Développement Web Full Stack

Chapitre 1 : Introduction au Full Stack

Ibrahim ALAME

Cours Magistral - 2h

November 25, 2025

Plan du chapitre

Qu'est-ce que le développement Full Stack ?

Définition

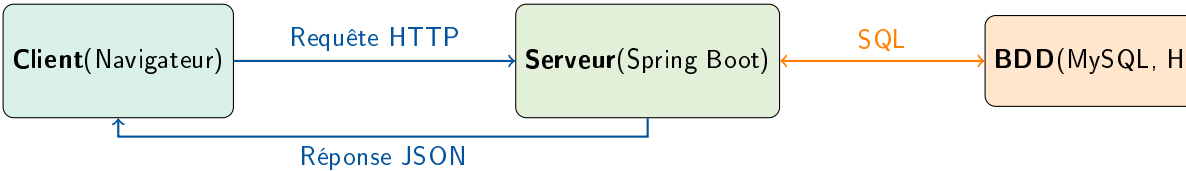
Un développeur **Full Stack** maîtrise à la fois :

- Le **Frontend** : interface utilisateur (ce que l'utilisateur voit)
- Le **Backend** : logique serveur et base de données

V Frontend
HTML, CSS, JavaScript
Vue.js, React, Angular

♠ Backend
Java, Python, Node.js
Spring Boot, Django, Express

Architecture Client/Serveur



- Le **client** envoie des requêtes HTTP
- Le **serveur** traite les requêtes et interagit avec la BDD
- Les données sont échangées au format **JSON**

SPA vs MPA

MPA - Multi Page Application

- Chaque page = nouvelle requête serveur
- Rechargement complet de la page
- Exemple : Sites web traditionnels

SPA - Single Page Application

- Une seule page HTML chargée
- Mise à jour dynamique via JavaScript
- Exemple : Gmail, Facebook

Notre choix : SPA avec Vue.js

Vue.js est un framework JavaScript parfait pour créer des SPA modernes et réactives.

Communication via API REST

REST = Representational State Transfer

Architecture pour concevoir des services web basés sur HTTP.

Méthode HTTP	Action CRUD	Exemple
GET	Read (Lire)	`GET /api/todos`
POST	Create (Créer)	`POST /api/todos`
PUT	Update (Modifier)	`PUT /api/todos/1`
DELETE	Delete (Supprimer)	`DELETE /api/todos/1`

- Les données sont envoyées/reçues en **JSON**
- Chaque ressource a une **URL unique** (endpoint)

Qu'est-ce que Vue.js ?

Vue.js en bref

- Framework JavaScript **progressif**
- Créé par **Evan You** en 2014
- Focalisé sur la **couche vue** (UI)
- Facile à apprendre et à intégrer

Pourquoi Vue.js ?

- Syntaxe intuitive et claire
- Excellente documentation
- Écosystème riche (Vue Router, Pinia)
- Performant et léger

Vue.js 3
Composition API + TypeScript

Vue CLI : Créer un projet

Avec Vite + TypeScript (recommandé)

```
# Creer un nouveau projet avec TypeScript
npm create vite@latest mon-projet -- --template vue-ts

# Se deplacer et installer
cd mon-projet
npm install

# Lancer le serveur de developpement
npm run dev
```

Alternative : Vue CLI

```
npm install -g @vue/cli
vue create mon-projet  # Selectionner TypeScript
```

Structure d'un projet Vue.js

```
mon-projet/
 node_modules/
 public/
  index.html
 src/
  assets/
  components/
  App.vue
  main.ts
 package.json
 tsconfig.json
 vite.config.ts
```

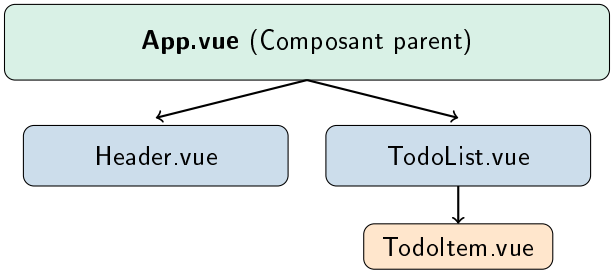
- `src/` : Code source
- `components/` : Composants réutilisables
- `App.vue` : Composant racine
- `main.ts` : Point d'entrée
- `tsconfig.json` : Config TypeScript

Les Composants : Pierre angulaire de Vue

Qu'est-ce qu'un composant ?

Un **composant** est une unité réutilisable qui encapsule :

- Le **template** (HTML) : structure
- Le **script** (JavaScript) : logique
- Le **style** (CSS) : apparence



Structure d'un composant Vue (SFC)

Single File Component avec Composition API (.vue)

```vue
<script setup lang="ts">
import { ref } from 'vue'

const message = ref<string>('Bonjour Vue.js!')

function saluer(): void {
  message.value = 'Vous avez clique!'
}
</script>

<template>
  <div class="greeting">
    <h1>{{ message }}</h1>
    <button @click="saluer">Cliquez-moi</button>
  </div>
</template>
```

Props : Passer des données aux composants

Composant parent

```
<template>
  <TodoItem :titre="maTache" :fait="false" />
</template>
```

Composant enfant (TodoItem.vue)

```
<script setup lang="ts">
interface Props {
  titre: string
  fait?: boolean
}
const props = withDefaults(defineProps<Props>(), {
  fait: false
})
</script>
```

ref, reactive et computed

ref() - Valeurs primitives

```
import { ref } from 'vue'

const compteur = ref<number>(0)
const nom = ref<string>('Vue')

// Modifier : utiliser .value
compteur.value++
```

reactive() - Objets

```
import { reactive } from 'vue'

const user = reactive({
  nom: 'Jean',
  age: 25
})
// Pas de .value
user.age++
```

computed() - Calculé

```
import { computed } from 'vue'

const message = computed(() => {
  return `Compteur: ${compteur.value}`
})

const estPair = computed(() => {
  return compteur.value % 2 === 0
})
```

computed : valeurs calculées, mises en cache automatiquement.

Spring Boot : Rappel

Spring Boot en bref

- Framework Java pour créer des applications web
- Configuration automatique (auto-configuration)
- Serveur embarqué (Tomcat)
- Parfait pour créer des **API REST**

Dépendances essentielles :

- Spring Web
- Spring Data JPA
- H2 Database (dev) / MySQL (prod)

Spring Boot
Java Backend Framework

Créer un endpoint REST

Controller REST simple

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

    @GetMapping("/hello")
    public String hello() {
        return "Bonjour depuis Spring Boot!";
    }

    @GetMapping("/todos")
    public List<Todo> getAllTodos() {
        return todoService.findAll();
    }

    @GetMapping("/todos/{id}")
    public Todo getTodoById(@PathVariable Long id) {
        return todoService.findById(id);
```

Format JSON

Requête GET /api/todos

```
[
  {
    "id": 1,
    "titre": "Apprendre Vue.js",
    "fait": false
  },
  {
    "id": 2,
    "titre": "Creer une API",
    "fait": true
  }
]
```

- **JSON** = JavaScript Object Notation
- Format léger et lisible
- Standard pour les API REST
- Spring Boot convertit automatiquement les objets Java en JSON

Annotation clé

`@RestController` = `@Controller` + `@ResponseBody`

Résumé du Chapitre 1

Ce que nous avons vu

- Architecture Full Stack
- SPA vs MPA
- API REST et méthodes HTTP
- Vue.js + TypeScript
- ref(), reactive(), computed()
- defineProps(), defineEmits()

Prochaine étape

Chapitre 2 : Communication Frontend-Backend

- Axios pour les appels HTTP
- Vue Router
- CRUD complet
- Gestion CORS

V + ♠ = Full Stack Power!

Questions ?

?

Des questions sur le chapitre 1 ?

N'hésitez pas à poser vos questions !