Chapter 3: The Face — Frontend Development in Vue.js

Introduction: What You Will Learn

The frontend is what users see. Without the backend, it's just a pretty picture. Without the frontend, users can't interact with your app.

By the end of this chapter, you will understand:

1. Why we use **Vue.js** (and why the CDN version)
2. The difference between **Imperative** (Vanilla JS) and **Declarative** (Vue) programming
3. How **Reactivity** works
4. How code polling simulates real-time updates

Part 1: The Three Pillars (with a Vue Twist)

HTML + Vue Directives

HTML defines the structure. With Vue, we sprinkle "magic attributes" (directives) into standard HTML.

```
<div id="app">
    <div v-if="!user.identifier">
        <!-- Login Form -->
        <input v-model="loginName" placeholder="Username">
        <button @click="doLogin">Login</button>
    </div>
</div>
```

Key Concepts:

- `v-if`: "Only show this element IF the condition is true."
- `v-model`: "Link this input box to a variable. If one changes, update the other."
- `@click`: "When clicked, run this function."

Javascript: The Vue Instance

Instead of writing scattered functions, we create one big object that controls the app.

```
const { createApp } = Vue;

createApp({
    data() {
        return {
            loginName: '',
            user: { identifier: null },
            messages: []
        }
    },
    methods: {
        async doLogin() {
            // ... logic to call API
        }
    }
}).mount('#app');
```

Part 2: Reactivity — The Magic

The Old Way (Vanilla JS / Imperative)

In standard JavaScript (like the inspiration PDF example), you have to manage the DOM yourself:

```
// Manually finding and updating
const div = document.getElementById("msg-box");
div.innerHTML = "New Message";
div.classList.add("highlight");
```

This is error-prone. If you forget to update one place, your UI is broken.

The Vue Way (Declarative)

You simply change the **Data**. Vue updates the **DOM**.

```
// In Javascript:
this.messages.push("New Message");

// In HTML:
// <div v-for="msg in messages">{{ msg }}</div>
```

Vue automatically detects the array changed and adds a new `<div>`. You never touch `document.getElementById`.

Oral Exam Defense:

> *"I used Vue.js because it allows for **Declarative Rendering**. I define what the UI should look like based on the state, and Vue handles the complex DOM manipulation to make it happen. This reduces bugs compared to manually selecting and updating elements."*

Part 3: The API Wrapper (`request.js`)

We use a helper file to make talking to our Backend easier. We use **Axios** (a library) instead of `fetch`.

Why Axios?

1. It automatically converts JSON (no need for `.json()`).
2. It throws errors automatically on bad status codes (400/500).

```
// request.js
const api = {
    async sendMessage(userId, conversationId, content) {
        try {
            const response = await axios.post(
                `${API_BASE}/conversations/${conversationId}/messages`,
                { content: content },
                { headers: { 'Authorization': `Bearer ${userId}` } }
            );
            return response.data;
        } catch (e) {
            throw e;
        }
    }
};
```

Part 4: Polling — Simulating Real-Time

The Problem

HTTP is **One-Way**. The Client asks, the Server answers. The Server *cannot* shout "Hey! New message!" to the Client.

The Solution: Polling

Every 2 seconds, we ask: "Anything new?"

```
startPolling() {
    // 1. Load immediately
    this.refreshConversations();

    // 2. Set strict timer
    this.pollInterval = setInterval(() => {
        this.refreshConversations();
        if (this.activeConv) {
            this.refreshMessages();
        }
    }, 2000); // 2000ms = 2 seconds
}
```

Trade-offs

- **Pros:** Extremely simple. Works on every server.
- **Cons:** Wastes battery (network requests even if no new messages). Delays (up to 2s lag).

Oral Exam Defense:

> *"I implemented Polling instead of WebSockets to keep the architecture simple and stateless. While inefficient for millions of users, for this project it provides a 'real-time enough' experience without the complexity of managing persistent WebSocket connections."*

Part 5: State Management

We use Vue's `data()` object as our **Single Source of Truth**.

```
data() {
    return {
        user: { ... },      // Who am I?
        conversations: [],  // My list of chats
        activeConv: null,   // What am I looking at?
        messages: []        // Messages in the active chat
    }
}
```

- If `activeConv` is `null`, the HTML shows "Select a conversation".
- If we set `activeConv = someChat`, the HTML automatically switches to the chat view.

This "State Machine" approach makes the logic very easy to follow.

Summary Checklist

Before moving to Chapter 4, ensure you understand:

- [ ] Why we use `v-if` and `v-for`
- [ ] How changing `this.data` updates the screen
- [ ] Why `axios` is easier than `fetch`
- [ ] How `setInterval` creates the polling loop
- [ ] Why we send the `Authorization` header in every request

Oral Exam Questions

Q: Why Vue.js (CDN) instead of React or Angular? "Vue's CDN build allows us to write a Single Page Application directly in one HTML file without a build step (No Webpack, No Node.js). This fits the requirement for a lightweight, dependency-free frontend."

Q: Explain how the chat updates when a new message arrives. "The `setInterval` timer triggers `refreshMessages()`. This functions calls the API. When the API returns new data, we assign it to `this.messages`. Vue's reactivity system detects this change and updates the DOM to display the new message bubbles."