boolean or an Observable/Promise that resolves to a boolean. |
| class CanDeactivateGuard implements CanDeactivate<T> {  canDeactivate(  component: T,  route: ActivatedRouteSnapshot,  state: RouterStateSnapshot  ): Observable<boolean>|Promise<boolean>|boolean { ... }} | { path: ..., canDeactivate: [CanDeactivateGuard] }  An interface for defining a class that the router should call first to determine if it should deactivate this component after a navigation. Should return a boolean or an Observable/Promise that resolves to a boolean. |

**Socket IO - import \* as io from 'socket.io-client'; const socket = io(url); socket.join(“roomName”); socket.to(“roomName”).emit(…);**

|  |  |
| --- | --- |
| Pour envoyer un message au serveur: | socket.emit(<nom événement>, message\_1>,...,<message\_m>); |
| Pour traiter un message provenant du serveur: | socket.on(eventName, callback); |
| Pour se déconnecter: | socket.close() |
| 'connect\_error'  'erreur'  'disconnect' | erreur de connexion  le serveur avise qu'une erreur s'est produite  le serveur avise que le socket a été déconnecté |

**4. Assurance qualité (bonne pratique, qualité du logiciel, code smells, … )**

Décrivez trois facteurs d’importance majeure pour assurer le succès d’un développement de logiciel en intégration continue, dans une approche agile. 1) La présence de tests 2) Des commits de granularité très fine 3) Du code de bonne qualité

**TDD Les cinq étapes du TDD:**

|  |
| --- |
| Ecriture d'un premier test |
| Exécuter le test et vérifier qu'il échoue (car le code qu'il teste n'a pas encore été implémenté) |
| Ecriture de l'implémentation pour faire passer le test (il existe différentes manières de corriger ce code) |
| Exécution des tests afin de contrôler que les tests passent et dans ce présent l'implémentation respectera les règles fonctionnelles des tests unitaires |
| Remaniement (Refractor) du code afin d'en améliorer la qualité mais en conservant les mêmes fonctionnalités |

**Avantages:**

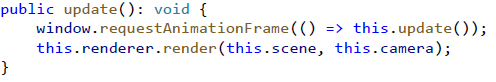
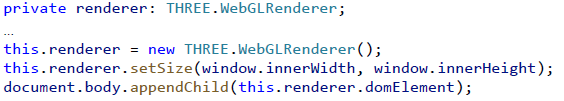
|  |  |
| --- | --- |
| Les tests unitaires sont réellement écrits | On remarque généralement que les tests unitaires sont souvent remis à plus tard (par manque de temps ou d'intérêt). Le fait de commencer par rédiger les tests permet de s'assurer que les tests seront écrits. |
| La satisfaction du développeur permet d'obtenir un code plus cohérent | En suivant la méthode traditionnelle le développeur écrit une unité et va ensuite procéder aux tests afin de s'assurer que ce qu'il a codé est valide. Cette méthode est peu satisfaisante pour un développeur car il est déjà conscient que le code qu'il a écrit est juste. A l'inverse, en appliquant la méthode TDD, il va commencer par rédiger les tests puis ensuite passer à l'implémentation de l'unité afin de faire passer les tests. Ce procédé est plus satisfaisant et motivant car il s'agit d'une sorte de défis de faire valider les tests, C'est une sensation d'accomplissement. Il est important de ne pas ignorer les aspects psychologiques si on veut s'assurer que le travail réalisé par les développeurs soit propre et efficace. |
| Clarification des détails de l'interface et du comportement | En effet, lorsque le développeur écrit du code de test pour tester une implémentation qui n'existe pas encore, il va devoir penser aux détails de la méthode dont il a besoin pour écrire la spécification. Aussi, il va alors s'interroger sur le nom de la méthode, sa valeur de retour, ses paramètres, comportement,...cela permet de clarifier la conception et d'écrire seulement du code utile. |

**5. Gestionnaire de versions Git (son utilité, comprendre son utilisation, les avantages, ...)**

Utilité et avantages: pouvoir revenir à des versions antérieures, facilite le travail collaboratif, facilite la gestion de conflits

|  |  |
| --- | --- |
| git tag 1.0.1-some-other-versionning | Lightweight tag |
| git tag -a v2.1-0 -m "some message" | Tag avec message qui lui est associé |
| git tag | Imprime une liste de tous les tags |
| git show v2.1-0 | Affiche les informations liées au tag (commit hash, commit message, tag message, etc). |
| git reset HEAD some\_file.cpp | Enlève le fichier du staging area (la zone où les fichiers sont avant de commit). |
| git reset HEAD~ | Annule le dernier commit et remet les changements dans le working directory. |
| git checkout -b my\_branch | Crée une branche qui dérive du commit actuel et switch à cette branche. |
| git reset --hard HEAD | Annule le dernier commit et les changements qui viennent avec. |

**6. ThreeJS (Vous devez écrire du code, comprendre du code et/ou compléter du code)**



|  |  |
| --- | --- |
| Mesh = new Mesh(new Geometry (1,2,3), new MeshPhongMaterial(color, transparent: true)) | To create a new mesh object |
| public checkIntersectIce(mouse: THREE.Vector2): THREE.Intersection[] {  this.raycaster.setFromCamera(mouse, this.camera);  let intersects = this.raycaster.intersectObject(this.scene  .getObjectByName("rink").getObjectByName("whiteice"), true);  return intersects; } | Shoot a raycaster |
| This.scene.add(new DirectionalLight(10, 10, 10)) || AmbienLight(couleur, intensite) || spotlight | Ajouter de la lumiere dans la scene |

**7. EXTRA**

**Question Plateforme MVC sur le client et Socket.io (5pts - A18)**

|  |  |  |
| --- | --- | --- |
| Fichier app.component.ts (la documentation du conteneur Map est fournie en annexe) :  import { Component } from ’@angular/core’;  import \* as io from ’socket.io-client’;  @Component({  selector: ’app-root’,  templateUrl: ’./app.component.html’,  styleUrls: [’./app.component.css’]  })  export class AppComponent {  topicList: Map<string,string[]>;  Socket; topic: string; public ngOnInit() {  this.topicList = new Map();  this.socket = io(’http://localhost:8000’);}  public join(topic) {  this.socket.emit(’enter’,topic);  this.socket.on(topic,(message) => {  if (!this.topicList.has(topic)){  this.topicList.set(topic,[message]);}  else {  let mList = this.topicList.get(topic);  mList.push(message);  this.topicList.set(topic,mList);}});}  public selectTopic(topic) {  this.topic = topic;}  public send(topic,message) {this.socket.emit(topic,message);}} | // app.component.html:  <h1>Discussions</h1>  <p>  <input [(ngModel)]="topic">  <button (click)="join(this.topic)">Joindre</button>  </p>  <p>  <input [(ngModel)]="message">  <button (click)="send(this.topic,this.message)">Envoyer message </button>  </p>  <div \*ngFor="let t of topicList.keys()" [ngClass]="{current: t === this.topic}">  <h2> {{t}}</h2>  <p \*ngFor="let message of topicList.get(t)"  class="message" (click)="selectTopic(t)" > {{message}} </p>  </div>  Fichier app.component.css (pas nécessaire de le modifier) : p.message { background-color: #99ffff; } div.current p { background-color: #00cccc; } // Remarque: #00cccc est plus foncé que #99ffff | Code sur le serveur :  let app = require(’express’)();  let http = require(’http’).Server(app);  let io = require(’socket.io’)(http);  let port = 8000;  // À compléter  http.listen(port, function(){  console.log(’listening on \*:’ + port);  });  io.on(’connection’, function(socket){  let topicList = [];  socket.on(’enter’,  (topic) => {  socket.join(topic);  if (!topicList.includes(topic)) {  topicList.push(topic);  socket.on(topic,  (message) => { io.to(topic).emit(topic,message)});});}); |

**MEAN est composé de quatre technologies différentes :**

MongoDB : une base de données de type NoSQL / Express : un cadriciel basé sur Node.js. Très utile pour créer des applications webs et des APIS / Angular : un cadriciel JavaScript qui permet de développer des pages webs selon un modèle MVC / Node : un environnement d'exécution JavaScript construit sur le moteur JavaScript V8 de Chrome. / MEAN peut être essentiellement divisé en trois parties : client (Angular), serveur (Node + Express) et données (MongoDB).

**SQL vs NoSQL**  Les BD relationnelles pas nécessairement adaptées aux grands volumes de données / Les BD NoSQL visent à favoriser les données distribuées / Les BD NoSQL utilisent des modèles de données plus simples, donc moins structurées / Les BD relationnelles reposent sur des technologies matures / Schémas statiques (BD relationnelles) vs schémas dynamiques (BD NoSQL) / Les BD relationnelles assurent l’atomicité des transactions et la cohérence des données

**MongoDB**  Basé sur le concept de document / Un serveur contient des bases de données / Une base de données contient des collections / Une collection contient documents / Distribution horizontale des données (sharding) – Attention: ne pas confondre avec la redondance des données, il s’agit ici de partitionner les données etplacer les partitions sur différents serveurs **Structure d’un document** Un document est essentiellement un objet JSON / Type de données: Double, Object, Array, Binary data, Object id, Boolean, Date, Null, Regular Expression, JavaScript, Symbol, 32-bit integer, Timestamp, 64-bit integer, Min key, Max key / Attribut \_id réservé pour définir une clé primaire

|  |  |
| --- | --- |
| @Injectable()  export class MessageService {  public messages: Message[];  public constructor(  private socketService: SocketService,  private authenticationService: AuthenticationService,  ) {  this.messages = [];  }  public get username(): string | null {  return this.authenticationService.username;  }  public sendMessage(message: string): void {  message = message.trim();  if (this.username && message.length > 0) {  this.socketService.emit(Events.MESSAGE, {  content: message,  sender: this.username,  } as Message);  }  }  public onNewMessage(): Observable<Message> {  return this.receiveMessage().pipe(  tap((message) => this.messages.push(message)),  );  }  private receiveMessage: () => Observable<Message> = () => {  return this.socketService.on<Message>(Events.MESSAGE);  }  } | describe("MessageService", () => {  let socketServiceSpy: jasmine.SpyObj<SocketService>;  beforeEach(() => {  socketServiceSpy = jasmine.createSpyObj("SocketService", ["emit", "on"]);  socketServiceSpy.on.and.returnValue(of(message));  TestBed.configureTestingModule({  providers: [  MessageService,  {provide: SocketService, useValue: socketServiceSpy},  {provide: AuthenticationService, useValue: {username: username}},  ], }); });  it("should be created", inject([MessageService], (service: MessageService) => {  expect(service).toBeTruthy();  }));  it("should send content normally", inject([MessageService], (service: MessageService) => {  service.sendMessage(messageContent);  expect(socketServiceSpy.emit).toHaveBeenCalledTimes(1);  }));  it("should not send content if username not defined", inject([MessageService], (service: MessageService) => {  Object.defineProperty(service["authenticationService"], "username", {value: undefined});  service.sendMessage(messageContent);  expect(socketServiceSpy.emit).toHaveBeenCalledTimes(0);  }));  it("should add content to list", inject([MessageService], (service: MessageService) => {  service.onNewMessage().subscribe();  expect(service.messages[0]).toEqual(message);  }));  }); |

|  |  |
| --- | --- |
| @injectable()  export class MessageSocket {  public constructor(  @inject(Types.MultiplayerService) private multiplayerService: MultiplayerService,  ) {  }  public init(io: SocketIO.Server): void {  io.on(Events.CONNECT, (socket: SocketIO.Socket) => this.onMessage(socket));  }  private onMessage(socket: SocketIO.Socket): void {  socket.on(Events.MESSAGE, async (message: Message) => {  message.timestamp = MessageBuilderService.getTimestamp();  const roomId: string | undefined = this.multiplayerService.getUserRoom(socket.id);  socket.server.in(roomId).emit(Events.MESSAGE, message);  }); }} | chai.use(spies);  describe("BlobService", () => {  let service: BlobService;  beforeEach(() => {  container.snapshot();  service = container.get<BlobService>(Types.BlobService);  });  afterEach(() => {  container.restore();  });  describe("uploadFile", () => {  it("should return an url from gcloud service", async () => {  chai.spy.on(BlobService["bucket"], "upload", () => null);  const gcloudUrl: string = await service.uploadFile("folder", "lol.bmp");  chai.expect(gcloudUrl).to.equal("https://storage.googleapis.com/projet2-v2/folder/lol.bmp");  });  it("should try to delete a folder on cloud storage, never fails", async () => {  chai.spy.on(BlobService["bucket"], "deleteFiles", () => null);  await service.deleteFolder("folder");  }); });}); |

**Exemple Singleton/Transient. NG-NG-NG-XP**

|  |  |  |  |
| --- | --- | --- | --- |
| // transient  @Injectable()  export class MessageService { | @Component({...  providers: [MessageService], // new service every gen  })  export class GameChatComponent implements OnInit { | @Injectable({  providedIn: "root", // singleton  })  export class AuthenticationService { | // inversify.config.ts  container.bind(Types.MultiplayerSocket).to(MultiplayerSocket);  container.bind(Types.MultiplayerService).to(MultiplayerService).inSingletonScope(); |

**RÉPONSES AUX ANCIENS EXAMENS**

**Lorsqu’on fait un «commit», il faut s’assurer que le nombre de lignes de code concernées soit minimal. Pourquoi ?**

S’il y a un bogue, c’est plus facile de trouver sa source.

**Ce code correspond à un «middleware» en Express. Expliquez de manière claire et concise ce concept fondamental en Express.**

Un middleware permet de traiter une requête et de former une réponse. On spécifie une route qui détermine le patron de requête qu’il traite. On peut spécifier le type de requête (GET, POST, etc) ou il peut être générique est s’appliquer à tous les types de requêtes. Lorsque son traitement est terminé, il

peut envoyer la réponse, ou passer au prochain middleware.

**Comment un composant peut-il accéder aux propriétés des composants qui sont ses enfants?**

Le parent doit référencer ses enfants à l’aide d’une référence du DOM. Par la suite, lorsque la vue est initialisée, on peut accéder à l’interface public des enfants.

Ex: ViewChild(“enfant”) enfant: ElementRef<Type>; <div #enfant/>; this.enfant.nativeElement.faireDequoi();

**Programmation asynchrone (1 point) - Soit le code suivant, pour solliciter la création d’une grille de mots croisés à un service REST.**

public creerGrille(info: InfoGrille): Promise<Identificateur> {return this.http.post(URL\_SERVICE,info).toPromise().then(identificateur => this.decode(identificateur)).catch(() => Promise.reject("Erreur"))} Expliquez ce que fait ce code.

On demande service http de Angular d’exécuter une requête POST. L’observable reçu et transformé en promesse. Si la promesse est tenue, on retourne une nouvelle promesse avec l’identificateur décodé. Sinon, on retourne une promesse rejetée.

**Quelle est la requête pour obtenir les cinq premiers restaurants trouvés qui ont un score supérieur à 5 ?** document.find({"score" :{$gt : 5}}).limit(5)

|  |  |  |
| --- | --- | --- |
| import {Component} from "@angular/core";  import {AbstractControl, FormControl,  FormGroup, Validators} from "@angular/forms";  @Component({  selector: "app-loterie",  templateUrl: "./loterie.component.html",  styleUrls: ["./loterie.component.css"],  })  export class LoterieComponent {  public readonly loterieForm: FormGroup;  public submitted: boolean = false; // should be init in ctor  public constructor( ) {  this.loginForm = new FormGroup({  numero: new FormControl(  "", [Validators.required, Validators.min(this.min), Validators.max(this.max),]),});  }  public get numero(): AbstractControl {  return this.loterieForm.controls.numero;  }  public onSubmit(): void { this.submitted = true; }  } | <h1>Loterie</h1>  <form \*ngIf="!submitted" [formGroup]="loterieForm" #formDir="ngForm" (ngSubmit)="onSubmit()">  <p>Choisissez un numero entre {{min}} et {{max}}:  <input [class.error]="numero.invalid && (numero.dirty || numero.touched)" required formControlName="numero">  </p>  <button [hidden]="loterieForm.invalid || loterieForm.pending" type="submit">Soumettre</button>  </form>  <p \*ngIf="submitted">  Vous avez choisi {{numero.value}}  </p> | app.component.css :  .error { background-color: red}  input { border: solid black 1pt; } |