Fingerprinting

Problem Statement

GPS-based positioning is **unreliable indoors** due to signal interference, making it difficult to precisely locate a device (e.g., a smartphone).

Proposed Solution: Fingerprinting

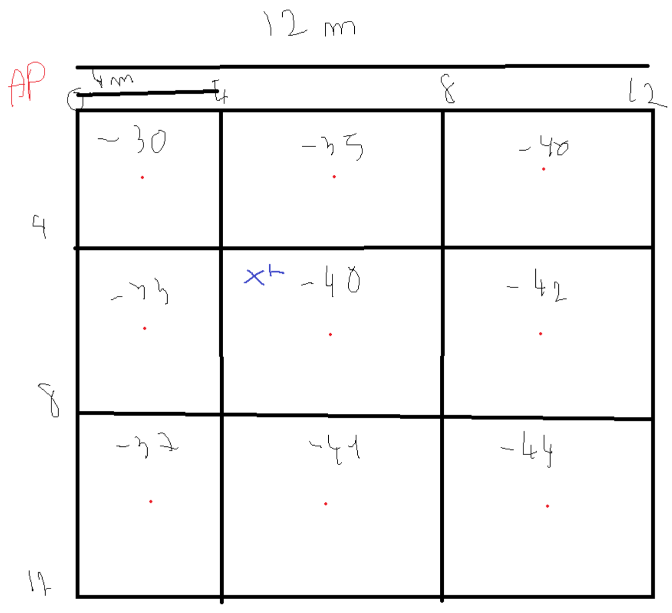
Fingerprinting is a **positioning technique** that estimates the location of a device by comparing its **real-time signal strength (RSSI)** to a **pre-recorded database** of signal measurements at known locations.

How Fingerprinting Works

1. **Identify the k-nearest reference points** (cells) with the most similar signal strengths.
2. **Compute a weighted barycentric position** based on these neighbors.
3. **Estimate the device's final position** using the weighted coordinates.

This approach enhances accuracy by leveraging previously mapped signal patterns.

Theoretical of fingerprinting



grille de 12x12m
chaque carré 4mx4m

chaque centre de carré a une coordonnée (x,y,z?) et un RSSI

tel quelque part dans la grille -> trouver les RSSI les plus semblables (3 k)

faire le barycentric pour calculer la position du tel

Proposed Idea

The concept involves creating a **12m × 12m grid**, composed of smaller squares of **4m × 4m**.

- An **Access Point (AP)** is placed at the **top-left corner** of the grid (**0,0,0**).

- The **signal strength (RSSI)** decreases as the distance from the AP increases.
- Each square contains a **known reference point** with its **coordinates (x, y, z) and RSSI value**.
- A **mobile RSSI value** is then introduced, and the **device's position is estimated** using the **K-closest neighbors** algorithm.

Potential Issues

- **Initial setup**: Requires configuring all reference points in the grid and collecting RSSI values.
- **Environmental factors**: Walls, furniture, and people can affect the RSSI readings.
- **Large environments**: Higher accuracy requires a denser grid with more reference points.

Advantages

✔ **High indoor positioning accuracy**.
✔ **Consistent results** in environments that do not change frequently.
✔ **Works effectively in multi-floor buildings**.

Explanation of the Technologies Used

Our application was developed using Vue.js, a progressive JavaScript framework that enables the creation of interactive and modular user interfaces. Vue.js facilitates component management and communication, allowing us to structure our application in a clear and efficient way.

For 3D rendering, we used Three.js, a JavaScript library that simplifies the creation and manipulation of 3D scenes in the browser. Three.js is built on WebGL (Web Graphics Library), a low-level graphics API that leverages GPU acceleration for rendering 2D and 3D graphics in web browsers. WebGL provides optimized performance by using the underlying OpenGL ES, enabling the rendering of complex objects with advanced lighting and texturing computations. Thanks to Three.js, we were able to create and manipulate 3D objects, manage the camera, and add dynamic interactions.

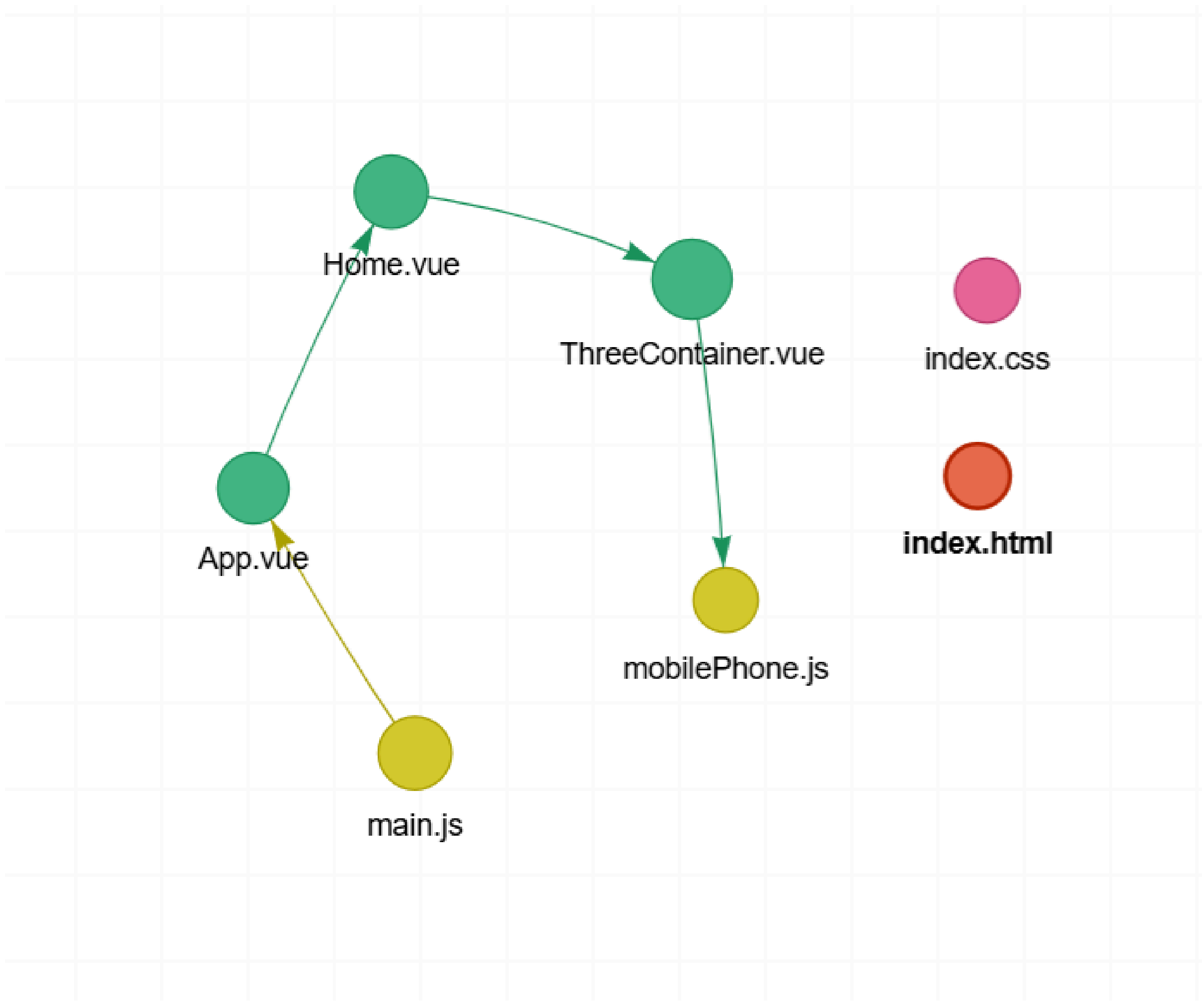
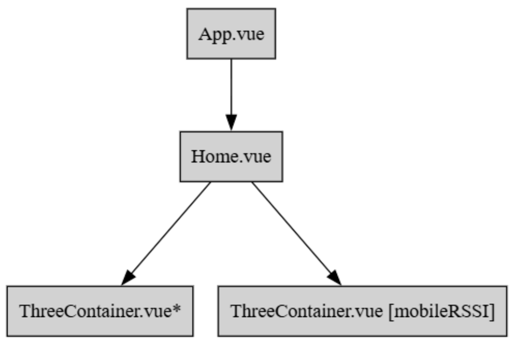


Diagram of the app files

App diagram Explanation



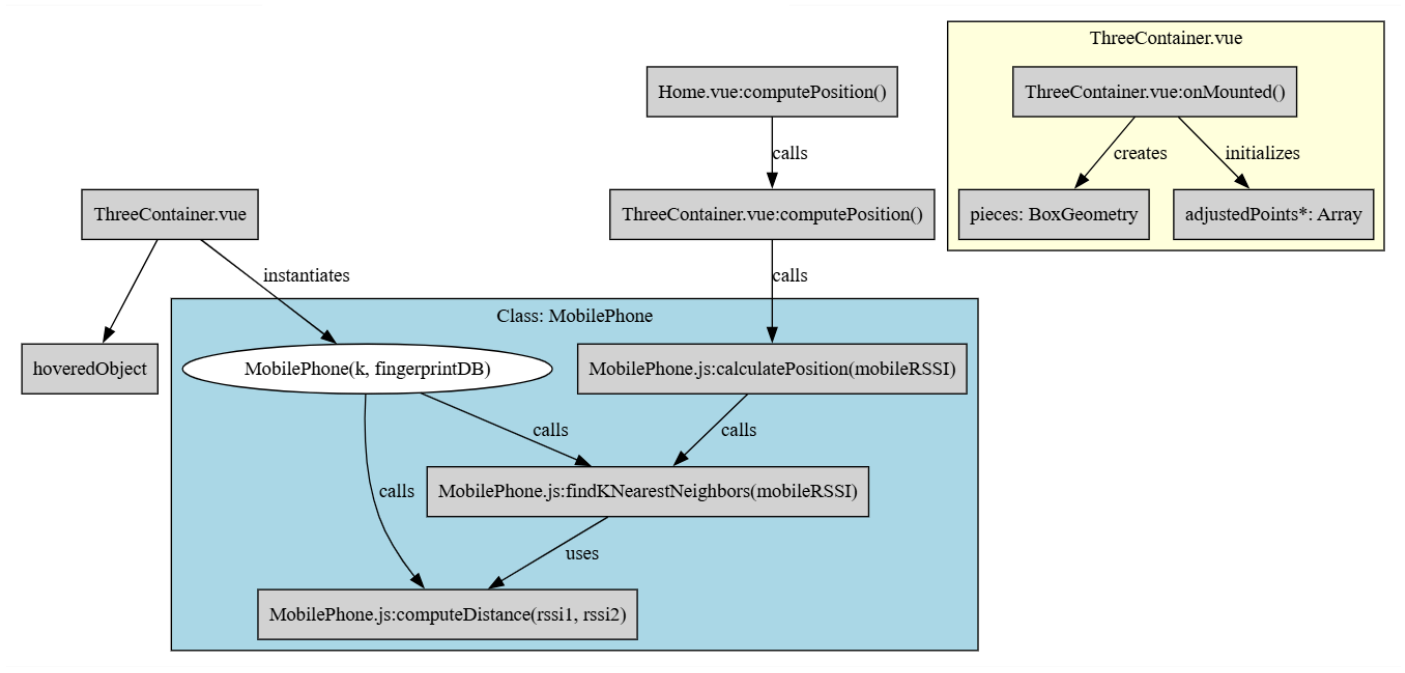

ThreeContainer.vue
ThreeContainer.vue:onMounted()
creates
initializes
pieces: BoxGeometry
adjustedPoints*: Array
Home.vue:computePosition()
calls
ThreeContainer.vue:computePosition()
calls
ThreeContainer.vue
instantiates
hoveredObject
Class: MobilePhone
MobilePhone(k, fingerprintDB)
MobilePhone.js:calculatePosition(mobileRSSI)
calls
calls
calls
MobilePhone.js:findKNearestNeighbors(mobileRSSI)
uses
MobilePhone.js:computeDistance(rssi1, rssi2)

Diagram of the web app

The diagram illustrates the overall architecture of our application and the interactions between different components and classes. Vue.js is represented by the `App.vue`, `Home.vue`, and `ThreeContainer.vue` files, where `Home.vue` integrates `ThreeContainer.vue`, which is responsible for rendering the 3D scene.

The `MobilePhone` class plays a key role in location tracking using a k-nearest neighbors (k-NN)-based positioning algorithm. Its constructor takes two parameters: `k`, the number of neighbors to consider, and `fingerprintDB`, a database of reference points with RSSI values. Three main methods are defined:

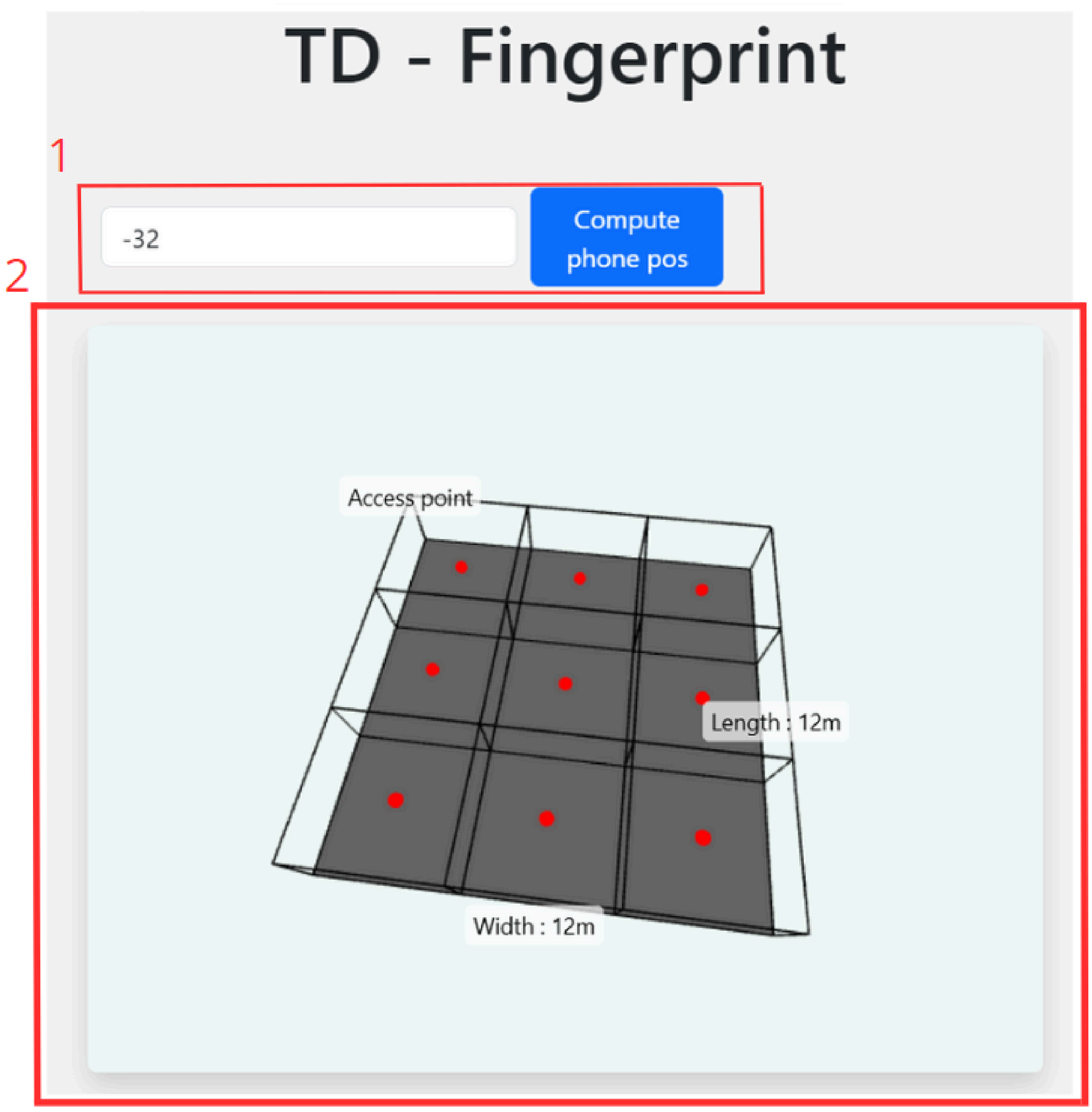
- `computeDistance(rssi1, rssi2)`, which calculates the distance between two RSSI values.

- `findKNearestNeighbors(mobileRSSI)`, which identifies the k closest points based on signal strength.
- `calculatePosition(mobileRSSI)`, which uses weighted interpolation to estimate the mobile's position.

In `ThreeContainer.vue`, the 3D scene is initialized using Three.js, and elements such as rooms (`pieces`) and reference points (`adjustedPoints`) are created. The `computePosition()` method in `ThreeContainer.vue` calls `calculatePosition()` from `MobilePhone.js` to determine the estimated position of the mobile device.

Together, these components enable an interactive 3D scene, where users can visualize the space, interact with objects, and obtain real-time information about the signal strength.

Design



1. As input, we can enter the phone's RSSI value. When clicking the **"Compute phone pos"** button, the function `calculatePosition(mobileRSSI)` is called to determine the phone's location.

2. The **3D view** displays the defined rooms, each measuring **4×4 meters**, along with the **position of the access point**. The **red points** represent fingerprinting reference points used for localization.