Deployment Details for Microwave Radar Analysis Model Interaction

July 25, 2025

Overview

This document outlines the implementation details of the Microwave Radar Analysis Model Interaction System. The system is designed for modular, extensible radar signal and electronic warfare analysis. It consists of:

- **Frontend:** An Angular-based dashboard that enables users to configure input parameters, launch simulations, and visualize results using a 3D CesiumJS map.
- **Backend Microservices:** Spring Boot–based services, each responsible for handling domain-specific processing:
    - **AuthenticationService:** Manages user login.
    - **AnalysisManagerService:** Receives input from the frontend and forwards it to the appropriate model service.
    - **AbsorptionLossService:** Computes atmospheric absorption loss based on terrain and weather data.
    - **ECMModelService:** Estimates ECM (Electronic Countermeasure) effectiveness such as jamming strength.
    - **RadarModelService:** Calculates radar performance metrics like received signal, SIR, and probability of detection.
- **Database:** A centralized PostgreSQL 14 instance stores user credentials for authentication, input parameters for simulation scenarios, and the corresponding model outputs. It serves as persistent storage for simulation data across the system.

Backend

Root (Backend/)

The backend contains the backend microservices powering the radar modeling platform. It includes three domain-specific computational services, a user authentication module, and an analysis manager that orchestrates cross-service operations.

```
Backend/
├── AbsorptionLossService/
├── AnalysisManagerService/
├── AuthenticationService/
├── ECMModelService/
├── RadarModelService/
├── README.md
└── settings.xml
```

Service Modules

1. RadarModelService/
   - Computes radar-specific outputs:
     - pd (Probability of Detection)
     - sreceiveRadar (Received Signal Strength)
     - siradar (Signal-to-Interference Ratio)
2. ECMModelService/
   - Computes ECM-specific output:
     - jra (Jammer Range Advantage)
3. AbsorptionLossService/
   - Computes atmospheric attenuation:
     - ldash (Total Atmospheric Loss)
4. AuthenticationService/
   - Handles user authentication
5. AnalysisManagerService/
   - Acts as middleware between the Angular frontend

Key Source Code Walkthrough

settings.xml

Needed to download maven dependences offline

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
https://maven.apache.org/xsd/settings-1.0.0.xsd">
<localRepository>./offline-m2</localRepository>
</settings>
```

AbsorptionLossService

```
AbsorptionLossService.
├── offline-m2
├── pom.xml
├── README.md
└── src
    └── main
        ├── java
        │   └── com
        │       └── xai
        │           └── absorptionloss
        │               ├── AbsorptionController.java
        │               ├── AbsorptionService.java
        │               ├── Application.java
        │               ├── entity
        │               │   ├── AtmosphericOutputs.java
        │               │   ├── AtmosphericParameters.java
        │               │   └── WeatherCondition.java
        │               └── repository
        │                   ├── AtmosphericOutputsRepository.java
        │                   └── AtmosphericParametersRepository.java
        └── resources
            └── application.properties
```

README.md

```
## AbsorptionLossService

### Purpose
```

This microservice calculates **atmospheric absorption loss** based on weather and terrain input parameters. It is consumed by the frontend dashboard for modeling radar propagation.

### Endpoint

http://localhost:8081/api/absorption/calculate (POST)

### Input

```json
{
  "rfr": ...,
  "sfr": ...,
  "visFog": ...,
  "freqR": ...,
  "freqJ": ...,
  "freqOp": ...,
  "latR": ...,
  "longR": ...,
  "heightR": ...,
  "latT": ...,
  "longT": ...,
  "heightT": ...,
  "latJ": ...,
  "longJ": ...,
  "heightJ": ...,
  "weatherConditions": [
    {
      "weatherType": ...,
      "wholeGlobe": false,
      "lat1": ..,
      "lat2": ...,
      "lat3": ...,
      "lat4": ...,
      "long1": ...,
      "long2": ...,
      "long3": ...,
      "long4": ...,
      "height": ...
    },
    {
      "weatherType": ...,
      "wholeGlobe": true,
      "lat1": null,
      "lat2": null,
      "lat3": null,
      "lat4": null,
      "long1": null,
      "long2": null,
      "long3": null,
      "long4": null,
      "height": ...
    }
  ]
}

```

### Field Descriptions

#### Main Parameters
- `rfr`: Rainfall rate (in mm/hr)
- `sfr`: Snowfall rate (unit depends on context; could be mm/hr or cm/hr)
- `visFog`: Visibility in fog (in meters)
- `freqR`: Receiver frequency (in GHz or MHz, depending on system)
- `freqJ`: Jammer frequency (in GHz or MHz)
- `freqOp`: Operating frequency used in absorption calculations (in GHz or MHz)

#### Receiver Location
- `latR`: Latitude of the receiver (in degrees)
- `longR`: Longitude of the receiver (in degrees)
- `heightR`: Height of the receiver (in meters)

#### Transmitter Location
- `latT`: Latitude of the transmitter (in degrees)
- `longT`: Longitude of the transmitter (in degrees)
- `heightT`: Height of the transmitter (in meters)

#### Jammer Location
- `latJ`: Latitude of the jammer (in degrees)
- `longJ`: Longitude of the jammer (in degrees)
- `heightJ`: Height of the jammer (in meters)

### Weather Conditions (List)
Each item in the `weatherConditions` array represents a different weather zone or condition.

- `weatherType`: Type of weather (e.g., `"Rain"`, `"Fog"`, `"Snow"`)

- `wholeGlobe`: Boolean indicating if the weather condition affects the entire globe
- `lat1` to `lat4`: Latitude corners defining a bounding box for the weather zone (in degrees)
- `long1` to `long4`: Longitude corners defining a bounding box for the weather zone (in degrees)
- `height`: Altitude of the weather zone (in meters)

If `wholeGlobe` is `true`, all `latX` and `longX` fields are expected to be `null`.

### Output

```json
{ "id".., "ldash": ... }
```

* `id`: Output record ID (auto-generated)
* `ldash`: Computed atmospheric absorption loss in dB

### File Roles & Flow

* **`AbsorptionLossController.java`**

  * Defines the `/api/absorption/calculate` POST endpoint.
  * Accepts validated JSON input and returns formatted output.
  * Calls `AbsorptionLossService`.

* **`AbsorptionLossService.java`**

  * Contains core logic to calculate `ldash` (absorption loss) based on input.
  * Delegates mathematical modeling logic to `AbsorptionLossModel`.

* **`AbsorptionLossModel.java`**

  * Contains the actual scientific formulas for computing loss based on the given atmospheric data.

* **`AbsorptionLossRequest.java`**

  * DTO class for deserializing incoming JSON data.
  * Includes fields for all required parameters.

* **`AbsorptionLossResponse.java`**

  * DTO for structuring the API response in the format `{ "absorptionOutput": { "ldash": value } }`.

### Database Storage

* **Table:** `atmospheric_parameters`
* **Stored Fields:**

  * `id` (Long, auto-generated)
  * `rfr` (double, required)
  * `sfr` (double, required)
  * `visFog` (double, required)
  * `freqR` (double, required)
  * `freqJ` (double, required)
  * `freqOp` (double, required)
  * `latR` (double, required)
  * `longR` (double, required)
  * `heightR` (double, required)
  * `latT` (double, required)
  * `longT` (double, required)
  * `heightT` (double, required)
  * `latJ` (double, required)
  * `longJ` (double, required)
  * `heightJ` (double, required)

---

* **Table:** `atmospheric_outputs`
* **Stored Fields:**

  * `id` (Long, auto-generated)
  * `lDash` (double, calculated)
  * `parameter_id` (Long, required, foreign key to `atmospheric_parameters.id`)

---

* **Table:** `weather_condition`
* **Stored Fields:**

  * `id` (Long, auto-generated)
  * `weatherType` (String, required)
  * `wholeGlobe` (boolean, required)
  * `lat1` (Double, optional)
  * `lat2` (Double, optional)
  * `lat3` (Double, optional)
  * `lat4` (Double, optional)

```
  * `long1` (Double, optional)
  * `long2` (Double, optional)
  * `long3` (Double, optional)
  * `long4` (Double, optional)
  * `height` (Double, optional)
  * `parameters_id` (Long, required, foreign key to `atmospheric_parameters.id`)

---

### Summary of Connections

```
AnalysisController <--> AbsorptionLossController <--> AbsorptionLossService <-->
AbsorptionLossModel
```

```

pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <!-- Basic project metadata -->
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.xai</groupId>                    <!-- Group identifier for your org
-->
    <artifactId>AbsorptionLossService</artifactId> <!-- Name of this microservice -->
    <version>0.0.1-SNAPSHOT</version>              <!-- Current project version -->

    <!-- Inherit Spring Boot defaults (dependency versions, plugin config) -->
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.1.0</version>
    </parent>

    <!-- Dependencies required for this microservice -->
    <dependencies>
        <!-- Provides embedded Tomcat and REST APIs -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <!-- Enables Spring Data JPA for DB operations -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>

        <!-- PostgreSQL JDBC driver -->
        <dependency>
            <groupId>org.postgresql</groupId>
            <artifactId>postgresql</artifactId>
            <version>42.6.0</version>
        </dependency>
    </dependencies>

    <!-- Build plugins configuration -->
    <build>
        <plugins>
            <!-- Spring Boot plugin for building and running app with `mvn
spring-boot:run` -->
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

Application.java

```
package com.xai.absorptionloss;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

AbsorptionController.java

```
package com.xai.absorptionloss;

import com.xai.absorptionloss.entity.AtmosphericParameters;
import com.xai.absorptionloss.entity.AtmosphericOutputs;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/absorption")
@CrossOrigin(origins = "http://localhost:4200")
public class AbsorptionController {

    @Autowired
    private AbsorptionService service;

    @PostMapping("/calculate")
    public AtmosphericOutputs calculate(@RequestBody AtmosphericParameters input) {
        return service.calculateAbsorption(input);
    }
}
```

AbsorptionService.java

```
package com.xai.absorptionloss;

import com.xai.absorptionloss.entity.AtmosphericOutputs;
import com.xai.absorptionloss.entity.AtmosphericParameters;
import com.xai.absorptionloss.entity.WeatherCondition;
import com.xai.absorptionloss.repository.AtmosphericOutputsRepository;
import com.xai.absorptionloss.repository.AtmosphericParametersRepository;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
public class AbsorptionService {

    @Autowired
    private AtmosphericParametersRepository inputRepo;

    @Autowired
    private AtmosphericOutputsRepository outputRepo;

    // Main method that handles the full loss calculation
    public AtmosphericOutputs calculateAbsorption(AtmosphericParameters input) {
```

```java
    inputRepo.save(input); // Persist input for logging/auditing

    // Extract transmitter location
    double lat2 = input.getLatT();
    double lon2 = input.getLongT();
    double h2 = input.getHeightT();

    // Calculate total distance between receiver and transmitter (3D)
    double distance = calculateDistance3D(
        input.getLatR(), input.getLongR(), input.getHeightR(),
        lat2, lon2, h2
    );

    // Compute atmospheric absorption coefficient (α)
    double alpha = calculateAlpha(input);

    // Total absorption loss (dB/km * km)
    double lDashDb = alpha * (distance / 1000.0);

    // Package and save output
    AtmosphericOutputs output = new AtmosphericOutputs();
    output.setLDash(lDashDb);
    output.setParameters(input);
    return outputRepo.save(output);
}

// Calculates absorption coefficient α based on weather conditions and frequency
private double calculateAlpha(AtmosphericParameters input) {
    List<WeatherCondition> conditions = input.getWeatherConditions();
    if (conditions == null || conditions.isEmpty()) {
        return 0.01; // Default: clear weather (light loss)
    }

    double totalAlpha = 0.0;
    for (WeatherCondition condition : conditions) {
        String type = condition.getWeatherType().toLowerCase();
        switch (type) {
            case "rain":
                // Empirical formula for rain attenuation
                totalAlpha += 0.0101 * Math.pow(input.getFreqOp(), 1.276) *
                              Math.pow(input.getRfr(), 0.9991);
                break;
            case "snow":
                // Empirical formula for snow attenuation
                totalAlpha += 0.0001 * Math.pow(input.getFreqOp(), 2) *
                              input.getSfr() + 0.02;
                break;
            case "fog":
                    // Based on visibility and frequency (if visibility > 0)
                    if (input.getVisFog() > 0) {
                        totalAlpha += (0.05 * Math.pow(input.getFreqOp(), 2)) /
                                      (input.getVisFog() / 1000.0); // visibility in
km
                    }
                    break;
                case "clear":
                default:
                    totalAlpha += 0.01; // baseline value
                    break;
            }
        }
        return totalAlpha;
    }

    // Calculates the haversine (great-circle) distance between two lat/lon points in
meters
    private double haversine(double lat1, double lon1, double lat2, double lon2) {
        double R = 6371e3; // Earth's radius in meters
        double dLat = Math.toRadians(lat2 - lat1);
        double dLon = Math.toRadians(lon2 - lon1);
        double a = Math.sin(dLat / 2) * Math.sin(dLat / 2) +
                   Math.cos(Math.toRadians(lat1)) * Math.cos(Math.toRadians(lat2)) *
                   Math.sin(dLon / 2) * Math.sin(dLon / 2);
        return 2 * R * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
    }

    // Combines 2D surface distance and altitude difference to compute full 3D
distance
    private double calculateDistance3D(double lat1, double lon1, double h1,
                                       double lat2, double lon2, double h2) {
        double horizontal = haversine(lat1, lon1, lat2, lon2); // surface distance in
meters
        return Math.sqrt(horizontal * horizontal + Math.pow(h1 - h2, 2)); //
Pythagorean 3D
    }
}
```

AtmosphericOutputs.java

```
package com.xai.absorptionloss.entity;

import jakarta.persistence.*;

/**
 * Entity representing the output of the atmospheric absorption loss model.
 * Stores the final result (lDash) and links it to the input parameters.
 */
@Entity
public class AtmosphericOutputs {

    // Primary key for AtmosphericOutputs table (auto-generated)
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    // The computed absorption loss value (L′ or lDash)
    private double lDash;

    // One-to-one relation with the input parameter set
    @OneToOne
    @JoinColumn(name = "parameter_id") // Foreign key to AtmosphericParameters
    private AtmosphericParameters parameters;

    // --- Getters and Setters ---

    public Long getId() { return id; }

    public void setId(Long id) { this.id = id; }

    public double getLDash() { return lDash; }

    public void setLDash(double lDash) { this.lDash = lDash; }

    public AtmosphericParameters getParameters() { return parameters; }

    public void setParameters(AtmosphericParameters parameters) { this.parameters =
parameters; }
}
```

AtmosphericParameters.java

```
package com.xai.absorptionloss.entity;

import com.fasterxml.jackson.annotation.JsonManagedReference;
import jakarta.persistence.*;
import java.util.List;

@Entity
public class AtmosphericParameters {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id; // Primary key
    // --- Weather-related coefficients and frequency values ---
    private double rfr;     // Rainfall rate (mm/hr)
    private double sfr;     // Snowfall rate
    private double visFog;  // Fog visibility (meters)
    private double freqR;   // Receiver frequency
    private double freqJ;   // Jammer frequency
    private double freqOp;  // Operating frequency used for absorption calculations

    // --- Receiver location ---
    private double latR;
    private double longR;
    private double heightR;

    // --- Transmitter location ---
    private double latT;
    private double longT;
    private double heightT;

    // --- Jammer location ---
    private double latJ;
    private double longJ;
    private double heightJ;

    // One-to-many relationship: each parameter set can have multiple weather
conditions
    @OneToMany(mappedBy = "parameters", cascade = CascadeType.ALL, orphanRemoval =
true)
    @JsonManagedReference
    private List<WeatherCondition> weatherConditions;

    // --- Getters & Setters ---

    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }

    public double getRfr() {
        return rfr;
    }
    public void setRfr(double rfr) {
        this.rfr = rfr;
    }

    public double getSfr() {
        return sfr;
    }
    public void setSfr(double sfr) {
        this.sfr = sfr;
    }

    public double getVisFog() {
        return visFog;
    }
    public void setVisFog(double visFog) {
        this.visFog = visFog;
    }

    public double getFreqR() {
        return freqR;
    }
    public void setFreqR(double freqR) {
        this.freqR = freqR;
    }

    public double getFreqJ() {
        return freqJ;
    }
    public void setFreqJ(double freqJ) {
        this.freqJ = freqJ;
    }

    public double getFreqOp() {
        return freqOp;
    }
    public void setFreqOp(double freqOp) {
        this.freqOp = freqOp;
    }

    public double getLatR() {
        return latR;
    }
    public void setLatR(double latR) {
        this.latR = latR;
    }

    public double getLongR() {
        return longR;
    }
    public void setLongR(double longR) {
        this.longR = longR;
    }
```

```
public double getHeightR() {
    return heightR;
}
public void setHeightR(double heightR) {
    this.heightR = heightR;
}

public double getLatT() {
    return latT;
}
public void setLatT(double latT) {
    this.latT = latT;
}

public double getLongT() {
    return longT;
}
public void setLongT(double longT) {
    this.longT = longT;
}

public double getHeightT() {
    return heightT;
}
public void setHeightT(double heightT) {
    this.heightT = heightT;
}

public double getLatJ() {
    return latJ;
}
public void setLatJ(double latJ) {
    this.latJ = latJ;
}

public double getLongJ() {
    return longJ;
}
public void setLongJ(double longJ) {
    this.longJ = longJ;
}

public double getHeightJ() {
    return heightJ;
}
public void setHeightJ(double heightJ) {
    this.heightJ = heightJ;
}
```

```java
    public List<WeatherCondition> getWeatherConditions() {
        return weatherConditions;
    }

    public void setWeatherConditions(List<WeatherCondition> weatherConditions) {
        this.weatherConditions = weatherConditions;

        // Ensure bidirectional linkage for JPA cascade persist
        if (weatherConditions != null) {
            for (WeatherCondition wc : weatherConditions) {
                wc.setParameters(this);
            }
        }
    }
}
```

WeatherCondition.java

```java
package com.xai.absorptionloss.entity;

import com.fasterxml.jackson.annotation.JsonBackReference;
import jakarta.persistence.*;

@Entity
public class WeatherCondition {

    // Primary key with auto-increment
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String weatherType;   // e.g., "Rain", "Fog"
    private boolean wholeGlobe;   // true if weather affects entire globe

    // Geographical bounding box coordinates (lat/long corners)
    private Double lat1;
    private Double lat2;
    private Double lat3;
    private Double lat4;
    private Double long1;
    private Double long2;
    private Double long3;
    private Double long4;

    private Double height;  // Altitude in meters (or km)
```

```
// Many weather conditions can belong to one AtmosphericParameters
@ManyToOne
@JoinColumn(name = "parameters_id")
@JsonBackReference  // Prevents recursive serialization
private AtmosphericParameters parameters;

// Getters and setters
public Long getId() {
    return id;
}
public void setId(Long id) {
    this.id = id;
}

public String getWeatherType() {
    return weatherType;
}
public void setWeatherType(String weatherType) {
    this.weatherType = weatherType;
}

public boolean isWholeGlobe() {
    return wholeGlobe;
}
public void setWholeGlobe(boolean wholeGlobe) {
    this.wholeGlobe = wholeGlobe;
}

public Double getlat1() {
    return lat1;
}
public void setlat1(Double lat1) {
    this.lat1 = lat1;
}

public Double getlat2() {
    return lat2;
}
public void setlat2(Double lat2) {
    this.lat2 = lat2;
}

public Double getlat3() {
    return lat3;
}
public void setlat3(Double lat3) {
    this.lat3 = lat3;
}
```

```java
public Double getlat4() {
    return lat4;
}
public void setlat4(Double lat4) {
    this.lat4 = lat4;
}

public Double getlong1() {
    return long1;
}
public void setlong1(Double long1) {
    this.long1 = long1;
}

public Double getlong2() {
    return long2;
}
public void setlong2(Double long2) {
    this.long2 = long2;
}

public Double getlong3() {
    return long3;
}
public void setlong3(Double long3) {
    this.long3 = long3;
}

public Double getlong4() {
    return long4;
}
public void setlong4(Double long4) {
    this.long4 = long4;
}

public Double getheight() {
    return height;
}
public void setheight(Double height) {
    this.height = height;
}

public AtmosphericParameters getParameters() {
    return parameters;
}
public void setParameters(AtmosphericParameters parameters) {
    this.parameters = parameters;
    }
}
```

AtmosphericOutputsRepository.java

```
package com.xai.absorptionloss.repository;

import com.xai.absorptionloss.entity.AtmosphericOutputs;
import org.springframework.data.jpa.repository.JpaRepository;

public interface AtmosphericOutputsRepository extends
JpaRepository<AtmosphericOutputs, Long> {}
```

AtmosphericParametersRepository.java

```
package com.xai.absorptionloss.repository;

import com.xai.absorptionloss.entity.AtmosphericParameters;
import org.springframework.data.jpa.repository.JpaRepository;

public interface AtmosphericParametersRepository extends
JpaRepository<AtmosphericParameters, Long> {}
```

ECMModelService.

```
ECMModelService
├── pom.xml
├── README.md
└── src
    └── main
        ├── java
        │   └── com
        │       └── xai
        │           └── ecmmodel
        │               ├── Application.java
        │               ├── EcmController.java
        │               ├── EcmService.java
        │               ├── entity
        │           │   ├── ECMOutputs.java
        │           │   └── ECMParameters.java
        │           └── repository
        │               ├── ECMOutputsRepository.java
        │               └── ECMParametersRepository.java
        └── resources
            └── application.properties
```

README.md

~~~
## ECMModelService

### Purpose

This microservice calculates **Jamming-to-Radar Advantage (J/R Advantage)** based on
various ECM (Electronic Countermeasure) parameters. It is used by the frontend
dashboard to evaluate ECM effectiveness in radar environments.

---

### Endpoint

`http://localhost:8082/api/ecm/calculate` (POST)

---

### Input

```json
{
  "pJ": ...,
  "gJ": ...,
  "lJ": ...,
  "azimuthJ": ...,
  "elevationJ": ...,
  "beamwidthAzJ": ...,
  "beamwidthElJ": ...,
  "bJ": ...,
  "freqJ": ...,
  "latJ": ...,
  "longJ": ...,
  "heightJ": ...,
  "jammerType": ...,
  "latR": ...,
  "longR": ...,
~~~

~~~
  "heightR": ...,
  "bR": ...,
  "freqR": ...
}
```

---

### Field Descriptions

#### Jammer Parameters

* `pJ`: Power of jammer (watts)
* `gJ`: Gain of jammer antenna (dB)
* `lJ`: Loss in jammer system (dB)
* `azimuthJ`, `elevationJ`: Orientation angles (degrees)
* `beamwidthAzJ`, `beamwidthElJ`: Beamwidth of jammer in azimuth/elevation (degrees)
* `bJ`: Bandwidth of jammer (Hz)
* `freqJ`: Jammer frequency (GHz)
* `latJ`, `longJ`, `heightJ`: Jammer location (degrees/meters)
* `jammerType`: Type of jammer (e.g., "noise", "DRFM")

#### Radar Parameters

* `latR`, `longR`, `heightR`: Radar location (degrees/meters)
* `bR`: Radar receiver bandwidth (Hz)
* `freqR`: Radar frequency (GHz)

---

### Output

```json
{
  "id": ...,
  "jRa": ...
}
```
* `id`: Output record ID (auto-generated)
* `jRa`: Computed Jamming-to-Radar Advantage in dB

---

### File Roles & Flow

* **`Application.java`**
  Entry point for the Spring Boot application.
~~~

```
* **`EcmController.java`**
  REST controller exposing the `/api/ecm/calculate` POST endpoint.
  It receives ECM parameters and returns the calculated output.

* **`EcmService.java`**
  Contains the main logic to:

  * Store input parameters
  * Calculate 3D distance
  * Apply signal loss and gain formulas
  * Save and return the result

* **`ECMParameters.java`**
  JPA entity class that maps to the input parameters table.

* **`ECMOutputs.java`**
  JPA entity class that maps to the output results table (J/R Advantage).

* **`ECMParametersRepository.java` & `ECMOutputsRepository.java`**
  Spring Data JPA repositories for database operations.

---

### Database Storage

* **Table:** `ecmparameters`
* **Stored Fields:**

  * `id` (Long, auto-generated)
  * `pJ` (double, required)
  * `gJ` (double, required)
  * `lJ` (double, required)
  * `azimuthJ` (double, required)
  * `elevationJ` (double, required)
  * `beamwidthAzJ` (double, required)
  * `beamwidthElJ` (double, required)
  * `bJ` (double, required)
  * `freqJ` (double, required)
  * `latJ` (double, required)
  * `longJ` (double, required)
  * `heightJ` (double, required)
  * `jammerType` (String, required)
  * `latR` (double, required)
  * `longR` (double, required)
  * `heightR` (double, required)
  * `bR` (double, required)
  * `freqR` (double, required)
```

~~~
---

* **Table:** `ecmoutputs`
* **Stored Fields:**

  * `id` (Long, auto-generated)
  * `jRa` (double, calculated)
  * `parameter_id` (Long, required, foreign key to `ecmparameters.id`)

---

### Summary of Connections

```
AnalysisController <--> EcmController <--> EcmService <--> ECMModel
```

---
~~~

pom.xml

```xml
<!-- Maven project configuration -->
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <!-- Maven model version -->
    <modelVersion>4.0.0</modelVersion>

    <!-- Project group ID -->
    <groupId>com.xai</groupId>

    <!-- Project artifact ID -->
    <artifactId>ECMModelService</artifactId>

    <!-- Project version -->
    <version>0.0.1-SNAPSHOT</version>

    <!-- Inherit defaults from Spring Boot parent -->
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.1.0</version>
    </parent>

    <!-- Project dependencies -->
```

```
    <dependencies>
        <!-- Spring Boot web starter (for building web apps and REST APIs) -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <!-- Spring Boot JPA starter (for database access with JPA) -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>

        <!-- PostgreSQL JDBC driver -->
        <dependency>
            <groupId>org.postgresql</groupId>
            <artifactId>postgresql</artifactId>
            <version>42.6.0</version>
        </dependency>
    </dependencies>

    <!-- Build settings -->
    <build>
        <plugins>
            <!-- Spring Boot Maven plugin (for building and running Spring Boot apps)
-->
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

Application.java

```
package com.xai.ecmmodel;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

EcmController.java

```java
package com.xai.ecmmodel;

import com.xai.ecmmodel.entity.ECMParameters;
import com.xai.ecmmodel.entity.ECMOutputs;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/ecm")
@CrossOrigin(origins = "http://localhost:4200")
public class EcmController {

    @Autowired
    private EcmService service;

    @PostMapping("/calculate")
    public ECMOutputs calculate(@RequestBody ECMParameters input) {
        return service.calculateEcm(input);
    }
}
```

EcmService.java

```java
package com.xai.ecmmodel;

import com.xai.ecmmodel.entity.ECMOutputs;
import com.xai.ecmmodel.entity.ECMParameters;
import com.xai.ecmmodel.repository.ECMOutputsRepository;
import com.xai.ecmmodel.repository.ECMParametersRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service // Marks this class as a Spring service component
public class EcmService {

    private static final double C = 3e8; // Speed of light in meters/second

    @Autowired
    private ECMParametersRepository inputRepo; // Repository to save input parameters

    @Autowired
    private ECMOutputsRepository outputRepo; // Repository to save calculation output
    // Main method to perform ECM calculations
    public ECMOutputs calculateEcm(ECMParameters input) {
        // Save the input parameters to the database
        inputRepo.save(input);

        // Calculate 3D distance between jammer and radar
        double distance = calculateDistance3D(
            input.getLatJ(), input.getLongJ(), input.getHeightJ(),
            input.getLatR(), input.getLongR(), input.getHeightR()
        );

        // Calculate ERP in dB: Effective Radiated Power
        double erp_dB = 10 * Math.log10(input.getpJ())
                        + input.getgJ()
                        - input.getlJ();

        // Calculate Free Space Path Loss in dB
        double fspl_dB = 20 * Math.log10(
            4 * Math.PI * distance * input.getFreqJ() * 1e9 / C
        );

        // Calculate bandwidth factor and convert to dB
        double bandwidthFactor = Math.min(input.getbJ(), input.getbR())
                                 / input.getbR();
        double bandwidth_dB = 10 * Math.log10(bandwidthFactor);

        // Calculate Jamming-to-Radar Advantage in dB
        double j_ra_dB = erp_dB - fspl_dB + bandwidth_dB;

        // Create output object and save result to the database
        ECMOutputs output = new ECMOutputs();
        output.setJRa(j_ra_dB);
        output.setParameters(input);
        return outputRepo.save(output);
    }

    // Helper method to calculate great-circle (2D) distance using the Haversine
formula
    private double haversine(double lat1, double lon1,
                             double lat2, double lon2) {
        double R = 6_371_000; // Earth radius in meters
        double dLat = Math.toRadians(lat2 - lat1);
        double dLon = Math.toRadians(lon2 - lon1);

        double a = Math.sin(dLat/2) * Math.sin(dLat/2)
                 + Math.cos(Math.toRadians(lat1))
                   * Math.cos(Math.toRadians(lat2))
                   * Math.sin(dLon/2)
                   * Math.sin(dLon/2);

        return 2 * R * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
    }

    // Helper method to calculate full 3D distance including altitude
    private double calculateDistance3D(double lat1, double lon1, double h1,
                                       double lat2, double lon2, double h2) {
        double horizontal = haversine(lat1, lon1, lat2, lon2); // 2D distance
        return Math.sqrt(horizontal * horizontal + Math.pow(h1 - h2, 2)); //
Pythagorean distance
    }
}
```

ECMOutputs.java

```
package com.xai.ecmmodel.entity;

import jakarta.persistence.*;

@Entity // Marks this class as a JPA entity (mapped to a table)
public class ECMOutputs {

    @Id // Marks the primary key field
    @GeneratedValue(strategy = GenerationType.IDENTITY) // Auto-generates the ID
(auto-increment)
    private Long id;

    private double jRa; // Jamming-to-Radar Advantage value

    @OneToOne // One-to-one relationship with ECMParameters
    @JoinColumn(name = "parameter_id") // Foreign key column in the database
    private ECMParameters parameters;

    // Getter and setter


    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }

    public double getJRa() { return jRa; }
    public void setJRa(double jRa) { this.jRa = jRa; }

    public ECMParameters getParameters() { return parameters; }
    public void setParameters(ECMParameters parameters) { this.parameters =
parameters; }
}
```

ECMParameters.java

```
package com.xai.ecmmodel.entity;

import jakarta.persistence.*;

@Entity // Marks this class as a JPA entity (maps to a database table)
public class ECMParameters {

    @Id // Primary key
    @GeneratedValue(strategy = GenerationType.IDENTITY) // Auto-generated ID
(incremented by database)
    private Long id;

    // Jammer power, gain, and loss
    private double pJ, gJ, lJ;

    // Jammer direction and orientation
    private double azimuthJ, elevationJ;

    // Jammer beamwidths in azimuth and elevation
    private double beamwidthAzJ, beamwidthElJ;

    // Jammer bandwidth and frequency
    private double bJ, freqJ;

    // Jammer location: latitude, longitude, and height
    private double latJ, longJ, heightJ;

    // Type of jammer
    private String jammerType;

    // Radar location and parameters: lat, long, height, bandwidth, and frequency
    private double latR, longR, heightR, bR, freqR;

    // Getters and setters for all fields

    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }

    public double getpJ() { return pJ; }
    public void setpJ(double pJ) { this.pJ = pJ; }

    public double getgJ() { return gJ; }
    public void setgJ(double gJ) { this.gJ = gJ; }

    public double getlJ() { return lJ; }
    public void setlJ(double lJ) { this.lJ = lJ; }
    public double getAzimuthJ() { return azimuthJ; }
    public void setAzimuthJ(double azimuthJ) { this.azimuthJ = azimuthJ; }

    public double getElevationJ() { return elevationJ; }
    public void setElevationJ(double elevationJ) { this.elevationJ = elevationJ; }

    public double getBeamwidthAzJ() { return beamwidthAzJ; }
    public void setBeamwidthAzJ(double beamwidthAzJ) { this.beamwidthAzJ =
beamwidthAzJ; }

    public double getBeamwidthElJ() { return beamwidthElJ; }
    public void setBeamwidthElJ(double beamwidthElJ) { this.beamwidthElJ =
beamwidthElJ; }

    public double getbJ() { return bJ; }
    public void setbJ(double bJ) { this.bJ = bJ; }

    public double getFreqJ() { return freqJ; }
    public void setFreqJ(double freqJ) { this.freqJ = freqJ; }

    public double getLatJ() { return latJ; }
    public void setLatJ(double latJ) { this.latJ = latJ; }

    public double getLongJ() { return longJ; }
    public void setLongJ(double longJ) { this.longJ = longJ; }

    public double getHeightJ() { return heightJ; }
    public void setHeightJ(double heightJ) { this.heightJ = heightJ; }

    public String getJammerType() { return jammerType; }
    public void setJammerType(String jammerType) { this.jammerType = jammerType; }

    public double getLatR() { return latR; }
    public void setLatR(double latR) { this.latR = latR; }

    public double getLongR() { return longR; }
    public void setLongR(double longR) { this.longR = longR; }

    public double getHeightR() { return heightR; }
    public void setHeightR(double heightR) { this.heightR = heightR; }

    public double getbR() { return bR; }
    public void setbR(double bR) { this.bR = bR; }

    public double getFreqR() { return freqR; }
    public void setFreqR(double freqR) { this.freqR = freqR; }
}
```

ECMOutputsRepository.java

```
package com.xai.ecmmodel.repository;

import com.xai.ecmmodel.entity.ECMOutputs;
import org.springframework.data.jpa.repository.JpaRepository;

public interface ECMOutputsRepository extends JpaRepository<ECMOutputs, Long> {}
```

ECMParametersRepository.java

```
package com.xai.ecmmodel.repository;

import com.xai.ecmmodel.entity.ECMParameters;
import org.springframework.data.jpa.repository.JpaRepository;

public interface ECMParametersRepository extends JpaRepository<ECMParameters, Long>
{}
```

RadarModelService.

```
RadarModelService
├── pom.xml
├── README.md
└── src
    └── main
        ├── java
        │   └── com
        │       └── xai
        │           └── radarmodel
        │               ├── Application.java
        │               ├── entity
        │               │   ├── RadarOutputs.java
        │               │   └── RadarParameters.java
        │               ├── RadarController.java
        │               ├── RadarService.java
        │               └── repository
        │                   ├── RadarOutputsRepository.java
        │                   └── RadarParametersRepository.java
        └── resources
```

└── **application.properties**

README.md

~~~
## RadarModelService

### Purpose

This microservice calculates **radar signal performance metrics** such as received
power, signal-to-interference ratio (SIR), and probability of detection (Pd), based
on radar and target configuration inputs.

---

### Endpoint

**POST** `http://localhost:8083/api/radar/calculate

---

### Input

```json
{
  "gr": ...,
  "lo": ...,
  "freqR": ...,
  "pr": ...,
  "br": ...,
  "fr": ...,
  "k": ...,
  "t": ...,
  "latT": ...,
  "longT": ...,
  "heightT": ...,
  "pfa": ...
}
```

#### Field Descriptions

* `gr`: Radar gain (dB)
* `lo`: System loss (dB)
* `freqR`: Radar frequency (GHz)
* `pr`: Transmit power (W)
* `br`: Receiver bandwidth (MHz)
* `fr`: Noise figure (dB)
~~~

* `k`: Boltzmann constant (J/K)
* `t`: System temperature (K)
* `latT`, `longT`, `heightT`: Target location (latitude, longitude in degrees, height in meters)
* `pfa`: Probability of false alarm (between 0 and 1)

### Output

```json
{
  "id": ...,
  "sReceiveRadar": ...,
  "sIRadar": ...,
  "pd": ...
}
```

#### Output Fields

* `id`: Output record ID (auto-generated)
* `sReceiveRadar`: Received signal power at radar (in dB)
* `sIRadar`: Signal-to-interference ratio (in dB)
* `pd`: Probability of detection (unitless, between 0 and 1)

### File Roles & Flow

#### Controller

* `RadarController.java`
  Defines the `/api/radar/calculate` endpoint. Accepts a `RadarParameters` object and returns `RadarOutputs`.

#### Service

* `RadarService.java`
  Core logic for radar equation computation, noise calculation, and Pd estimation.

#### Entities

* `RadarParameters.java`
  Input parameters for radar performance calculations.

* `RadarOutputs.java`
  Output results including received power, SIR, and Pd.

#### Repositories

* `RadarParametersRepository.java`
  JPA repository for persisting input parameters.

* `RadarOutputsRepository.java`
  JPA repository for persisting radar calculation results.

### Database Storage

* **Table:** `radar_parameters`
* **Stored Fields:**

  * `id` (Long, auto-generated)
  * `gr` (double, required)
  * `lo` (double, required)
  * `freqR` (double, required)
  * `pr` (double, required)
  * `br` (double, required)
  * `fr` (double, required)
  * `k` (double, required)
  * `t` (double, required)
  * `latT` (double, required)
  * `longT` (double, required)
  * `heightT` (double, required)
  * `pfa` (double, required)

---

* **Table:** `radar_outputs`
* **Stored Fields:**

  * `id` (Long, auto-generated)
  * `sReceiveRadar` (double, calculated)
  * `sIRadar` (double, calculated)
  * `pd` (double, calculated)
  * `parameter_id` (Long, required, foreign key to `radar_parameters.id`)

### Summary of Connections

```
AnalysisController <--> RadarController <--> RadarService <--> RadarModel
```

pom.xml
```
<!-- Maven project configuration -->
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <!-- Maven model version -->
    <modelVersion>4.0.0</modelVersion>

    <!-- Group ID (usually your domain in reverse) -->
    <groupId>com.xai</groupId>

    <!-- Project name -->
    <artifactId>RadarModelService</artifactId>

    <!-- Project version -->
    <version>0.0.1-SNAPSHOT</version>

    <!-- Use Spring Boot's default configurations -->
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.1.0</version>
    </parent>

    <!-- Project dependencies -->
    <dependencies>
        <!-- Spring Boot starter for building REST APIs -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <!-- Spring Boot starter for JPA and database access -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>

        <!-- PostgreSQL JDBC driver -->
        <dependency>
            <groupId>org.postgresql</groupId>
            <artifactId>postgresql</artifactId>
            <version>42.6.0</version>
        </dependency>
    </dependencies>
    <!-- Build configuration -->
    <build>
        <plugins>
            <!-- Plugin to package and run Spring Boot applications -->
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

Application.java

```java
package com.xai.radarmodel;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

RadarController.java

```java
package com.xai.radarmodel;

import com.xai.radarmodel.entity.RadarParameters;
import com.xai.radarmodel.entity.RadarOutputs;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/radar")
@CrossOrigin(origins = "http://localhost:4200")
public class RadarController {

    @Autowired
    private RadarService service;

    @PostMapping("/calculate")
    public RadarOutputs calculate(@RequestBody RadarParameters input) {
        return service.calculateRadar(input);
    }
}
```

RadarService.java

```
package com.xai.radarmodel;

import com.xai.radarmodel.entity.RadarParameters;
import com.xai.radarmodel.entity.RadarOutputs;
import com.xai.radarmodel.repository.RadarParametersRepository;
import com.xai.radarmodel.repository.RadarOutputsRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service // Marks this class as a Spring service
public class RadarService {

    @Autowired
    private RadarParametersRepository inputRepo; // Repository to save radar inputs

    @Autowired
    private RadarOutputsRepository outputRepo; // Repository to save radar outputs

    private static final double C = 3e8;       // Speed of light (m/s)
    private static final double SIGMA = 1.0;   // Radar cross-section (constant for
now)
    private static final double EPSILON = 1e-10; // Small value to avoid division by
zero

    // Main method to perform radar signal and detection calculations
    public RadarOutputs calculateRadar(RadarParameters input) {
        inputRepo.save(input); // Save input to database

        // Calculate wavelength
        double lambda = C / (input.getFreqR() * 1e9);

        // Calculate distance between origin and target radar
        double r = calculateDistance3D(0, 0, 0, input.getLatT(), input.getLongT(),
input.getHeightT());

        // Convert gain and loss from dB to linear
        double gR = Math.pow(10, input.getGr() / 10);
        double lO = Math.pow(10, input.getLo() / 10);

        // Calculate received power (radar equation)
        double pReceived = (input.getPr() * gR * gR * SIGMA * lambda * lambda) /
```

```
                           (Math.pow(4 * Math.PI, 3) * Math.pow(r, 4) * (lO +
EPSILON));

        // Convert received power to dB
        double sReceiveRadarDb = 10 * Math.log10(pReceived + EPSILON);

        // Calculate system noise (thermal noise)
        double fRLinear = Math.pow(10, input.getFr() / 10);
        double noise = (input.getK() + EPSILON) * (input.getT() + EPSILON)
                * (input.getBr() * 1e6 + EPSILON) * (fRLinear + EPSILON);
        double noiseDb = 10 * Math.log10(noise + EPSILON);

        // Calculate Signal-to-Interference ratio (SIR) in linear and dB
        double sinrLinear = (pReceived + EPSILON) / (noise + EPSILON);
        double sIRadarDb = 10 * Math.log10(sinrLinear + EPSILON);

        // Calculate SNR and probability of detection (Pd)
        double snrLinear = Math.pow(10, (sReceiveRadarDb - noiseDb) / 10);
        double threshold = input.getPfa() > 0 && input.getPfa() < 1
                         ? Math.sqrt(-Math.log(input.getPfa()))
                         : 0;
        double pd = 0.5 * (1 + erf((Math.sqrt(snrLinear) - threshold) /
Math.sqrt(2)));

        // Create output object and save to database
        RadarOutputs output = new RadarOutputs();
        output.setSReceiveRadar(sReceiveRadarDb);
        output.setSIRadar(sIRadarDb);
        output.setPd(Double.isNaN(pd) ? 0.0 : pd); // Handle NaN case
        output.setParameters(input);

        return outputRepo.save(output); // Save and return result
    }

    // Calculate distance over Earth's surface using haversine formula
    private double haversine(double lat1, double lon1, double lat2, double lon2) {
        double R = 6371e3; // Earth radius in meters
        double dLat = Math.toRadians(lat2 - lat1);
        double dLon = Math.toRadians(lon2 - lon1);
        double a = Math.sin(dLat / 2) * Math.sin(dLat / 2) +
                   Math.cos(Math.toRadians(lat1)) * Math.cos(Math.toRadians(lat2)) *
                   Math.sin(dLon / 2) * Math.sin(dLon / 2);
        return 2 * R * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
    }

    // Calculate full 3D distance (includes altitude difference)
    private double calculateDistance3D(double lat1, double lon1, double h1,
                                       double lat2, double lon2, double h2) {
```

```
        double d = haversine(lat1, lon1, lat2, lon2);
        return Math.sqrt(d * d + Math.pow(h1 - h2, 2));
    }

    // Approximate error function (used for Pd calculation)
    private double erf(double x) {
        double t = 1.0 / (1.0 + 0.5 * Math.abs(x));
        double tau = t * Math.exp(-x * x - 1.26551223 + t * (1.00002368 +
                    t * (0.37409196 + t * (0.09678418 + t * (-0.18628806 +
                    t * (0.27886807 + t * (-1.13520398 + t * (1.48851587 +
                    t * (-0.82215223 + t * 0.17087277)))))))));
        return x >= 0 ? 1 - tau : tau - 1;
    }
}
```

RadarOutputs.java

```
package com.xai.radarmodel.entity;

import jakarta.persistence.*;

@Entity // Marks this class as a JPA entity (represents a DB table)
public class RadarOutputs {

  @Id // Primary key
  @GeneratedValue(strategy = GenerationType.IDENTITY) // Auto-generated ID
  private Long id;

  // Received signal power at the radar (in dB)
  private double sReceiveRadar;

  // Signal-to-interference ratio at radar (in dB)
  private double sIRadar;

  // Probability of detection
  private double pd;

  // Link to input parameters used for the calculation
  @OneToOne
  @JoinColumn(name = "parameter_id") // Foreign key to RadarParameters table
```

```java
    private RadarParameters parameters;

    // Getters and Setters

    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }

    public double getSReceiveRadar() { return sReceiveRadar; }
    public void setSReceiveRadar(double sReceiveRadar) { this.sReceiveRadar = sReceiveRadar; }

    public double getSIRadar() { return sIRadar; }
    public void setSIRadar(double sIRadar) { this.sIRadar = sIRadar; }

    public double getPd() { return pd; }
    public void setPd(double pd) { this.pd = pd; }

    public RadarParameters getParameters() { return parameters; }
    public void setParameters(RadarParameters parameters) { this.parameters = parameters; }
}
```

RadarParameters.java

```java
package com.xai.radarmodel.entity;

import jakarta.persistence.*;

@Entity // Marks this class as a JPA entity (maps to a database table)
public class RadarParameters {

    @Id // Primary key
    @GeneratedValue(strategy = GenerationType.IDENTITY) // Auto-increment ID
    private Long id;

    // Radar antenna gain (in dB)
    private double gr;

    // System loss (in dB)
    private double lo;
```

```
// Radar frequency (in GHz)
private double freqR;

// Transmit power (in watts)
private double pr;

// Bandwidth (in MHz)
private double br;

// Receiver noise figure (in dB)
private double fr;

// Boltzmann constant
private double k;

// System temperature (in Kelvin)
private double t;

// Target coordinates (latitude, longitude, and height in meters)
private double latT, longT, heightT;

// Probability of false alarm (used in detection threshold calculation)
private double pfa;

// Getters and Setters for all fields

public Long getId() { return id; }
public void setId(Long id) { this.id = id; }

public double getGr() { return gr; }
public void setGr(double gr) { this.gr = gr; }

public double getLo() { return lo; }
public void setLo(double lo) { this.lo = lo; }

public double getFreqR() { return freqR; }
public void setFreqR(double freqR) { this.freqR = freqR; }
```

```
    public double getPr() { return pr; }
    public void setPr(double pr) { this.pr = pr; }

    public double getBr() { return br; }
    public void setBr(double br) { this.br = br; }

    public double getFr() { return fr; }
    public void setFr(double fr) { this.fr = fr; }

    public double getK() { return k; }
    public void setK(double k) { this.k = k; }

    public double getT() { return t; }
    public void setT(double t) { this.t = t; }

    public double getLatT() { return latT; }
    public void setLatT(double latT) { this.latT = latT; }

    public double getLongT() { return longT; }
    public void setLongT(double longT) { this.longT = longT; }

    public double getHeightT() { return heightT; }
    public void setHeightT(double heightT) { this.heightT = heightT; }

    public double getPfa() { return pfa; }
    public void setPfa(double pfa) { this.pfa = pfa; }
}
```

RadarOutputsRepository.java

```
package com.xai.radarmodel.repository;

import com.xai.radarmodel.entity.RadarOutputs;
import org.springframework.data.jpa.repository.JpaRepository;

public interface RadarOutputsRepository extends JpaRepository<RadarOutputs, Long> {}
```

RadarParametersRepository.java

```
package com.xai.radarmodel.repository;

import com.xai.radarmodel.entity.RadarParameters;
import org.springframework.data.jpa.repository.JpaRepository;

public interface RadarParametersRepository extends JpaRepository<RadarParameters,
Long> {}
```

AnalysisManagerService.

AnalysisManagerService

```
├── offline-m2
├── pom.xml
├── README.md
└── src
    └── main
        ├── java
        │   └── com
        │       └── xai
        │           └── analysismanager
        │               ├── AnalysisManagerServiceApplication.java
        │               ├── controller
        │               │   └── AnalysisController.java
        │               └── dto
        │                   └── AnalysisRequest.java
        └── resources
            └── application.properties
```

README.md

```
## AnalysisManagerService

### Purpose

This microservice **orchestrates computation across three domain-specific services**:
Absorption Loss, ECM Model, and Radar Model. It collects inputs, sends them to the
respective services via REST, and aggregates their outputs into a unified response.

---
```

~~~~
###  Input

* **POST** `http://localhost:8084/api/analysis/request`
* **Request Body**: JSON format with exactly **one model** and its **parameter
object**.

Example:

```json
For Absorption Loss Model
{
  "parameters": {
    "absorption": {...}
  }
}

For ECM
{
  "parameters": {
    "ecm": {...}
  }
}

For Radar Model
{
  "parameters": {
    "radar": {...}
  }
}
```

---

###  Output

* If valid:

  * Returns **200 OK** with the exact JSON output from the model microservice (e.g.
radar, ECM, or absorption).

  Example:

```json
For Absorption Loss Model
{"id".., "ldash": ... }

For ECM
~~~~

~~~
{ "id".., "jra": ... }

For Radar Model
{
  "id"..,
  "pd": ...,
  "sreceiveRadar": ..,
  "siradar": ..
}
```
* If error:
  * Returns **200 OK** with the error in JSON


  Example:

```json
For Absorption Loss Model
{ "error": ... }
```

---

### File Roles & Flow

* **`AnalysisManagerServiceApplication.java`**

  * Spring Boot main class.
  * Starts the service on port `8084`.

* **`AnalysisController.java`**

  * Defines `/api/analysis/run` endpoint.
  * Accepts composite input (Absorption, ECM, Radar).
  * Delegates processing to `AnalysisService`.

* **`AnalysisService.java`**

  * Sends REST requests to:

    * Absorption model at `http://localhost:8081/api/absorption/calculate`
    * ECM model at `http://localhost:8082/api/ecm/calculate`
    * Radar model at `http://localhost:8083/api/radar/calculate`
  * Aggregates all outputs into a single result JSON.

* **`AnalysisRequest.java`**

  * DTO representing incoming JSON.
~~~

~~~
  * Holds three sub-objects: `absorptionParams`, `ecmParams`, and `radarParams`.

* **`AnalysisResult.java`**

  * DTO for combined output.
  * Includes: `absorptionOutput`, `ecmOutput`, `radarOutput`.

---

### Database Storage

**None.** This service does **not use a database**. It only acts as a stateless
aggregator and coordinator.

---

### Summary of Connections

```
Frontend <--> AnalysisController
                     |
                     v
     +---------------+---------------+
     |               |               |
AbsorptionService ECMModelService RadarModelService
    (8081)            (8082)           (8083)
```

---
~~~

pom.xml

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                             http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <!-- Inherit default dependencies and configurations from Spring Boot parent -->
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.2.0</version>
        <relativePath/> <!-- Lookup parent from repository -->
    </parent>

    <!-- Project metadata -->
```

```xml
    <groupId>com.xai</groupId>
    <artifactId>analysis-manager-service</artifactId>
    <version>1.0.0</version>
    <packaging>jar</packaging>

    <name>Analysis Manager Service</name>
    <description>Central orchestrator for radar, ECM, and absorption loss
services</description>

    <!-- Java version used -->
    <properties>
        <java.version>21</java.version>
    </properties>

    <dependencies>
        <!-- For building RESTful APIs -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <!-- For using WebClient to call other microservices -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-webflux</artifactId>
        </dependency>

        <!-- For JSON serialization/deserialization -->
        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-databind</artifactId>
        </dependency>

        <!-- For reducing boilerplate code using annotations -->
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>

        <!-- For unit and integration testing -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <!-- Package and run the application using Spring Boot -->
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

AnalysisManagerServiceApplication.java

```
package com.xai.analysismanager;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class AnalysisManagerServiceApplication {
    public static void main(String[] args) {
        SpringApplication.run(AnalysisManagerServiceApplication.class, args);
    }
}
```

AnalysisController.java

```
package com.xai.analysismanager.controller;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.reactive.function.client.WebClient;
import com.xai.analysismanager.dto.AnalysisRequest;

import java.util.Map;

@RestController
@RequestMapping("/api/analysis")
@CrossOrigin(origins = "http://localhost:4200") // Allow frontend to call this API
public class AnalysisController {

    // Injected from application.properties – endpoint URLs of microservices
    @Value("${model.radar.url}")
    private String radarUrl;
    @Value("${model.ecm.url}")
    private String ecmUrl;

    @Value("${model.absorption.url}")
    private String absorptionUrl;

    // WebClient used to forward the request to the actual model microservices
    private final WebClient webClient = WebClient.create();

    // Accepts a POST request with a scenario ID and model input
    @PostMapping("/request")
    public ResponseEntity<Object> proxyModelRequest(@RequestBody AnalysisRequest
request) {
        // Ensure only 1 model (radar/ecm/absorption) is requested at a time
        if (request.getParameters().size() != 1) {
            return ResponseEntity.badRequest().body(Map.of("error", "Only one model
should be requested at a time."));
        }

        // Extract model name and input parameters
        String model = request.getParameters().keySet().iterator().next();
        Object params = request.getParameters().get(model);

        // Get endpoint URL for the requested model
        String endpoint = resolveModelEndpoint(model);
        if (endpoint == null) {
            return ResponseEntity.badRequest().body(Map.of("error", "Missing or
unknown model: " + model));
        }

        try {
            // Forward request to the microservice using WebClient
            Object response = webClient.post()
                    .uri(endpoint)
                    .bodyValue(params)
                    .retrieve()
                    .bodyToMono(Object.class)
                    .block(); // Blocking call for simplicity

            if (response == null) {
                return ResponseEntity.badRequest().body(Map.of("error", "invalid
input"));
            }

            return ResponseEntity.ok(response);
        } catch (Exception e) {
            // Handle all possible call failures
            return ResponseEntity.badRequest().body(Map.of("error", "model not
running properly"));
        }
    }

    // Maps model name to the corresponding service endpoint
    private String resolveModelEndpoint(String modelName) {
        return switch (modelName.toLowerCase()) {
            case "radar" -> radarUrl;
            case "ecm" -> ecmUrl;
            case "absorption" -> absorptionUrl;
            default -> null;
        };
    }
}
```

AnalysisRequest.java

```
package com.xai.analysismanager.dto;

import java.util.Map;

/**
 * DTO (Data Transfer Object) for receiving analysis request data.
 * Contains scenario ID and a map of input parameters to pass to services.
 */
public class AnalysisRequest {

    // Unique identifier for the scenario being analyzed
    private String scenarioId;

    // Key-value pairs of parameters relevant to the services
    private Map<String, Object> parameters;

    public String getScenarioId() {
        return scenarioId;
    }

    public void setScenarioId(String scenarioId) {
        this.scenarioId = scenarioId;
    }

    public Map<String, Object> getParameters() {
        return parameters;
    }
    public void setParameters(Map<String, Object> parameters) {
        this.parameters = parameters;
    }

}
```

AuthenticationService

AuthenticationService

```
├── pom.xml
├── README.md
└── src
    └── main
        ├── java
        │   └── com
        │       └── xai
        │           └── authentication
        │               ├── entity
        │               │   └── User.java
        │               ├── LoginController.java
        │               ├── repository
        │               │   └── UserRepository.java
        │               ├── TraineeApplication.java
        │               └── UserService.java
        └── resources
            └── application.properties
```

README.md

```
## AuthenticationService

### Purpose

This microservice handles **user registration and login** with basic credential
checks. It is used by the frontend to allow access to protected parts of the system.

---
```

```
### Input / Output

| Action   | Endpoint                  | Input (JSON)                         |
Output (JSON)
|
| -------- | ------------------------- | ------------------------------------ |
-------------------------------------------------------------------------------------
------------------------- |
| Register | `POST http://localhost:8085/api/auth/register` | `{ "name": "user",
"password": "pass" }` | `{ "message": "User registered" }` or `{ "error": "Username
already exists" }`                                 |
| Login    | `POST http://localhost:8085/api/auth/login`    | `{ "name": "user",
"password": "pass" }` | `{ "message": "Login successful", "user": { "id": ...,
"name": ... } }` or `{ "error": "Invalid credentials" }` |

---

### File Roles & Flow

* **`TraineeApplication.java`**

  * Bootstraps the Spring Boot application.

* **`LoginController.java`**

  * Defines the `/register` and `/login` endpoints.
  * Accepts JSON payload and delegates logic to `UserService`.

* **`UserService.java`**

  * Contains logic to:

    * Register user if name is unique
    * Validate login credentials
  * Uses `UserRepository` to query/save in DB.

* **`UserRepository.java`**

  * JPA interface to query `users` table.
  * Provides `findByName()` and `existsByName()`.

* **`User.java`**

  * Entity mapped to `users` table.
  * Fields: `id`, `name`, `password`, `createdAt`.

---
```

~~~~
### Database Storage

* **Table:** `users`
* **Stored Fields:**

  * `id` (auto-generated)
  * `name` (unique, required)
  * `password` (stored as plain text for now)
  * `createdAt` (timestamp)

---

### Summary of Connections

```
Frontend <--> LoginController <--> UserService <--> UserRepository <--> PostgreSQL
```

_
~~~~

pom.xml

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                             http://maven.apache.org/xsd/maven-4.0.0.xsd">

  <modelVersion>4.0.0</modelVersion>

  <groupId>com.trainee</groupId>         <!-- Base package -->
  <artifactId>prototype25</artifactId>   <!-- Project name -->
  <name>RoleBasedUsersManagementSystem</name>
  <version>1.0.0</version>
  <packaging>jar</packaging>

  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.7.0</version>             <!-- Spring Boot version -->
    <relativePath/>
  </parent>

  <properties>
    <java.version>17</java.version>      <!-- Java version used -->
  </properties>
```

```xml
  <dependencies>
    <!-- For building REST APIs -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <!-- For database interaction using JPA -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>

    <!-- PostgreSQL driver -->
    <dependency>
      <groupId>org.postgresql</groupId>
      <artifactId>postgresql</artifactId>
      <scope>runtime</scope>
    </dependency>
  </dependencies>

  <build>
    <plugins>
      <!-- Enables packaging and running Spring Boot apps -->
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
    </plugins>
  </build>
</project>
```

User.java

```java
package com.xai.authentication.entity;

import javax.persistence.*;
import java.time.LocalDateTime;

@Entity
@Table(name = "users") // Maps this class to the "users" table in the database
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY) // Auto-incrementing primary
key
```

```
    private Long id;

    @Column(length = 50, nullable = false, unique = true) // Unique username,
required
    private String name;

    @Column(nullable = false) // Password is required
    private String password;

    private LocalDateTime createdAt; // Timestamp for when user was created

    public User() {}

    public User(String name, String password) {
        this.name = name;
        this.password = password;
    }

    // Standard getters and setters
    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }

    public String getPassword() { return password; }
    public void setPassword(String password) { this.password = password; }

    public LocalDateTime getCreatedAt() { return createdAt; }
    public void setCreatedAt(LocalDateTime createdAt) { this.createdAt = createdAt; }
}
```

LoginController.java

```
package com.xai.authentication;

import java.util.Map;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
```

```java
@RestController
@RequestMapping("/api/auth")
@CrossOrigin(origins = "http://localhost:4200") // Allows frontend at port 4200 to
access this API
public class LoginController {

    @Autowired
    private UserService userService;

    // DTO class for receiving login/register requests
    public static class UserDto {
        private String name;
        private String password;
        public String getName() { return name; }
        public void setName(String name) { this.name = name; }
        public String getPassword() { return password; }
        public void setPassword(String password) { this.password = password; }
    }

    // Endpoint to register a new user
    @PostMapping("/register")
    public ResponseEntity<Map<String, String>> register(@RequestBody UserDto dto) {
        boolean created = userService.register(dto.getName(), dto.getPassword());
        if (!created) {
            // Username already exists
            return ResponseEntity.status(HttpStatus.CONFLICT)
                    .body(Map.of("error", "Username already exists"));
        }
        return ResponseEntity.ok(Map.of("message", "User registered"));
    }

    // Endpoint to authenticate user login
    @PostMapping("/login")
    public ResponseEntity<Map<String, Object>> login(@RequestBody UserDto dto) {
        return userService.findByName(dto.getName())
            .filter(user -> user.getPassword().equals(dto.getPassword())) // Validate
password
            .map(user -> Map.of(
                "message", "Login successful",
                "user", Map.of(
                    "id", user.getId(),
                    "name", user.getName()
                )
            ))
            .map(ResponseEntity::ok) // Return success response
            .orElse(ResponseEntity.status(HttpStatus.UNAUTHORIZED)
                .body(Map.of("error", "Invalid credentials"))); // If user not found
or wrong password
```

```
    }

}
```

UserRepository.java

```
package com.xai.authentication.repository;

import java.util.Optional;
import org.springframework.data.jpa.repository.JpaRepository;

import com.xai.authentication.entity.User;

public interface UserRepository extends JpaRepository<User, Long> {
    Optional<User> findByName(String name);
    boolean existsByName(String name);
}
```

UserService.java

```
package com.xai.authentication;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.xai.authentication.entity.User;
import com.xai.authentication.repository.UserRepository;

import java.time.LocalDateTime;
import java.util.Optional;

@Service
public class UserService {

    @Autowired
    private UserRepository userRepository; // Injects the JPA repository for User
entity

    // Verifies username and password match
    public boolean authenticate(String name, String rawPassword) {
        return userRepository.findByName(name)
                .map(user -> user.getPassword().equals(rawPassword)) // Simple
equality check (no hashing)
```

```
                .orElse(false); // If user not found, return false
    }

    // Registers a new user if the username doesn't already exist
    public boolean register(String name, String password) {
        if (userRepository.existsByName(name)) return false; // Prevent duplicate 
usernames
        User user = new User();
        user.setName(name);
        user.setPassword(password); // Stores raw password (should be hashed in 
production)
        user.setCreatedAt(LocalDateTime.now());
        userRepository.save(user); // Persist user in database
        return true;
    }

    // Returns user by name (wrapped in Optional)
    public Optional<User> findByName(String name) {
        return userRepository.findByName(name);
    }
}
```

TraineeApplication.java

```
package com.xai.authentication;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class TraineeApplication {
    public static void main(String[] args) {
        SpringApplication.run(TraineeApplication.class, args);
    }
}
```

Frontend

Root (Frontend/)

```
Frontend/
├── angular.json         # Angular build configuration (important for build and serve)
├── package.json         # Declares dependencies and scripts
├── package-lock.json    # Lockfile for reproducible builds
├── tsconfig.json        # Global TypeScript compiler config
├── tsconfig.app.json    # App-specific TS config
├── tsconfig.spec.json   # Unit test TS config
├── assets/              # Static assets and Cesium offline resources
└──src/                  # Main source code
```

src/pages/ (Main Application Logic)

```
src/pages/
├── auth/       # Login & authentication
├── dashboard/  # Main dashboard with Cesium + charts
```

File Descriptions

Each page folder (e.g., auth, dashboard) follows this structure:

File	Description
*.html	HTML template for the component
*.ts	Component logic (TypeScript class with view-model logic)
*.css	Component-specific styling

CesiumJS Offline Setup

CesiumJS assets are stored in 'assets/cesium' to allow offline visualization of geospatial simulations.

Viewer is initialized in dashboard.ts, pointing to these local Cesium assets via:

```
(window as any).CESIUM_BASE_URL = 'assets/cesium';
```

Key Source Code Walkthrough

src/pages/

```
├── auth
│   ├── auth.html
│   └── auth.ts
```

auth.html

```
<!-- Full-page wrapper for centering the form -->
<div class="wrapper">
    <!-- Login form, bound to Angular's form system -->
    <form (ngSubmit)="login()" #loginForm="ngForm" class="login-form">
        <!-- Form heading -->
        <h2>Login</h2>
        <!-- Username input -->
        <label for="username">Name:</label>
        <input
            id="username"
            type="username"
            [(ngModel)]="name"
            name="username"
            placeholder="Enter username"
            required
        />
        <!-- Password input -->
        <label for="password">Password:</label>
        <input
            type="password"
            id="password"
            [(ngModel)]="password"
            name="password"
            required
        />
        <!-- Submit button -->
        <button type="submit">Login</button>
        <!-- Error message shown if login fails -->
        <p class="error">{{ errorMsg }}</p>
        <!-- Optional register link (currently commented out) -->
        <!--
        <p style="text-align: center; margin-top: 10px;">
            Don't have an account?
            <a [routerLink]="['/register']"
               style="color: #2a9d8f; text-decoration: none; font-weight: bold;">
               Register here
            </a>
        </p>
        -->
    </form>
</div>
```

auth.ts

```
import { Component, OnInit } from '@angular/core';
import { Router } from '@angular/router';
import { CommonModule } from '@angular/common';
import { FormsModule } from '@angular/forms';
import { RouterModule } from '@angular/router'; // Import for routing support
@Component({
  selector: 'app-auth',
  standalone: true, // Allows the component to be used independently
  imports: [FormsModule, RouterModule, CommonModule], // Import necessary modules
  templateUrl: './auth.html',
  styleUrls: ['./auth.css']
})
export class Auth implements OnInit {
  // Form fields
  users: string[] = [];       // (Unused if user list isn't fetched)
  name = '';                  // Bound to username input
  password = '';              // Bound to password input
  errorMsg = '';              // Error message shown on form
  constructor(private router: Router) {}
  ngOnInit(): void {
    // Initialization logic here if needed
    // this.users fetch is now redundant if you're using free-text input
  }
  // Called on login form submission
  login(): void {
    this.errorMsg = ''; // Clear previous error
    // Validate input fields
    if (!this.name || !this.password) {
      this.errorMsg = 'Please select a username and enter password.';
      return;
    }
    // Send POST request to backend login endpoint
    fetch('http://127.0.0.1:8085/api/auth/login', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ name: this.name, password: this.password })
    })
    .then(res => res.json().then(body => ({ status: res.status, body }))) // Combine
status + body
    .then(({ status, body }) => {
      if (status === 200) {
        const user = body.user;
        localStorage.setItem('currentUser', JSON.stringify(user)); // Save user in
localStorage
        // Navigate to dashboard after successful login
        this.router.navigate(['/dashboard']);
      } else if (status === 401) {
        this.errorMsg = 'Invalid credentials'; // Wrong username/password
      } else {
        this.errorMsg = body.error || 'Server error. Try again.';
      }
    })
    .catch(() => {
      this.errorMsg = 'Network error. Try again later.'; // Server unreachable or
failed fetch
    });
  }
}
```

src/pages/
├── dashboard
│   ├── dashboard.html
│   └── dashboard.ts

dashboard.html

```
<!-- Main Wrapper -->
<div class="wrapper">
<!-- Application Layout Container (Grid) -->
<div class="app-container">
<!-- Header section for the dashboard -->
<header class="header">
  <!-- Title of the dashboard -->
  <h1>Radar Microservices Dashboard</h1>
  <!-- Navigation menu -->
  <nav class="nav-menu">
    <a>Home</a>                         <!-- Placeholder link for Home -->
    <a>Microservice Setup</a>          <!-- Placeholder link for microservice setup
-->
    <a>Results Analysis</a>            <!-- Placeholder link for results analysis -->
    <a>Settings</a>                    <!-- Placeholder link for settings -->
    <a>Help & Documentation</a>        <!-- Placeholder link for help/docs -->
    <a [routerLink]="['/auth']">Log Out</a> <!-- Link to route user back to login -->
  </nav>

</header>
<!-- Sidebar for microservice input forms -->
```

```html
<aside class="sidebar">
  <h2>Microservice Inputs</h2>
  <!-- Dropdown to select which microservice form to display -->
  <label>
    Select Microservice:
    <select #serviceSelect>
      <option value="absorption-loss">Absorption Loss</option>
      <option value="ecm-model">ECM Model</option>
      <option value="radar-model">Radar Model</option>
    </select>
  </label>
  <!-- ================= Absorption Loss Inputs ================= -->
  <div id="form-absorption-loss" class="service-form">
    <!-- Add dynamic weather conditions -->
    <button id="add-weather-btn" #addWeatherBtn type="button">+ Add Weather
Condition</button>
    <div #weatherContainer></div>
    <!-- Weather & environmental factors -->
    <label>Rainfall Rate (RFR mm/hr):
      <input id="rfr" type="number" value="10">
    </label>
    <label>Snowfall Rate (SFR mm/hr):
      <input id="sfr" type="number" value="5">
    </label>
    <label>Visibility Fog (m):
      <input id="visFog" type="number" value="3">
    </label>
    <!-- Frequency parameters -->
    <label>Radar Frequency (Freq_r GHz):
      <input id="freqR" type="number" value="1.2e9">
    </label>
    <label>Jammer Frequency (Freq_j GHz):
      <input id="freqJ" type="number" value="1e9">
    </label>
    <label>Operating Frequency (Freq_op GHz):
      <input id="freqOp" type="number" value="1.2e9">
    </label>
    <!-- Radar location -->
    <label>Radar Latitude:
      <input id="latR" type="number" value="34.2">
    </label>
    <label>Radar Longitude:
      <input id="longR" type="number" value="69.2">
    </label>
    <label>Radar Height (m):
      <input id="heightR" type="number" value="500">
    </label>
    <!-- Target location -->
    <label>Target Latitude:
      <input id="latT" type="number" value="34.1">
    </label>
    <label>Target Longitude:
      <input id="longT" type="number" value="69.1">
    </label>
    <label>Target Height (m):
      <input id="heightT" type="number" value="2000">
    </label>
    <!-- Jammer location -->
    <label>Jammer Latitude:
      <input id="latJ" type="number" value="34.1">
    </label>
    <label>Jammer Longitude:
      <input id="longJ" type="number" value="69.1">
    </label>
    <label>Jammer Height (m):
      <input id="heightJ" type="number" value="2000">
    </label>
  </div>
  <!-- ================= ECM Model Inputs ================= -->
  <div id="form-ecm-model" class="service-form">
    <!-- Jammer signal and antenna parameters -->
    <label>Jammer Power (P_j W):
      <input id="pJ" type="number" value="100">
    </label>
    <label>Jammer Gain (G_j dB):
      <input id="gJ" type="number" value="20">
    </label>
    <label>Jammer Loss (L_j dB):
      <input id="lJ" type="number" value="10">
    </label>
    <label>Azimuth (Azimuth_j):
      <input id="azimuthJ" type="number" value="35">
    </label>
    <label>Elevation (Elevation_j):
      <input id="elevationJ" type="number" value="24">
    </label>
    <label>Beamwidth Azimuth:
      <input id="beamwidthAzJ" type="number" value="25">
    </label>
    <label>Beamwidth Elevation:
      <input id="beamwidthElJ" type="number" value="25">
    </label>
    <label>Jammer Bandwidth (B_j Hz):
      <input id="bJ" type="number" value="500e6">
    </label>
    <label>Jammer Frequency (Freq_j GHz):
```

```
  <input id="freqJ" type="number" value="1">
</label>
<!-- Jammer location -->
<label>Jammer Latitude:
  <input id="latJ" type="number" value="34.1">
</label>
<label>Jammer Longitude:
  <input id="longJ" type="number" value="69.1">
</label>
<label>Jammer Height (m):
  <input id="heightJ" type="number" value="2000">
</label>
<!-- Jammer configuration -->
<label>Jammer Type:
  <select id="jammerType">
    <option>Responsive</option>
    <option>Non-Responsive</option>
  </select>
</label>
<label id="jammerSubWrapper">
  Sub Option:
  <select id="jammerSubType">
    <option>Self - protection</option>
    <option>Escort (non stationary standoff)</option>
    <option>Stationary stand off</option>
  </select>
</label>
<!-- Antenna configuration -->
<label>
  Antenna Type:
  <select #antennaTypeECM id="antennaType">
    <option>Gain Pattern</option>
    <option>Measured gain</option>
    <option>User Defined Constant Gain</option>
  </select>
</label>
<label #antennaSubWrapperECM id="antennaSubWrapper" style="display:none;">
  Sub Option:
  <select id="antennaSubType">
    <option>Omnidirectional</option>
    <option>Uniform</option>
    <option>Cosine</option>
    <option>Taylor's Beam</option>
    <option>Consecant Sequare beam</option>
    <option>Fan Beam</option>
    <option>Gaussian</option>
    <option>Parabolic</option>
    <option>Electronically Scanned Array</option>
    </select>
  </label>
  <!-- Radar parameters -->
  <label>Radar Latitude:
    <input id="latR" type="number" value="34.2">
  </label>
  <label>Radar Longitude:
    <input id="longR" type="number" value="69.2">
  </label>
  <label>Radar Height (m):
    <input id="heightR" type="number" value="500">
  </label>
  <label>Radar Bandwidth (B_r Hz):
    <input id="bR" type="number" value="200e6">
  </label>
  <label>Radar Frequency (Freq_r GHz):
    <input id="freqRm" type="number" value="1.2">
  </label>
</div>
<!-- ================= Radar Model Inputs ================= -->
<div id="form-radar-model" class="service-form">
  <!-- Radar gain and loss parameters -->
  <label>Radar Gain (G_r dB):
    <input id="gr" type="number" value="8.3">
  </label>
  <label>Radar Loss (L_o dB):
    <input id="lo" type="number" value="3.7">
  </label>
  <!-- Polarization settings -->
  <label>Polarization_R:
    <select id="polarizationR">
      <option>Linear Horizontal</option>
      <option>Linear Vertical</option>
      <option>RCP</option>
      <option>LCP</option>
    </select>
  </label>
  <label>Polarization_J:
    <select id="polarizationJ">
      <option>Linear 45</option>
      <option>RCP</option>
      <option>LCP</option>
    </select>
  </label>
  <!-- Antenna configuration -->
  <label>
    Antenna Type:
    <select #antennaTypeRadar id="antennaType">
      <option>Gain Pattern</option>
      <option>Measured gain</option>
      <option>User Defined Constant Gain</option>
    </select>
  </label>
  <label #antennaSubWrapperRadar id="antennaSubWrapper" style="display:none;">
    Sub Option:
    <select id="antennaSubType">
      <option>Omnidirectional</option>
      <option>Uniform</option>
      <option>Cosine</option>
      <option>Taylor's Beam</option>
      <option>Consecant Sequare beam</option>
      <option>Fan Beam</option>
      <option>Gaussian</option>
      <option>Parabolic</option>
      <option>Electronically Scanned Array</option>
    </select>
  </label>
  <!-- Radar signal and system parameters -->
  <label>Radar Frequency (Freq_r GHz):
    <input id="freqR" type="number" value="12.4e9">
  </label>
  <label>Radar Power (P_r W):
    <input id="pr" type="number" value="15200000">
  </label>
  <label>Radar Bandwidth (B_r Hz):
    <input id="br" type="number" value="56.7e6">
  </label>
  <label>Noise Figure (F_r dB):
    <input id="fr" type="number" value="6.2">
  </label>
  <label>Boltzmann Constant (k):
    <input id="k" type="number" value="1.38e-23">
  </label>
  <label>System Temperature (T K):
    <input id="t" type="number" value="290">
  </label>
  <!-- Beam properties -->
  <label>Bandwidth Elevation:
    <input id="BMWEl_R" type="number" value="-32">
  </label>
  <label>Bandwidth Azimuth:
    <input id="BMWAz_R" type="number" value="67">
  </label>
  <label>Elevation Limit:
    <input id="elLimit" type="number" value="65">
  </label>
```

```
    <!-- Target location -->
    <label>Target Latitude:
      <input id="latT" type="number" value="22.4">
    </label>
    <label>Target Longitude:
      <input id="longT" type="number" value="88.3">
    </label>
    <label>Target Height (m):
      <input id="heightT" type="number" value="150">
    </label>
    <label>False Alarm Prob (pfa):
      <input id="pfa" type="number" value="1.2e-6">
    </label>
  </div>
  <!-- Button to execute selected microservice -->
  <button #runBtn class="run-btn">Run Microservice</button>
</aside>
<!-- Main content area -->
<main class="main">
  <!-- Section for displaying 3D radar visualization -->
  <section class="map-container">
    <h2>3D Radar Visualization</h2>
    <!-- Placeholder for 3D map or radar visualization -->
    <div #map3D class="map-placeholder"></div>
  </section>
</main>
<!-- Output panel displaying logs and microservice results -->
<aside class="output-panel">
  <!-- Section for displaying log or status messages -->
  <h3>Log Messages</h3>
  <div #logBox class="log-box"></div>
  <!-- Section for displaying microservice output charts or results -->
  <h3>Microservice Output</h3>
  <div #chartsPlaceholder class="charts-placeholder"></div>
</aside>
<!-- Footer section providing export options and status display -->
<footer class="footer"
        style="grid-area: footer; background: #1a2525; padding: 1rem; display: flex;
justify-content: space-between; align-items: center;">
  <!-- Left section: export buttons for user to download results -->
  <div>
    <!-- Button to export the output data to a PDF file -->
    <button (click)="exportToPDF()">Export PDF</button>
    <!-- Button to export the output data to a CSV file -->
    <button (click)="exportToCSV()">Export CSV</button>
  </div>
  <!-- Right section: shows current status and progress bar -->
  <div class="status" style="text-align: right;">
```

```
    <!-- Displays current system or task status -->
    <p #statusText>Status: Idle</p>
    <!-- Progress bar to visually indicate task progress -->
    <progress #progressBar max="100" value="0"></progress>
  </div>
</footer>
</div>
</div>
```

dashboard.ts

```
// Angular imports
import {
  Component,
  AfterViewInit,
  ViewChild,
  ElementRef,
  ViewEncapsulation
} from '@angular/core';
import { CommonModule } from '@angular/common';
import { FormsModule } from '@angular/forms';
import { RouterModule } from '@angular/router';
import { HttpClientModule } from '@angular/common/http';
// External libraries
declare const Cesium: any;           // For 3D map rendering
import { jsPDF } from 'jspdf';       // For PDF export
import { errorContext } from 'rxjs/internal/util/errorContext';
declare const Chart: any;            // For charting (if used)
@Component({
  selector: 'app-dashboard',
  standalone: true,  // Enables this component to be used without NgModule
  imports: [CommonModule, FormsModule, HttpClientModule, RouterModule],
  templateUrl: './dashboard.html',
  styleUrls: ['./dashboard.css'],
  encapsulation: ViewEncapsulation.None  // Allows global CSS styles
})
export class Dashboard implements AfterViewInit {
  // Form element references (linked to HTML template using @ViewChild)
  @ViewChild('serviceSelect') serviceSelect!: ElementRef<HTMLSelectElement>;       //
Dropdown to choose service
  @ViewChild('addWeatherBtn') addWeatherBtn!: ElementRef<HTMLButtonElement>;       //
Button to add weather input
  @ViewChild('weatherContainer') weatherContainer!: ElementRef<HTMLDivElement>;    //
Container holding weather blocks
  @ViewChild('runBtn') runBtn!: ElementRef<HTMLButtonElement>;                     //
Button to execute microservice
  @ViewChild('map3D') map3D!: ElementRef<HTMLDivElement>;                          //
3D map display area
```

```
  @ViewChild('chartsPlaceholder') chartsPlaceholder!: ElementRef<HTMLDivElement>;  //
Placeholder for result tables/charts
  @ViewChild('logBox') logBox!: ElementRef<HTMLDivElement>;                         //
Console-style log output box
  @ViewChild('statusText') statusText!: ElementRef<HTMLParagraphElement>;           //
Status display text
  @ViewChild('progressBar') progressBar!: ElementRef<HTMLProgressElement>;          //
Progress bar
  // Antenna dropdowns and sub-sections (for ECM and Radar)
  @ViewChild('antennaTypeECM') antennaTypeECM!: ElementRef<HTMLSelectElement>;
  @ViewChild('antennaSubWrapperECM') antennaSubECM!: ElementRef<HTMLElement>;
  @ViewChild('antennaTypeRadar') antennaTypeRadar!: ElementRef<HTMLSelectElement>;
  @ViewChild('antennaSubWrapperRadar') antennaSubRadar!: ElementRef<HTMLElement>;
  // To store input/output for each session run (used for exporting results later)
  private sessionRuns: {
    model: string;
    input: any;
    output: any;
  }[] = [];
  // API endpoints for each microservice model
  serviceEndpoints = {
    'absorption-loss': 'http://localhost:8084/api/analysis/request',
    'ecm-model':       'http://localhost:8084/api/analysis/request',
    'radar-model':     'http://localhost:8084/api/analysis/request',
  };
ngAfterViewInit(): void {
  // Set base path for Cesium assets
  (window as any).CESIUM_BASE_URL = 'assets/cesium';
  // Attach event listeners
  this.serviceSelect.nativeElement.addEventListener('change', () =>
this.onServiceChange());  // Handle service switch
  this.addWeatherBtn.nativeElement.addEventListener('click', () =>
this.addWeatherBlock(true)); // Add weather block
  this.runBtn.nativeElement.addEventListener('click', () => this.callMicroservice());
// Run selected microservice
  // Toggle antenna sub-options when dropdown changes
  this.antennaTypeECM.nativeElement.addEventListener('change', () =>
this.toggleAntenna('ecm'));
  this.antennaTypeRadar.nativeElement.addEventListener('change', () =>
this.toggleAntenna('radar'));
  // Default selections on load
  this.serviceSelect.nativeElement.value = 'absorption-loss';
  this.onServiceChange();     // Show appropriate form
  this.toggleAntenna('ecm');  // Show/hide ECM antenna options
  this.toggleAntenna('radar');// Show/hide radar antenna options
}
private onServiceChange(): void {
  // Hide all service forms
  const svc = this.serviceSelect.nativeElement.value;
  document.querySelectorAll<HTMLElement>('.service-form').forEach(d =>
d.style.display = 'none');
  // Show selected service form
  (document.getElementById(`form-${svc}`) as HTMLElement).style.display = 'block';
  if (svc === 'absorption-loss') {
    // If absorption-loss, allow multiple weather blocks
    this.weatherContainer.nativeElement.innerHTML = '';
    this.addWeatherBlock(true); // Add default weather block
    this.addWeatherBtn.nativeElement.style.display = 'block';
  } else {
    // Other services don't need weather input
    this.weatherContainer.nativeElement.innerHTML = '';
    this.addWeatherBtn.nativeElement.style.display = 'none';
  }
}
private addWeatherBlock(isDefault = false): void {
  // Dynamically adds a weather input block to the DOM
  const container = this.weatherContainer.nativeElement;
  const block = document.createElement('div');
  block.className = 'weather-block';
  // Weather block fields
  block.innerHTML = `
    <label>Weather Type:
      <select name="weatherType">
        <option value="clear">clear</option><option value="rain">rain</option>
        <option value="snow">snow</option><option value="fog">fog</option>
      </select>
    </label>
    <label><input type="checkbox" name="wholeGlobe"> Whole Globe</label>
    <div class="coords">
      <label>Lat Min:<input name="lat1" type="number"></label>
      <label>Lat Max:<input name="lat2" type="number"></label>
      <label>Lat Min:<input name="lat3" type="number"></label>
      <label>Lat Max:<input name="lat4" type="number"></label>
      <label>Long Min:<input name="long1" type="number"></label>
      <label>Long Max:<input name="long2" type="number"></label>
      <label>Long Min:<input name="long3" type="number"></label>
      <label>Long Max:<input name="long4" type="number"></label>
      <label>Height Max:<input name="height" type="number"></label>
    </div>
    <button type="button" class="remove-weather">Remove</button>
  `;
  // Toggle coordinate fields when 'Whole Globe' is checked
  const chk = block.querySelector<HTMLInputElement>('input[name=wholeGlobe]')!;
  const coords = block.querySelector<HTMLElement>('.coords')!;
  chk.addEventListener('change', () => coords.style.display = chk.checked ? 'none' :
'grid');
```

```
  // Remove weather block when 'Remove' is clicked
  block.querySelector<HTMLButtonElement>('.remove-weather')!
    .addEventListener('click', () => block.remove());
  // Apply default settings
  if (isDefault) {
    chk.checked = true;
    coords.style.display = 'none';
  }
  container.appendChild(block); // Add to container
}
private toggleAntenna(section: 'ecm' | 'radar'): void {
  // Show antenna sub-options only if "Gain Pattern" is selected
  const type = section === 'ecm'
    ? this.antennaTypeECM.nativeElement.value
    : this.antennaTypeRadar.nativeElement.value;
  const wrapper = section === 'ecm'
    ? this.antennaSubECM.nativeElement
    : this.antennaSubRadar.nativeElement;
  wrapper.style.display = type === 'Gain Pattern' ? 'block' : 'none';
}
private async callMicroservice(): Promise<void> {
  const svc = this.serviceSelect.nativeElement.value as keyof typeof
this.serviceEndpoints;
  const formDiv = document.getElementById(`form-${svc}`)!;

  // Collect input fields from the form
  const entries = Array.from(formDiv.querySelectorAll<HTMLInputElement |
HTMLSelectElement>('input,select'))
    .filter(el => el.id || el.name)
    .map(el => {
      if (el.tagName === 'BUTTON') return null;
      const key = el.id || el.name!;
      const val = el instanceof HTMLInputElement && el.type === 'number'
        ? +el.value : el.value;
      return [key, val];
    })
    .filter(Boolean) as [string, any][];
  // Collect weather data if using absorption-loss model
  if (svc === 'absorption-loss') {
    const weatherData: any[] = [];
  this.weatherContainer.nativeElement.querySelectorAll<HTMLElement>('.weather-block')
      .forEach(block => {
        const obj: any = {
          weatherType: (block.querySelector('select[name=weatherType]') as
HTMLSelectElement).value,
          wholeGlobe: (block.querySelector('input[name=wholeGlobe]') as
HTMLInputElement).checked
        };
        if (!obj.wholeGlobe) {
          ['lat1', 'lat2', 'long1', 'long2', 'heightMin', 'height']
            .forEach(n => obj[n] = +(block.querySelector(`input[name=${n}]`) as
HTMLInputElement).value);
        }
        weatherData.push(obj);
      });
    entries.push(['weatherConditions', weatherData]);
  }
  const data = Object.fromEntries(entries);
  // Log request
  this.logBox.nativeElement.innerHTML += `<p>[${new Date().toLocaleTimeString()}]
Calling ${svc}…</p>`;
  this.statusText.nativeElement.textContent = 'Status: Running';
  this.progressBar.nativeElement.value = 30;
  try {
    // Send POST request
    const res = await fetch(this.serviceEndpoints[svc], {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({
        parameters: {
          [svc.includes('radar') ? 'radar' : svc.includes('ecm') ? 'ecm' :
'absorption']: data
        }
      })
    });
    if (!res.ok) throw new Error(`HTTP error! status: ${res.status}`);
    const result = await res.json();
    // Save for exporting
    this.sessionRuns.push({ model: svc, input: data, output: result });
    // Handle model failure
    if (result?.error || result?.message?.toLowerCase().includes('model not
running')) {
      this.logBox.nativeElement.innerHTML += `<p>[${new Date().toLocaleTimeString()}]
Model error: not running properly.</p>`;
      this.statusText.nativeElement.textContent = 'Status: Failed';
      this.progressBar.nativeElement.value = 0;
      return;
    }
    // Success: render output
    this.statusText.nativeElement.textContent = 'Status: Completed';
    this.progressBar.nativeElement.value = 100;
    this.renderResults(result);
    this.renderMap(data);
  } catch (err: any) {
    // Handle fetch/network error
    const isConnectionRefused = err.message?.includes('ERR_CONNECTION_REFUSED') ||
err.message?.includes('Failed to fetch');
    this.logBox.nativeElement.innerHTML += `<p>[${new Date().toLocaleTimeString()}]
${isConnectionRefused ? 'Network error' : 'Unexpected error'} on ${svc}</p>`;
    this.statusText.nativeElement.textContent = 'Status: Failed';
    this.progressBar.nativeElement.value = 0;
  }
}
private renderResults(result: any): void {
  // Create container and table
  const container = document.createElement('div');
  container.className = 'vertical-result';
  const table = document.createElement('table');

  // Add key-value result rows to the table
  Object.entries(result).forEach(([k, v]) => {
    if (['id', 'parameters', 'ldashDb', 'ldashDbLog'].includes(k)) return;
    const row = table.insertRow();
    row.insertCell().textContent = k;
    row.insertCell().textContent = typeof v === 'number' ? v.toFixed(4) : String(v);
  });
  // Append table to output area
  this.chartsPlaceholder.nativeElement.innerHTML = '';
  this.chartsPlaceholder.nativeElement.appendChild(container).appendChild(table);
}
private renderMap(inputData: any): void {
  this.map3D.nativeElement.innerHTML = '';
  // Create Cesium viewer
  const viewer = new Cesium.Viewer(this.map3D.nativeElement, {
    terrainProvider: new Cesium.EllipsoidTerrainProvider(),
    sceneModePicker: true,
    timeline: false,
    animation: false,
    homeButton: true,
    fullscreenElement: this.map3D.nativeElement
  });
  const domes = [
    ['Target', 'longT', 'latT', 'heightT', Cesium.Color.RED],
    ['Radar', 'longR', 'latR', 'heightR', Cesium.Color.BLUE],
    ['Jammer', 'longJ', 'latJ', 'heightJ', Cesium.Color.ORANGE]
  ];
  // Helper to add 3D domes
  const addDomes = () => {
    viewer.entities.removeAll();
    domes.forEach(([name, lon, lat, h, color]) => {
      const L = +inputData[lon];
      const A = +inputData[lat];
      const H = +inputData[h];
      if (!isNaN(L) && !isNaN(A)) {
        const mode = viewer.scene.mode;
        const position = Cesium.Cartesian3.fromDegrees(L, A, H || 0);
        if (mode === Cesium.SceneMode.SCENE3D) {
          viewer.entities.add({
            name,
            position,
            ellipsoid: {
              radii: new Cesium.Cartesian3(2000.0, 2000.0, 1000.0),
              material: color.withAlpha(0.6),
              outline: false,
              heightReference: Cesium.HeightReference.RELATIVE_TO_GROUND
            }
          });
        } else {
          viewer.entities.add({
            name,
            position,
            ellipse: {
              semiMajorAxis: 2000.0,
              semiMinorAxis: 2000.0,
              material: color.withAlpha(0.6),
              height: H || 0
            }
          });
        }
      }
    });
  };
  // Initial dome render
  addDomes();
  // Zoom to view domes
  viewer.zoomTo(viewer.entities).then(() => {
    const homeCameraView = {
      destination: viewer.camera.positionWC.clone(),
      orientation: {
        heading: viewer.camera.heading,
        pitch: viewer.camera.pitch,
        roll: viewer.camera.roll
      }
    };
    // Override home button to reset to this view
    viewer.homeButton.viewModel.command.beforeExecute.addEventListener((commandInfo:
{ cancel: boolean; }) => {
      viewer.camera.setView(homeCameraView);
      commandInfo.cancel = true;
    });
  });
  // Re-render domes after changing scene mode (2D <-> 3D)
  viewer.scene.morphComplete.addEventListener(() => {
    addDomes();
  });
}
exportToPDF(): void {
  const doc = new jsPDF();
  let y = 10;
  // Set title
  doc.setFontSize(14);
  doc.text('Simulation Summary', 10, y);
  y += 10;
  // Iterate over each simulation run
  this.sessionRuns.forEach((run, index) => {
    // Add run heading
    doc.setFontSize(12);
    doc.text(`Run ${index + 1}: Model - ${run.model}`, 10, y);
    y += 8;
    // Input parameters section
    doc.setFontSize(10);
    doc.text('Input Parameters:', 10, y);
    y += 6;
    Object.entries(run.input).forEach(([k, v]) => {
      const line = `${k}: ${typeof v === 'object' ? JSON.stringify(v) : v}`;
      doc.text(line, 15, y);
      y += 6;
      // Start a new page if content exceeds page height
      if (y > 280) { doc.addPage(); y = 10; }
    });
    // Output parameters section
    doc.text('Output Results:', 10, y);
    y += 6;
    Object.entries(run.output).forEach(([k, v]) => {
      // Skip unnecessary keys
      if (['id', 'parameters', 'ldashDb', 'ldashDbLog'].includes(k)) return;
      const line = `${k}: ${typeof v === 'number' ? v.toFixed(4) : String(v)}`;
      doc.text(line, 15, y);
      y += 6;
      if (y > 280) { doc.addPage(); y = 10; }
    });
    // Add space before the next run
    y += 10;
    if (y > 280) { doc.addPage(); y = 10; }
  });
  // Save the generated PDF
  doc.save('dashboard.pdf');
}
exportToCSV(): void {
  const rows: string[] = [];
  // CSV header
  rows.push('Run,Type,Key,Value');
  // Process each session run
  this.sessionRuns.forEach((run, index) => {
    // Add input parameters
    Object.entries(run.input).forEach(([k, v]) => {
      const val = typeof v === 'object' ? JSON.stringify(v) : v;
      rows.push(`${index + 1},Input,${k},${val}`);
    });
    // Add output results
    Object.entries(run.output).forEach(([k, v]) => {
      // Skip unnecessary fields
      if (['id', 'parameters', 'ldashDb', 'ldashDbLog'].includes(k)) return;
      const val = typeof v === 'number' ? v.toFixed(4) : String(v);
      rows.push(`${index + 1},Output,${k},${val}`);
    });
  });
  // Create and trigger download of CSV file
  const blob = new Blob([rows.join('\r\n')], { type: 'text/csv' });
  const url = URL.createObjectURL(blob);
  const a = document.createElement('a');
  a.href = url;
  a.download = 'dashboard.csv';
  a.click();
  URL.revokeObjectURL(url); // Clean up
}
}
```