**Lưu ý CSS:**

- em: dựa vào font-size của phần tử cha gần nhất

- rem: luôn dựa vào font-size của phần tử html

Rem (Root em) là một đơn vị đo lường trong CSS:

1. Định nghĩa:

- Rem là viết tắt của "Root em"

- 1rem = kích thước font-size của phần tử gốc (root element, thường là thẻ <html>)

- Mặc định, 1rem = 16px (nếu không thay đổi font-size của html)

2. Đặc điểm:

- Tất cả giá trị rem đều dựa vào font-size của phần tử html

- Thay đổi font-size của html sẽ ảnh hưởng đến tất cả các giá trị rem trong trang

- Giúp tạo layout linh hoạt, dễ dàng điều chỉnh kích thước toàn bộ trang

**Lưu ý JS:**

Sự khác biệt này xuất phát từ cách QuerySnapshot được cấu trúc trong Firestore và cách các phương thức forEach và map hoạt động:

1. QuerySnapshot.forEach() :

- forEach là phương thức có sẵn của đối tượng QuerySnapshot trong Firestore

- Nó được thiết kế đặc biệt để lặp qua các documents trong QuerySnapshot

- Tự động truy cập từng document mà không cần thông qua thuộc tính .docs

2. QuerySnapshot.docs.map() :

- .docs là một thuộc tính của QuerySnapshot trả về một mảng các DocumentSnapshot

- map() là phương thức của Array trong JavaScript, không phải của QuerySnapshot

- Vì vậy để sử dụng map() , ta phải truy cập mảng documents thông qua .docs trước

Ví dụ:

// Cách 1: Sử dụng forEach (phương thức của QuerySnapshot)

querySnapshot.forEach((doc) => {

console.log(doc.data());

});

// Cách 2: Sử dụng map (phương thức của Array)

querySnapshot.docs.map((doc) => {

console.log(doc.data());

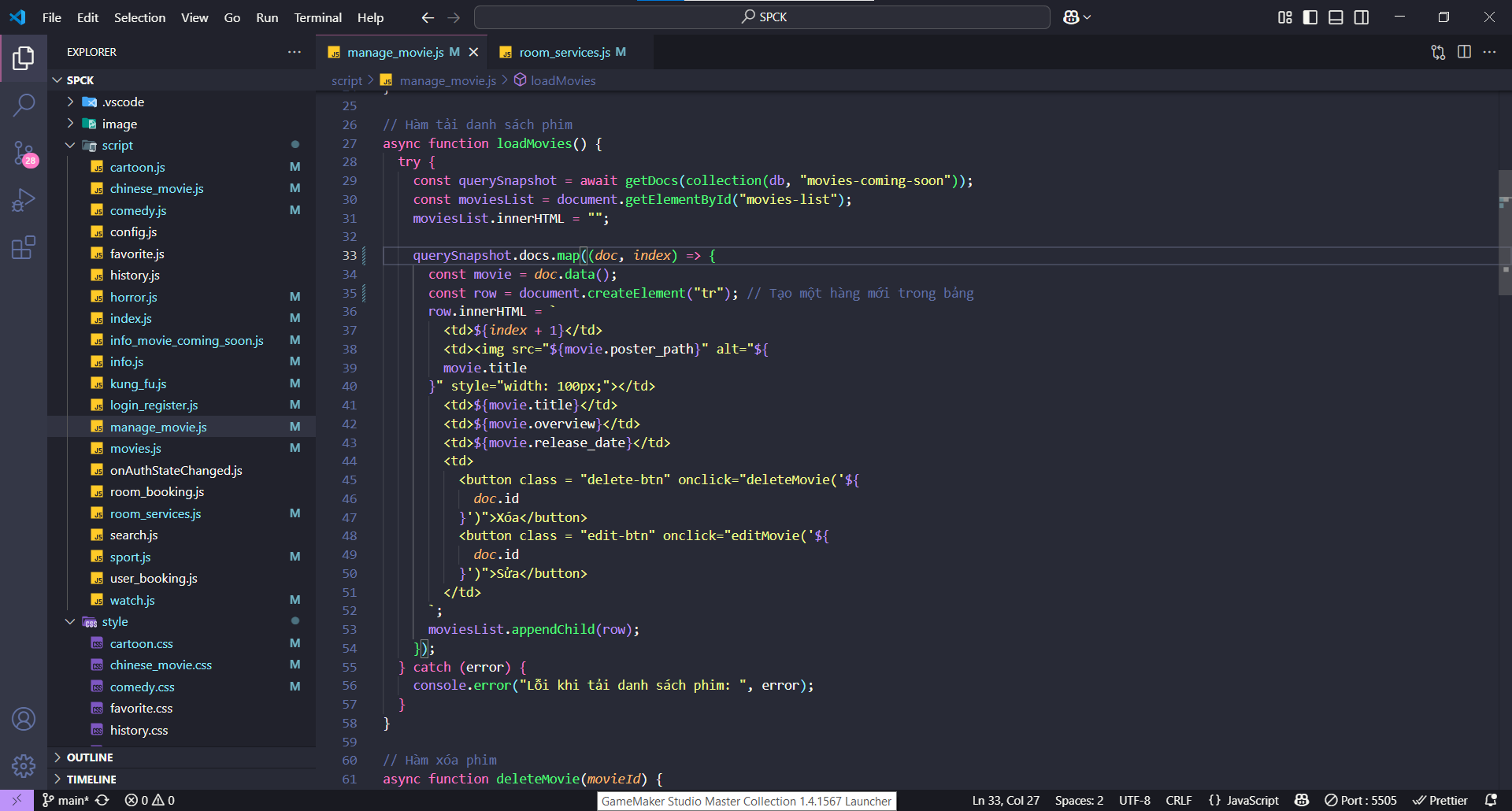
});

Lưu ý thêm:

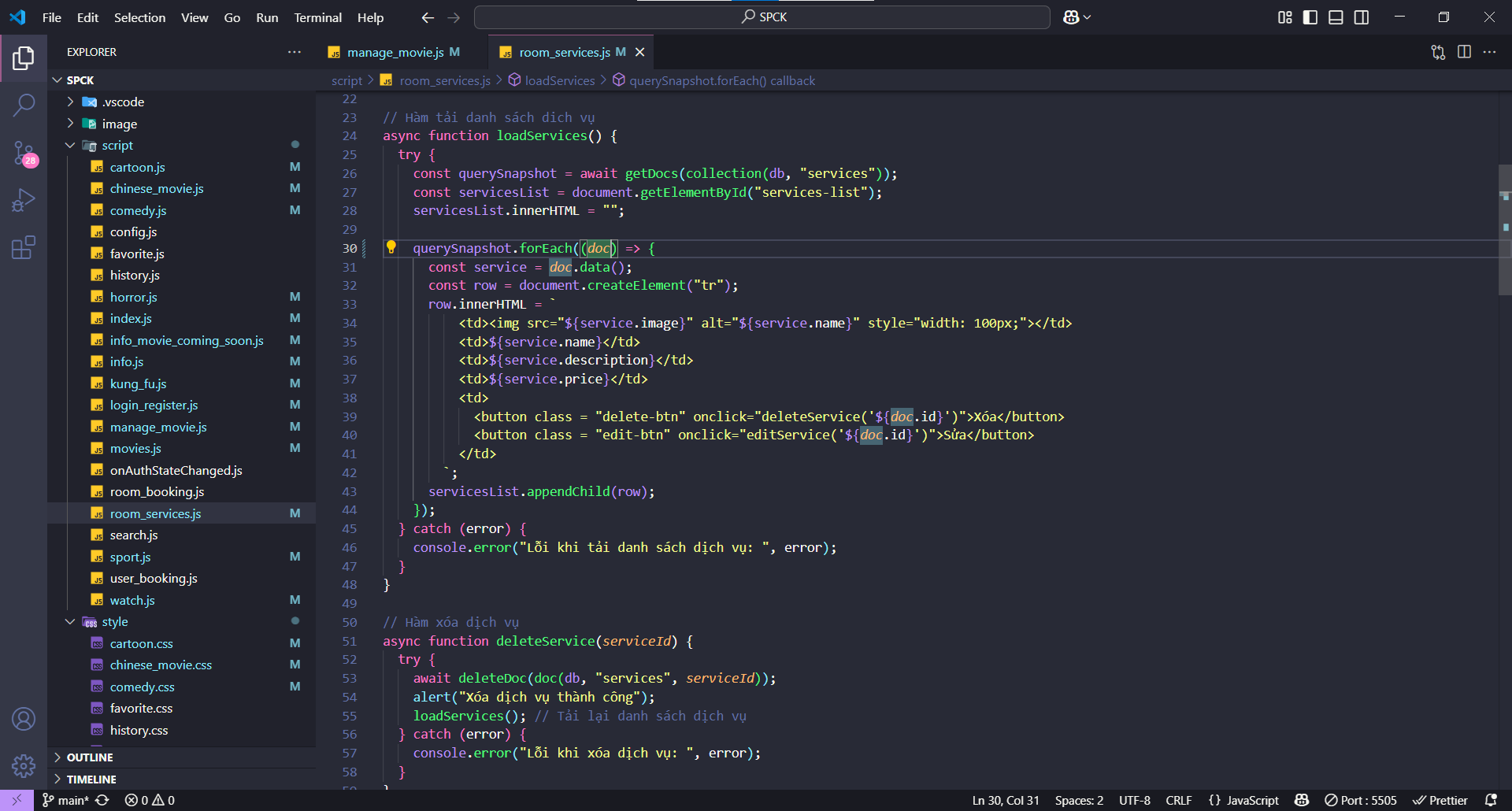
- forEach thường được sử dụng khi bạn chỉ muốn lặp qua và thực hiện một hành động nào đó

- map thường được sử dụng khi bạn muốn tạo một mảng mới từ kết quả của việc xử lý mỗi document

Tuy nhiên trong trường hợp cùng hiển thị dữ liệu lên table của manage\_movie và room\_services lại có sự khác nhau:



TH1: loadMovies() trong file manage\_movie.js querySnapshot cần sử dụng map để có biến đếm index để hiển thị số thứ tự



TH2: loadServices trong file room\_services thì querySnapshot chỉ cần sử dụng forEach() để lặp qua và hiển thị dữ liệu không cần biến đếm index để hiển thị số thứ tự

Lưu ý về firebase:

1. getDoc :

- Dùng để lấy một document (tài liệu) duy nhất

- Cần truyền vào tham số là reference đến một document cụ thể

- Trả về một document snapshot chứa dữ liệu của document đó

Ví dụ sử dụng getDoc :

const docRef = doc(db, "movies", "movie123");

const docSnap = await getDoc(docRef);

if (docSnap.exists()) {

  console.log("Document data:", docSnap.data());

}

2. getDocs :

- Dùng để lấy nhiều documents cùng lúc

- Có thể truyền vào collection reference hoặc query

- Trả về một query snapshot chứa nhiều documents

Ví dụ sử dụng getDocs :

// Lấy tất cả documents trong collection

const querySnapshot = await getDocs(collection(db, "movies"));

querySnapshot.forEach((*doc*) => {

  console.log(*doc*.id, " => ", *doc*.data());

});

// Hoặc lấy documents theo điều kiện query

const q = query(collection(db, "movies"), where("genre", "==", "action"));

const querySnapshot = await getDocs(q);

Sự khác biệt chính:

1. Số lượng documents:

   - getDoc : Lấy 1 document

   - getDocs : Lấy nhiều documents

2. Tham số đầu vào:

   - getDoc : Nhận document reference

   - getDocs : Nhận collection reference hoặc query

3. Kết quả trả về:

   - getDoc : DocumentSnapshot

   - getDocs : QuerySnapshot (chứa nhiều DocumentSnapshot)

4. Trường hợp sử dụng:

   - getDoc : Khi cần lấy thông tin chi tiết của một document cụ thể

   - getDocs : Khi cần lấy danh sách documents hoặc tìm kiếm theo điều kiện

3. Query trong Firestore là một công cụ mạnh mẽ để tìm kiếm và lọc dữ liệu. Hãy tìm hiểu chi tiết:

1. Query là gì?

- Là cách để xác định điều kiện tìm kiếm và lọc dữ liệu từ Firestore

- Cho phép bạn tìm các documents theo các tiêu chí cụ thể

- Có thể kết hợp nhiều điều kiện khác nhau

2. Các thành phần của query:

const q = query(

  collection(db, "movies"),      // Collection cần tìm kiếm

  where("genre", "==", "action"), // Điều kiện lọc

  orderBy("releaseDate", "desc"), // Sắp xếp kết quả

  limit(10)                      // Giới hạn số kết quả

);

3. Các phương thức phổ biến trong query:

a. where() : Lọc documents theo điều kiện

// Tìm phim có thể loại là action

where("genre", "==", "action")

// Tìm phim có rating lớn hơn 4

where("rating", ">", 4)

b. orderBy() : Sắp xếp kết quả

// Sắp xếp theo ngày tạo, mới nhất trước

orderBy("createAt", "desc")

// Sắp xếp theo tên

orderBy("name", "asc")

c. limit() : Giới hạn số lượng kết quả

// Chỉ lấy 10 kết quả

limit(10)

4. Cách sử dụng query:

// Tạo query

const q = query(

  collection(db, "movies"),

  where("genre", "==", "action"),

  orderBy("rating", "desc"),

  limit(5)

);

// Sử dụng với getDocs

const querySnapshot = await getDocs(q);

querySnapshot.forEach((*doc*) => {

  console.log(doc.data());

});

// Hoặc sử dụng với onSnapshot để lắng nghe thay đổi realtime

onSnapshot(q, (*snapshot*) => {

  snapshot.forEach((*doc*) => {

    console.log(doc.data());

  });

});

4. orderBy :

- Là một phương thức dùng để sắp xếp các documents trong một collection theo một hoặc nhiều trường

- Cú pháp cơ bản: orderBy("fieldName", "asc/desc")

  - fieldName : tên trường cần sắp xếp

  - asc : sắp xếp tăng dần (ascending)

  - desc : sắp xếp giảm dần (descending)

5. serverTimestamp() :

- Là một hàm đặc biệt của Firestore để tạo timestamp tại thời điểm document được lưu trên server

- Đảm bảo thời gian chính xác và đồng bộ giữa các thiết bị, không phụ thuộc vào thời gian local của client

- Thường dùng cho các trường như createdAt , updatedAt , lastLoginTime