Numéro anonymat :

Consignes

1. Une page A4 est autorisée.
2. Répondez sur la feuille excusivement.
3. Les questions sont à choix multiples. Seule une réponse complète et correcte donne 1 point. Un réponse incomplète ou fausse est comptée comme 0.

Questions Générales

Question 1

Le Web a été conçu dès l'origine autour de trois standards ouverts fondamentaux. Lesquels ?

A. HTTP (HyperText Transfer Protocol) : pour l'échange d'informations.
B. JavaScript : pour rendre les pages interactives
C. HTML (HyperText Markup Language) : pour structurer les pages Web.
D. Base de données relationnelles : pour stocker les données
E. URL (Uniform Resource Locator) : pour identifier chaque ressource du Web de manière unique et décentralisée.

Question 2

Pourquoi les standards (comme HTML, CSS, HTTP) sont-ils essentiels pour le Web ?

A. Ils garantissent que les sites soient jolis sur tous les écrans
B. Ils permettent aux navigateurs, serveurs et outils de communiquer de manière interopérable
C. Ils rendent le Web plus rapide à utiliser
D. Ils empêchent les développeurs d'utiliser JavaScript

Question 3

Pourquoi le protocole HTTP a-t-il été conçu comme un protocole *stateless* (sans état) ?

A. Pour simplifier la montée en charge des serveurs et permettre une meilleure scalabilité
B. Pour obliger les clients à stocker toutes les informations de session localement
C. Pour garantir une exécution plus rapide des scripts côté serveur
D. Pour sécuriser automatiquement les échanges entre client et serveur

Question 4

Lequel des systèmes suivants est un exemple typique de système *stateful* (avec gestion d'état) ?

A. Le protocole HTTP
B. Un serveur DNS
C. Une connexion à une base de données relationnelle avec session utilisateur
D. Un fichier HTML statique consulté par un navigateur

Back/front

Question 5

Pourquoi distingue-t-on le *front-end* et le *back-end* dans une application web ?

A. Pour permettre aux designers de travailler seuls sur toute l'application
B. Pour séparer l'interface utilisateur de la logique métier et des accès aux données
C. Pour obliger les développeurs à écrire deux fois plus de code
D. Parce que le front-end utilise Java et le back-end HTML

Question 6

Comment une application front-end interagit-elle généralement avec le back-end dans une architecture REST ?

A. En modifiant directement la base de données du serveur via JavaScript
B. En appelant des méthodes Java sur le serveur via des sockets
C. En important les fonctions du back-end dans le code HTML
D. En envoyant des requêtes HTTP (GET, POST, etc.) vers des endpoints exposés par le serveur

Question 7

Quelle est la principale différence entre *Vanilla JavaScript* et un *framework front-end* (comme Vue.js, React, Angular) ?

A. Vanilla JavaScript est du code JS natif sans outil supplémentaire, tandis qu'un framework fournit une structure et des outils pour faciliter le développement d'interfaces complexes
B. Vanilla JavaScript est plus lent que tous les frameworks modernes
C. Les frameworks ne permettent pas d'interagir avec le DOM
D. Vanilla JavaScript n'est plus utilisé en développement professionnel

Question 8

Ce code est-il censé afficher quelque chose dans la console ? Si non, pourquoi ?

```
fetch("https://example.com/status");
console.log("Fait");
```

A. Oui, il affichera la réponse du serveur
B. Non, car la requête échoue toujours
C. Non, car le résultat de la requête n'est jamais traité ou affiché
D. Oui, il affichera "Fait" suivi de la réponse

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Question 9

Quels sont les principaux défis à relever lors du développement du backend d'une application web moderne ?

A. Gérer la montée en charge pour supporter un grand nombre de requêtes simultanées.
B. Assurer la sécurité des données et des accès (authentification, autorisation...).
C. Gérer la persistance et la cohérence des données dans la base.
D. Réaliser un design graphique réactif et accessible sur mobile.
E. Organiser l'architecture logicielle (REST, microservices, etc.) de manière maintenable.

Question 10

Dans une architecture web classique en trois couches (frontend, backend, base de données), quelle(s) affirmation(s) est(sont) correcte(s) ?

A. Le frontend est responsable de l'affichage et de l'interaction avec l'utilisateur via le navigateur.
B. La base de données se connecte directement au navigateur pour fournir les données à afficher.
C. Le backend est chargé de la logique métier et de l'accès à la base de données.
D. Le frontend contient toutes les règles métier de l'application.
E. Le backend fournit une API (souvent REST ou GraphQL) que le frontend peut interroger pour obtenir ou modifier des données.

Question 11

Le frontend inclut typiquement :

A. Java et PostgreSQL
B. HTML, CSS, JavaScript
C. Bash et Python
D. JDBC et REST

Question 12

Dans une application web utilisant JavaScript, que permet un appel asynchrone via fetch() ?

A. Il permet de lancer une requête HTTP sans bloquer l'exécution du reste du code.
B. Il bloque l'affichage de la page jusqu'à réception complète des données.
C. Il est souvent utilisé avec .then() ou async/await pour gérer la réponse.
D. Il garantit que les réponses arrivent dans l'ordre des appels envoyés.

Security

Question 13

Quel est le rôle principal du protocole OAuth2 dans une application web ?

A. Chiffrer toutes les données échangées entre le front-end et le back-end
B. Authentifier un utilisateur en utilisant un serveur LDAP
C. Permettre à un utilisateur de donner un accès limité à ses données à une application tierce, sans partager son mot de passe
D. Vérifier automatiquement que le code JavaScript du client est fiable

Question 14

Dans un scénario OAuth2 (flux d'autorisation classique), quel échange se produit en premier ?

A. Le client demande directement un jeton d'accès au serveur de ressources
B. Le serveur de ressources envoie le mot de passe de l'utilisateur au client
C. Le client redirige l'utilisateur vers le serveur d'autorisation pour obtenir son consentement
D. L'utilisateur envoie son token directement au serveur de ressources

Question 15

Dans OAuth2, quel est le rôle du *Resource Owner* ?

A. C'est l'utilisateur qui autorise l'accès à ses données
B. C'est le serveur qui protège les données
C. C'est l'application cliente
D. C'est le serveur d'authentification qui délivre les tokens

Question 16

Dans le protocole OAuth2, à quoi sert le `client_id` ?

A. À identifier l'utilisateur connecté
B. À vérifier la validité du jeton d'accès
C. À identifier de manière unique l'application cliente auprès du serveur d'autorisation
D. À sécuriser l'accès aux données en chiffrant les requêtes

REST

Question 17

Qu'est-ce que signifie le terme REST dans le contexte du développement web ?

A. Un protocole sécurisé pour l'échange de données entre serveurs
B. Un format de données utilisé pour représenter des objets en JSON
C. Un système de gestion de sessions utilisateurs basé sur les cookies
D. Un style d'architecture pour concevoir des services web en utilisant les méthodes HTTP standards

Question 18

Parmi les contraintes suivantes, laquelle ne fait pas partie des contraintes définies par l'architecture REST ?

A. Client-Serveur
B. Stateless
C. Cacheable
D. Authentification par token

Question 19

Une API REST bien conçue doit respecter :

A. Le format XML uniquement
B. Le principe HATEOAS
C. L'utilisation d'un SGBD relationnel
D. La persistance côté client

Question 20

Parmi les requêtes suivantes, laquelle utilise le verbe HTTP de manière appropriée selon les principes REST ?

A. GET /users : Crée un nouvel utilisateur dans la base de données
B. PUT /users/42 : Met à jour complètement l'utilisateur d'identifiant 42
C. DELETE /orders : Récupère toutes les commandes existantes
D. POST /products/56 : Supprime le produit d'identifiant 56

Question 21

Voici un exemple de contrôleur REST Spring simple :

```
@RestController
@RequestMapping("/api")
public class MyController {

    @GetMapping("/greet")
    public ResponseEntity<String> greet() {
        return ResponseEntity.ok("Hello, welcome to the API!");
    }

    @PostMapping("/submit")
    public ResponseEntity<String> submitData(@RequestBody String data) {
        return ResponseEntity.ok("Data received: " + data);
    }
}
```

Quelle commande `curl` utiliseriez-vous pour appeler ces points de terminaison dans le contrôleur ci-dessus ?

A. `curl -X GET http://localhost:8080/api/greet`
B. `curl -X POST -d "some data" http://localhost:8080/api/submit`
C. `curl -X POST http://localhost:8080/api/greet`
D. `curl -X GET http://localhost:8080/api/submit`

Question 22

Le message JSON suivant est-il un *self-descriptive message* ? Justifiez votre réponse.

```
{
  "id": 123,
  "name": "John Doe",
  "age": 29
}
```

A. Oui, car il contient toutes les informations nécessaires pour comprendre la ressource.
B. Non, car ce message ne fournit aucune information contextuelle ou sémantique sur les données (pas de contexte ou de type de ressource).
C. Oui, car le format JSON est toujours auto-descriptif par défaut.
D. Non, car les données sont manquantes pour qu'il soit complet.

.....

.....

.....

.....

SPRING

Question 23

Quel principe de l'architecture REST est particulièrement bien supporté par Spring ?

A. L'architecture stateless, où chaque requête contient toutes les informations nécessaires pour être traitée sans dépendance sur l'état précédent.
B. La gestion d'état côté serveur pour maintenir la cohérence des transactions entre les clients.
C. La gestion centralisée des transactions et des états de session.
D. L'agrégation de services au sein d'un même endpoint pour améliorer la performance des requêtes.

Question 24

Quelle est la principale différence entre un *Service* et un *Controller* dans une application Spring ?

A. Un *Controller* est responsable de la logique métier, tandis qu'un *Service* gère les requêtes HTTP et la gestion des vues.
B. Un *Service* est utilisé uniquement pour gérer les erreurs, tandis qu'un *Controller* s'occupe de la logique métier.
C. Il n'y a pas de différence entre un *Service* et un *Controller*, ils ont exactement les mêmes responsabilités dans Spring.
D. Un *Controller* gère les requêtes HTTP et la communication avec le client, tandis qu'un *Service* contient la logique métier et les interactions avec la base de données ou autres services.

Question 25

Quelle commande Maven permet de lancer une application Spring Boot ?

A. `mvn spring-boot:run`
B. `mvn start-app`
C. `java -jar target/app.jar`
D. `spring run`

Question 26

Quelle annotation est utilisée pour indiquer une méthode POST ?

A. `@GetMapping`
B. `@PostMapping`
C. `@RestController`
D. `@RequestBody`

Question 27

Le code suivant se trouve dans un contrôleur Spring dédié à la gestion des poules dans une ferme virtuelle. Est-ce que ce code est à sa place dans le contrôleur ? Justifiez votre réponse.

```
@RestController
@RequestMapping("/farm")
public class FarmController {

    @GetMapping("/eggs/{age}")
    public int getEggsPerDay(@PathVariable int age) {
        if (age < 1) {
            return 0; // Les poussins ne pondent pas
        } else if (age <= 2) {
            return 1; // Poules jeunes (1-2 ans)
        } else {
            return 2; // Poules adultes (plus de 2 ans)
        }
    }
}
```

A. Oui, c'est correct. Il est acceptable de mettre la logique métier directement dans le contrôleur pour des cas simples comme celui-ci.

B. Oui, mais la logique devrait être placée dans un `@Service` et non directement dans le contrôleur, car il est plus facile de tester le contrôleur ainsi.

C. Non, la logique métier (calcul des œufs) devrait être dans un `Service` séparé, et non dans le contrôleur. Le contrôleur devrait uniquement gérer la réception de la requête et la réponse au client.

D. Non, ce calcul devrait être effectué uniquement dans une base de données et non dans le code du contrôleur.

.....

.....

.....

.....

Question 28

Que fait l'annotation `@Autowired` ?

A. Elle exécute une requête HTTP

B. Elle injecte automatiquement une dépendance

C. Elle configure une propriété de l'application

D. Elle mappe une requête vers une méthode

Question 29

Où configure-t-on le port du serveur dans une application Spring Boot ?

A. Dans le fichier `pom.xml`

B. Dans le contrôleur

C. Dans la base de données

D. Dans `application.properties`

Question 30

Est-il conseillé d'utiliser l'annotation `@PreAuthorize` dans le contrôleur pour définir des règles de sécurité ? Justifiez votre réponse.

A. Non, il est préférable de centraliser la logique de sécurité dans la configuration Spring Security pour une gestion uniforme et plus facile de la sécurité.

B. Oui, `@PreAuthorize` est très pratique pour définir des règles spécifiques par méthode et simplifie la configuration de sécurité.

C. Non, `@PreAuthorize` est une mauvaise pratique car il est préférable de définir des règles de sécurité uniquement dans des services externes.

D. Oui, mais `@PreAuthorize` est uniquement utilisé pour des situations de sécurité très simples, pas pour une gestion avancée des accès.

.....

.....

.....

.....

Question 31

Dans une application Spring Boot sécurisée avec Spring Security, quelle est la manière correcte de récupérer l'utilisateur authentifié dans un contrôleur REST sécurisé ?

A. Utiliser la méthode `SecurityContextHolder.getContext().getAuthentication().getName()` pour récupérer le nom de l'utilisateur.

B. Utiliser l'annotation `@AuthenticationPrincipal` pour injecter directement l'utilisateur authentifié dans la méthode du contrôleur.

C. Utiliser l'annotation `@RequestHeader("Authorization")` pour extraire le jeton d'authentification et récupérer l'utilisateur.

D. Utiliser l'annotation `@UserPrincipal` pour récupérer l'utilisateur connecté.

ORM

Question 32

Quelle est la fonction d'un ORM ?

A. Gérer la sécurité des accès

B. Convertir les pages HTML en PDF

C. Mapper des objets avec des tables relationnelles

D. Contrôler la mise en page des pages web

Question 33

Vous devez modéliser une relation entre deux entités : une **Ferme** et des **Poules**. Une ferme peut avoir plusieurs poules. Quelle est la façon correcte de représenter cette relation **1-à-N (One-to-Many)** en JPA ?

```
@Entity
public class Farm {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;

    @OneToMany(mappedBy = "farm")
    private List<Chicken> chickens;

    // getters and setters
}

@Entity
public class Chicken {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;
```

```
    @ManyToOne
    @JoinColumn(name = "farm_id")
    private Farm farm;

    // getters and setters
}
```

Quelle annotation est utilisée pour représenter correctement la relation **1-à-N** entre la **Ferme** et les **Poules** ?

A. `@ManyToOne` sur l'attribut `chickens` de la classe `Farm` et `@OneToMany` sur l'attribut `farm` de la classe `Chicken`.
B. `@OneToMany` sur l'attribut `farm` de la classe `Farm` et `@ManyToOne` sur l'attribut `chickens` de la classe `Chicken`.
C. `@ManyToMany` pour les deux entités.
D. `@OneToMany` sur l'attribut `chickens` de la classe `Farm` et `@ManyToOne` sur l'attribut `farm` de la classe `Chicken`.

Question 34

Quel est le rôle d'un `Repository` dans Spring Boot ?

A. Gérer les styles CSS
B. Accéder aux entités persistées en base de données
C. Générer une clé primaire
D. Authentifier l'utilisateur

Question 35

Que fait `@SpringBootApplication` ?

A. Elle lance uniquement le contrôleur principal
B. Elle connecte à la base de données
C. Elle combine les annotations `@Configuration`, `@EnableAutoConfiguration` et `@ComponentScan`
D. Elle désactive les logs

Question 36

Vous avez une entité **Chicken** (poule) et vous souhaitez créer un **repository** Spring Data JPA pour effectuer des opérations CRUD sur les poules. Voici l'entité **Chicken** et le **repository** correspondant. Quel est le rôle du **repository** et comment l'utiliser pour récupérer une poule par son 'id' ?

```
@Entity
public class Chicken {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;

    @ManyToOne
    @JoinColumn(name = "farm_id")
    private Farm farm;

    // getters et setters
}

public interface ChickenRepository extends JpaRepository<Chicken, Long> {
    // Méthode pour récupérer une poule par son id
}
```

.....

.....

.....

.....

Comment utiliseriez-vous le **ChickenRepository** dans un contrôleur pour récupérer une poule par son 'id' ?

A. Utiliser la méthode `findById()` fournie par `JpaRepository` pour récupérer une poule par son `id`.
B. Utiliser la méthode `getById()` pour récupérer une poule par son `id`.
C. Utiliser la méthode `findOne()` pour récupérer une poule par son `id`.
D. Utiliser une requête SQL native avec `@Query` pour récupérer la poule par son `id`.

Question 37

Comment Spring Boot gère-t-il la création automatique des tables ?

A. Via les fichiers HTML
B. Grâce à la configuration `spring.jpa.hibernate.ddl-auto`
C. Par injection avec @Autowired
D. Par un script bash

Question 38

Création d'entité dans un service Spring

Complétez le code suivant pour créer une entité **Chicken** et l'enregistrer dans la base de données à l'aide d'un service Spring. L'entité doit être persistée en utilisant le **repository JPA**.

```
@Service
public class ChickenService {

    private final ChickenRepository chickenRepository;

    @Autowired
    public ChickenService(ChickenRepository chickenRepository) {
        this.chickenRepository = chickenRepository;
    }

    public Chicken createChicken(String name, Long farmId) {
        Chicken chicken = new Chicken();
        chicken.setName(name);
        ______1____; // Comment associer la ferme à la poule ?

        ______2____; // Quelle méthode utiliser pour enregistrer la poule dans la base ?

        return chicken;
    }
}
```

Quel est le bon choix pour chaque espace (1 et 2) ?

A. `chicken.setFarm(new Farm(farmId))`, `chickenRepository.save(chicken)`
B. `chicken.setFarm(farmId)`, `chickenRepository.save(chicken)`
C. `chicken.setFarm(farmRepository.findById(farmId).get())`, `chickenRepository.save(chicken)`
D. `chicken.setFarm(farmId)`, `chickenRepository.saveAll(Collections.singletonList(chicken))`