ING1 - Développement WEB

Mardi 20 Mai 2025
08h00 - 10h00

Examen blanc TD2
Contrôle continu

Jordan SABLON **EIL Côte d'Opale, Calais**

Modalités de l'examen

- L'épreuve est à réaliser sur un ordinateur des salles informatiques de l'école
- L'accès à Internet n'est pas autorisé durant l'épreuve
- Seuls les documents fournis avec le sujet d'examen sont autorisés

Procédure pour rendre votre évaluation

Dans le dossier `backend` : supprimer le dossier « `node_modules` » s'il existe.

Dans le dossier `frontend` : supprimer les dossiers « `node_modules` » et « `.next` » s'ils existent.

Compressez votre dossier `code` dans une archive au format `.zip` ou `.rar` nommée **« web_NOM_Prenom »** et déposez-le à cet endroit sur votre machine : « V:/donnees/TESTEXAM ».

⚠Le non respect du format de dépôt ou du nommage de l'archive entrainera une pénalité sur la note de l'examen

Pré-requis techniques

Afin de pouvoir réaliser cet examen, votre ordinateur doit disposer des logiciels suivants :

- NodeJS
- NPM
- VS Code (ou tout autre environnement de développement de votre choix)
- Un navigateur web

Barême de notation

Le barême de notation est donné à titre indicatif à la fin du sujet d'examen.

Précisions

- Vous pouvez commencer au choix par l'exercice 1 ou l'exercice 2
- Vous disposez des supports et codes des TDs et TPs ainsi que de documentations supplémentaires qui pourraient vous être utiles dans le dossier « ressources »
- ⚠Attention à bien récupérer le contenu qui se trouve dans le dossier « V:/donnees/SUJET » et à bien travailler depuis votre session au risque de ne pas pouvoir sauvegarder votre travail !

Configuration de l'environnement de développement

1. Récupération du code source

Récupérer le code source fourni avec le sujet d'examen et ouvrir le dossier « code » dans VS Code (ou l'IDE de votre choix)

2. Exercice 1 : Node.js

Depuis le dossier « backend », exécutez dans un terminal la commande suivante pour démarrer le serveur : **« npm start »**

La console devrait vous indiquer ceci :

```
PS C:\Users\JordanSablon\Documents\dev\examen\backend> npm start

> backend@1.0.0 start
> nodemon

[nodemon] 2.0.22
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): src\**\*
[nodemon] watching extensions: ts
[nodemon] starting `ts-node-esm src/index.ts`
Server running at http://127.0.0.1:5000/
Connected successfully
```

Figure 1: Terminal VS Code indiquant que le serveur est démarré

Et le serveur devrait être accessible via l'adresse (http://localhost:5000/) et devrait afficher « Hello ! »

3. Exercice 2 : React

Depuis le dossier « frontend », exécutez dans un terminal la commande suivante afin de démarrer le serveur : **« npm start »**

La console devrait vous indiquer ceci :

```
PS C:\Users\JordanSablon\Documents\dev\examen\frontend> npm start

> frontend@0.1.0 start
> next dev

- ready started server on 0.0.0.0:3000, url: http://localhost:3000
- event compiled client and server successfully in 3.3s (306 modules)
- wait compiling...
- event compiled client and server successfully in 342 ms (306 modules)
- wait compiling /page (client and server)...
- event compiled client and server successfully in 2.6s (561 modules)
```

Figure 2: Terminal VS Code indiquant que le frontend est démarré

Et le serveur devrait être accessible via l'adresse (http://localhost:3000/) et devrait afficher « Hello ! »

Exercice 1 - Backend NodeJS

Dans cet exercice, vous travaillerez depuis l'application présente dans le dossier « backend »

Vous disposez d'une application NodeJS ainsi que d'un jeu de données mocké.

Pour rappel les données d'un mock sont stockées dans un fichier et sont réinitialisées à chaque redémarrage du serveur. Ils permettent de ne pas avoir recours à l'usage d'une base de données.

On retrouve les différents éléments de l'architecture étudiée en cours, à savoir :

- **une couche dédiée au routage :** permet d'appeler un contrôleur spécifique pour chaque type d'appel HTTP
- **une couche de contrôleur :** permet de récupérer les données de la requête et fait appel à un service dédié
- **une couche de service :** permet de réaliser le traitement souhaité sur la donnée

Travail à réaliser

Vous devez implémenter les routes suivantes dans l'application :

GET	/users	Retourne la liste des initiales de chaque utilisateur
GET	/users/admins	Retourne la liste des utilisateurs possédant le rôle administrateur

Vous pouvez tester le retour de vos appels HTTP via ces URLs :

`http://localhost:5000/users`

`http://localhost:5000/users/admins`

Exemple, le premier appel devra retourner le résultat suivant :

["ML","BÉ","LN","DC","MJ","LS","CT","GC","PR","FI"]

Exemple, le second appel devra retourner le résultat suivant :

[
{"id":2,"firstname":"Émilie","name":"Bernard","role":"administrateur"},
{"id":7,"firstname":"Théo","name":"Caron","role":"administrateur"},
{id":10,"firstname":"Inès","name":"Fontaine","role":"administrateur"}
]

Exercice 2 - Frontend React

Le but de cet exercice est de permettre la saisie d'un nombre et de vérifier si ce nombre fait partie de la suite de Fibonacci. On gardera en mémoire chaque valeur ajoutée.

Travail à réaliser

Créer un composant <Saisie /> qui contiendra un champ de saisie de texte (<input />). Ce champ de saisie sera précédé du texte "Saisir un nombre :". On ajoutera également un bouton "Ajouter" qui permettra d'ajouter la saisie afin de la stocker uniquement si la saisie est un nombre.

Créer un composant <Nombres /> qui prendra en propriété la liste des nombres saisis. Ce composant affichera, pour chaque nombre, s'il fait partie ou non de la suite de Fibonacci. Lorsque c'est le cas, le texte apparaitra en gras.

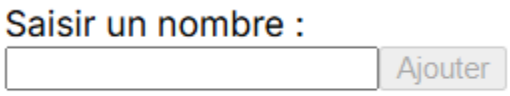
Le bouton "Ajouter" sera désactivé si le champ de saisie est vide.

Le composant <Nombres /> ne sera pas affiché si la liste de nombres est vide.

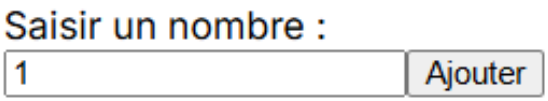
Cas d'usages

Voici quelques captures d'écran de l'application à réaliser :

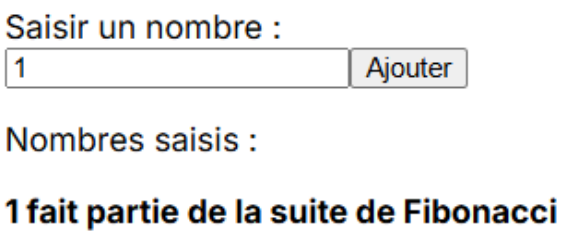
1. On arrive sur l'application, on obtient ce résultat :



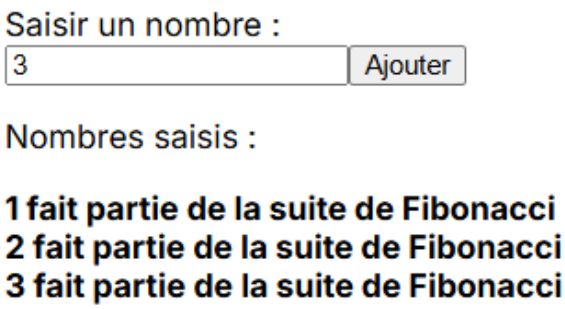
2. On saisie un premier nombre :



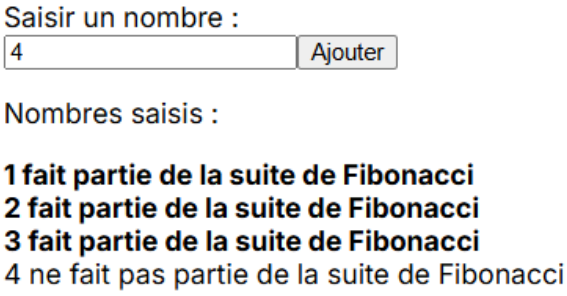
3. Après avoir cliqué sur "Ajouter", on obtient le résultat suivant :



4. Après plusieurs ajouts, on obtient le résultat suivant :



5. Après avoir ajouté un nombre qui n'appartient pas à la suite de Fibonacci, on obtient le résultat suivant :



Aide

Voici la fonction qui permet de déterminer si un nombre appartien à la suite de Fibonacci :

```
const isPerfectSquare = (x: number) => {
    return x > 0  Math.sqrt(x)
}


const isFibonacci = (n: number) => {
    return isPerfectSquare(5*n*n+4) || isPerfectSquare(5*n*n-4);
}
```

Voici comment vérifier si une variable est un nombre :

```
value.match(/^[0-9]+$/)
```

Barème

Le barème donné ci-dessous est uniquement à titre indicatif et est susceptible d'être adapté lors de la correction de l'examen.

Exercice n°1 - Backend NodeJS : 8 points

Critère n°1	GET /users	4 points
Critère n°2	GET /users/admins	4 points

Exercice n°2 - Frontend React : 12 points

Critère n°1	Gestion de la saisie	2 points
Critère n°2	Ajoute de la saisie à la liste	4 points
Critère n°3	Conditionner l'affichage du composant <Nombre />	1 point
Critère n°4	Vérifier si le nombre fait partie de la suite de Fibonacci	2 points
Critère n°6	Affichage du résultat	3 points

Fin de l'énoncé