

HMT-Project-Roadmap (Health Metrics Tracker)

Purpose: Track, aggregate, and visualize health indicators across facilities - similar to DHIS2's core functionality but focused scope.

Tech Stack: Java 21, Spring Boot 3.x, PostgreSQL, React 18, REST API

PHASE 0: Environment Setup & Repository Initialization

Step 0.1

Install Required Tools

- Install Java JDK 21 (Amazon Corretto or OpenJDK)
- Install IntelliJ IDEA Community Edition
- Install Node.js 20 LTS
- Install PostgreSQL 16
- Install Postman for API testing
- Verify installations: `java -version`, `node -v`, `psql --version`

Step 0.2

Create GitHub Repository

```
# Create new repo on GitHub: health-metrics-tracker
# Clone locally
git clone https://github.com/JoanKarantourou/health-metrics-tracker.git
cd health-metrics-tracker

# Create initial structure
mkdir backend frontend docs
touch README.md .gitignore

# Create comprehensive .gitignore
echo "# Java
*.class
*.jar
*.war
target/
.idea/
*.iml

# Node
node_modules/
build/
```

```
dist/
.env

# Database
*.db
*.sql.backup

# OS
.DS_Store
Thumbs.db" > .gitignore

# Initial commit
git add .
git commit -m "Initial repository structure"
git push origin main
```

Step 0.3

Create Project Documentation

Create `README.md` with:

- Project description
- Tech stack
- Setup instructions (placeholder)
- Architecture overview (placeholder)
- API documentation link (placeholder)

```
git add README.md
git commit -m "Add initial README"
git push origin main
```

PHASE 1: Backend - Spring Boot Foundation

Step 1.1

Initialize Spring Boot Project

- Go to <https://start.spring.io/>
- Configuration:
 - Project: Maven
 - Language: Java
 - Spring Boot: 3.2.x (latest stable)
 - Java: 21
 - Packaging: Jar
 - Group: com.healthmetrics

- Artifact: tracker
- Dependencies: Spring Web, Spring Data JPA, PostgreSQL Driver, Lombok, Validation, Spring Security, Spring Boot DevTools
- Download and extract to `backend/` folder
- Open in IntelliJ IDEA

```
cd backend
# Test the application runs
./mvnw spring-boot:run
# Should start on port 8080

git add .
git commit -m "Initialize Spring Boot project with core dependencies"
git push origin main
```

Step 1.2

Configure Database Connection

Create `backend/src/main/resources/application.yml`:

```
spring:
  application:
    name: health-metrics-tracker
  datasource:
    url: jdbc:postgresql://localhost:5432/health_metrics_db
    username: postgres
    password: your_password
  jpa:
    hibernate:
      ddl-auto: update
    show-sql: true
    properties:
      hibernate:
        format_sql: true
  server:
    port: 8080
```

Create database:

```
-- Run in PostgreSQL
CREATE DATABASE health_metrics_db;
```

```
git add .
git commit -m "Configure PostgreSQL database connection"
git push origin main
```

Step 1.3

Create Base Package Structure

```
backend/src/main/java/com/healthmetrics/tracker/  
├─ config/  
├─ controller/  
├─ dto/  
├─ entity/  
├─ exception/  
├─ repository/  
├─ service/  
└─ util/
```

Create empty `.gitkeep` files in each directory.

```
git add .  
git commit -m "Create backend package structure"  
git push origin main
```

PHASE 2: Core Domain Model - Health Facilities & Indicators

Step 2.1

Create Facility Entity

File: `backend/src/main/java/com/healthmetrics/tracker/entity/Facility.java`

```
@Entity  
@Table(name = "facilities")  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
public class Facility {  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long id;  
  
    @Column(nullable = false, unique = true)  
    private String code;  
  
    @Column(nullable = false)  
    private String name;  
  
    @Column(nullable = false)  
    private String type; // Hospital, Clinic, Health Center  
  
    private String region;  
    private String district;
```

```

    private Double latitude;
    private Double longitude;

    @Column(nullable = false)
    private Boolean active = true;

    @CreatedDate
    private LocalDateTime createdAt;

    @LastModifiedDate
    private LocalDateTime updatedAt;
}

```

Create corresponding Repository:

File: `repository/FacilityRepository.java`

```

@Repository
public interface FacilityRepository extends JpaRepository<Facility, Long>
{
    Optional<Facility> findByCode(String code);
    List<Facility> findByRegion(String region);
    List<Facility> findByActive(Boolean active);
}

```

```

git add .
git commit -m "Create Facility entity and repository"
git push origin main

```

Step 2.2

Create HealthIndicator Entity

File: `entity/HealthIndicator.java`

```

@Entity
@Table(name = "health_indicators")
@Data
@NoArgsConstructor
@AllArgsConstructor
public class HealthIndicator {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false, unique = true)
    private String code;

    @Column(nullable = false)
    private String name;
}

```

```

    private String description;

    @Column(nullable = false)
    private String category; // Maternal Health, Child Health, Disease, et
c.

    @Column(nullable = false)
    private String dataType; // NUMBER, PERCENTAGE, BOOLEAN

    private String unit; // cases, %, persons

    @Column(nullable = false)
    private Boolean active = true;
}

```

Create Repository:

File: `repository/HealthIndicatorRepository.java`

```

git add .
git commit -m "Create HealthIndicator entity and repository"
git push origin main

```

Step 2.3

Create DataValue Entity (Core Data Model)

File: `entity/DataValue.java`

```

@Entity
@Table(name = "data_values",
        uniqueConstraints = @UniqueConstraint(
            columnNames = {"facility_id", "indicator_id", "period_start"}
        ))
@Data
@NoArgsConstructor
@AllArgsConstructor
public class DataValue {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "facility_id", nullable = false)
    private Facility facility;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "indicator_id", nullable = false)
    private HealthIndicator indicator;
}

```

```

@Column(nullable = false)
private LocalDate periodStart;

@Column(nullable = false)
private LocalDate periodEnd;

@Column(nullable = false)
private String periodType; // DAILY, WEEKLY, MONTHLY, QUARTERLY, YEARL
Y

@Column(nullable = false)
private BigDecimal value;

private String comment;

@CreatedDate
private LocalDateTime createdAt;

@LastModifiedDate
private LocalDateTime updatedAt;

private String createdBy;
}

```

Create Repository with custom queries:

File: `repository/DataValueRepository.java`

```

@Repository
public interface DataValueRepository extends JpaRepository<DataValue, Long> {
    List<DataValue> findByFacilityId(Long facilityId);

    List<DataValue> findByIndicatorId(Long indicatorId);

    @Query("SELECT dv FROM DataValue dv WHERE dv.facility.id = :facilityId " +
        "AND dv.periodStart >= :startDate AND dv.periodEnd <= :endDate")
    List<DataValue> findByFacilityAndPeriod(
        @Param("facilityId") Long facilityId,
        @Param("startDate") LocalDate startDate,
        @Param("endDate") LocalDate endDate
    );

    @Query("SELECT dv FROM DataValue dv WHERE dv.indicator.id = :indicatorId " +
        "AND dv.facility.region = :region ")

```

```

        "AND dv.periodStart >= :startDate")
    List<DataValue> findByIndicatorAndRegion(
        @Param("indicatorId") Long indicatorId,
        @Param("region") String region,
        @Param("startDate") LocalDate startDate
    );
}

```

```

git add .
git commit -m "Create DataValue entity with relationships and custom queries"
git push origin main

```

PHASE 3: DTOs and Service Layer

Step 3.1

Create DTOs for API Responses

File: `dto/FacilityDTO.java`
 File: `dto/HealthIndicatorDTO.java`
 File: `dto/DataValueDTO.java`
 File: `dto/DataValueCreateRequest.java`
 File: `dto/ApiResponse.java` (generic wrapper)

```

git add .
git commit -m "Create DTO classes for API communication"
git push origin main

```

Step 3.2

Create FacilityService

File: `service/FacilityService.java`

```

@Service
@RequiredArgsConstructor
public class FacilityService {
    private final FacilityRepository facilityRepository;

    public List<FacilityDTO> getAllFacilities() {
        return facilityRepository.findAll().stream()
            .map(this::toDTO)
            .collect(Collectors.toList());
    }

    public FacilityDTO getFacilityById(Long id) {
        return facilityRepository.findById(id)
            .map(this::toDTO)

```

```

        .orElseThrow(() -> new ResourceNotFoundException("Facility not
found"));
    }

    public FacilityDTO createFacility(FacilityDTO dto) {
        // Validation logic
        Facility facility = toEntity(dto);
        Facility saved = facilityRepository.save(facility);
        return toDTO(saved);
    }

    // Implement: update, delete, findByRegion methods
    // Implement: toDTO, toEntity mapping methods
}

```

```

git add .
git commit -m "Implement FacilityService with CRUD operations"
git push origin main

```

Step 3.3

Create HealthIndicatorService

Similar structure to FacilityService.

```

git add .
git commit -m "Implement HealthIndicatorService"
git push origin main

```

Step 3.4

Create DataValueService with Aggregation Logic

File: `service/DataValueService.java`

```

@Service
@RequiredArgsConstructor
public class DataValueService {
    private final DataValueRepository dataValueRepository;
    private final FacilityRepository facilityRepository;
    private final HealthIndicatorRepository indicatorRepository;

    public DataValueDTO submitDataValue(DataValueCreateRequest request) {
        // Validate facility exists
        // Validate indicator exists
        // Check for duplicates
        // Save and return
    }

    public List<DataValueDTO> getDataValuesByFacility(

```

```

        Long facilityId,
        LocalDate startDate,
        LocalDate endDate
    ) {
        // Implementation
    }

    public Map<String, BigDecimal> aggregateByRegion(
        Long indicatorId,
        String periodType,
        LocalDate startDate,
        LocalDate endDate
    ) {
        // Group by region and sum/average values
        // Return aggregated data
    }

    // More aggregation methods for dashboard
}

```

```

git add .
git commit -m "Implement DataValueService with aggregation logic"
git push origin main

```

PHASE 4: REST API Controllers

Step 4.1

Create Global Exception Handler

File: `exception/GlobalExceptionHandler.java`

File: `exception/ResourceNotFoundException.java`

File: `exception/ValidationException.java`

```

git add .
git commit -m "Implement global exception handling"
git push origin main

```

Step 4.2

Create FacilityController

File: `controller/FacilityController.java`

```

@RestController
@RequestMapping("/api/facilities")
@RequiredArgsConstructor
@CrossOrigin(origins = "http://localhost:3000")
public class FacilityController {

```

```

private final FacilityService facilityService;

@GetMapping
public ResponseEntity<List<FacilityDTO>> getAllFacilities() {
    return ResponseEntity.ok(facilityService.getAllFacilities());
}

@GetMapping("/{id}")
public ResponseEntity<FacilityDTO> getFacilityById(@PathVariable Long
id) {
    return ResponseEntity.ok(facilityService.getFacilityById(id));
}

@PostMapping
public ResponseEntity<FacilityDTO> createFacility(
    @Valid @RequestBody FacilityDTO facilityDTO
) {
    return ResponseEntity.status(HttpStatus.CREATED)
        .body(facilityService.createFacility(facilityDTO));
}

// Implement: PUT, DELETE, GET by region endpoints
}

```

Test with Postman:

- GET <http://localhost:8080/api/facilities>
- POST <http://localhost:8080/api/facilities>

```

git add .
git commit -m "Implement FacilityController REST API"
git push origin main

```

Step 4.3

Create HealthIndicatorController

```

git add .
git commit -m "Implement HealthIndicatorController REST API"
git push origin main

```

Step 4.4

Create DataValueController

File: `controller/DataValueController.java`

```

@RestController
@RequestMapping("/api/data-values")

```

```

@RequiredArgsConstructor
@CrossOrigin(origins = "http://localhost:3000")
public class DataValueController {

    private final DataValueService dataValueService;

    @PostMapping
    public ResponseEntity<DataValueDTO> submitData(
        @Valid @RequestBody DataValueCreateRequest request
    ) {
        return ResponseEntity.status(HttpStatus.CREATED)
            .body(dataValueService.submitDataValue(request));
    }

    @GetMapping("/facility/{facilityId}")
    public ResponseEntity<List<DataValueDTO>> getByFacility(
        @PathVariable Long facilityId,
        @RequestParam @DateTimeFormat(iso = ISO.DATE) LocalDate startDate,
        @RequestParam @DateTimeFormat(iso = ISO.DATE) LocalDate endDate
    ) {
        return ResponseEntity.ok(
            dataValueService.getDataValuesByFacility(facilityId, startDate, endDate)
        );
    }

    @GetMapping("/aggregate/region")
    public ResponseEntity<Map<String, BigDecimal>> aggregateByRegion(
        @RequestParam Long indicatorId,
        @RequestParam String periodType,
        @RequestParam @DateTimeFormat(iso = ISO.DATE) LocalDate startDate,
        @RequestParam @DateTimeFormat(iso = ISO.DATE) LocalDate endDate
    ) {
        return ResponseEntity.ok(
            dataValueService.aggregateByRegion(indicatorId, periodType, startDate, endDate)
        );
    }
}

```

```

git add .
git commit -m "Implement DataValueController with aggregation endpoints"
git push origin main

```

PHASE 5: Database Seeding & Testing

Step 5.1

Create Database Initialization Script

File: `backend/src/main/resources/data.sql`

```
-- Sample facilities
INSERT INTO facilities (code, name, type, region, district, active, create
d_at) VALUES
('FAC001', 'Athens General Hospital', 'Hospital', 'Attica', 'Athens', tru
e, NOW()),
('FAC002', 'Thessaloniki Health Center', 'Health Center', 'Central Macedon
ia', 'Thessaloniki', true, NOW()),
-- Add 10-15 more facilities
;

-- Sample health indicators
INSERT INTO health_indicators (code, name, category, data_type, unit, acti
ve) VALUES
('IND001', 'Malaria Cases', 'Disease', 'NUMBER', 'cases', true),
('IND002', 'Vaccination Coverage', 'Child Health', 'PERCENTAGE', '%', tru
e),
-- Add 10 more indicators
;
```

Create Java seeder class:

File: `config/DataSeeder.java`

```
@Component
@RequiredArgsConstructor
public class DataSeeder implements ApplicationRunner {

    private final FacilityRepository facilityRepository;
    private final HealthIndicatorRepository indicatorRepository;
    private final DataValueRepository dataValueRepository;

    @Override
    public void run(ApplicationArguments args) {
        if (facilityRepository.count() == 0) {
            seedFacilities();
            seedIndicators();
            seedSampleData();
        }
    }

    // Implement seed methods with realistic data
}
```

```
git add .
git commit -m "Add database seeding with sample data"
git push origin main
```

Step 5.2

API Testing Documentation

Create `docs/API_TESTING.md` with:

- All endpoints
- Sample Postman requests
- Expected responses
- cURL commands

Create Postman Collection JSON:

File: `docs/Health-Metrics-Tracker.postman_collection.json`

```
git add .
git commit -m "Add API testing documentation and Postman collection"
git push origin main
```

PHASE 6: Frontend - React Application

Step 6.1

Initialize React Application

```
cd frontend
npx create-react-app client
cd client

# Install dependencies
npm install axios react-router-dom recharts date-fns
npm install @mui/material @mui/icons-material @emotion/react @emotion/styled
npm install react-query

# Start development server to verify
npm start

cd ../../
git add .
git commit -m "Initialize React application with dependencies"
git push origin main
```

Step 6.2

Create Project Structure

```
frontend/client/src/  
├── components/  
│   ├── common/  
│   ├── facilities/  
│   ├── indicators/  
│   └── dataentry/  
├── pages/  
├── services/  
├── hooks/  
├── utils/  
└── App.js
```

```
git add .  
git commit -m "Create React project structure"  
git push origin main
```

Step 6.3

Create API Service

File: `frontend/client/src/services/api.js`

```
import axios from 'axios';  
  
const API_BASE_URL = 'http://localhost:8080/api';  
  
const apiClient = axios.create({  
  baseURL: API_BASE_URL,  
  headers: {  
    'Content-Type': 'application/json',  
  },  
});  
  
export const facilityService = {  
  getAll: () => apiClient.get('/facilities'),  
  getById: (id) => apiClient.get(`/facilities/${id}`),  
  create: (data) => apiClient.post('/facilities', data),  
  update: (id, data) => apiClient.put(`/facilities/${id}`, data),  
};  
  
export const indicatorService = {  
  getAll: () => apiClient.get('/indicators'),  
  getById: (id) => apiClient.get(`/indicators/${id}`),  
};  
  
export const dataValueService = {  
  submit: (data) => apiClient.post('/data-values', data),  
};
```

```

getByFacility: (facilityId, startDate, endDate) =>
  apiClient.get(`/data-values/facility/${facilityId}`, {
    params: { startDate, endDate }
  }),
aggregateByRegion: (indicatorId, periodType, startDate, endDate) =>
  apiClient.get(`/data-values/aggregate/region`, {
    params: { indicatorId, periodType, startDate, endDate }
  }),
};

```

```

git add .
git commit -m "Create API service layer for React"
git push origin main

```

Step 6.4

Create Facilities List Component

File: `frontend/client/src/components/facilities/FacilityList.jsx`

```

import React, { useEffect, useState } from 'react';
import { facilityService } from '../../services/api';

function FacilityList() {
  const [facilities, setFacilities] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    const fetchFacilities = async () => {
      try {
        const response = await facilityService.getAll();
        setFacilities(response.data);
      } catch (err) {
        setError(err.message);
      } finally {
        setLoading(false);
      }
    };

    fetchFacilities();
  }, []);

  if (loading) return <div>Loading facilities...</div>;
  if (error) return <div>Error: {error}</div>;

  return (
    <div className="facility-list">
      <h2>Health Facilities</h2>

```

```

<table>
  <thead>
    <tr>
      <th>Code</th>
      <th>Name</th>
      <th>Type</th>
      <th>Region</th>
      <th>District</th>
      <th>Actions</th>
    </tr>
  </thead>
  <tbody>
    {facilities.map(facility => (
      <tr key={facility.id}>
        <td>{facility.code}</td>
        <td>{facility.name}</td>
        <td>{facility.type}</td>
        <td>{facility.region}</td>
        <td>{facility.district}</td>
        <td>
          <button>View</button>
          <button>Edit</button>
        </td>
      </tr>
    ))}
  </tbody>
</table>
</div>
);
}

```

```
export default FacilityList;
```

```

git add .
git commit -m "Create FacilityList component with API integration"
git push origin main

```

Step 6.5

Create Data Entry Form

File: `frontend/client/src/components/dataentry/DataEntryForm.jsx`

```

import React, { useState, useEffect } from 'react';
import { facilityService, indicatorService, dataValueService } from
'../../services/api';

function DataEntryForm() {
  const [facilities, setFacilities] = useState([]);

```

```

const [indicators, setIndicators] = useState([]);
const [formData, setFormData] = useState({
  facilityId: '',
  indicatorId: '',
  periodStart: '',
  periodEnd: '',
  periodType: 'MONTHLY',
  value: '',
  comment: ''
});

useEffect(() => {
  // Fetch facilities and indicators
  const fetchData = async () => {
    const [facilitiesRes, indicatorsRes] = await Promise.all([
      facilityService.getAll(),
      indicatorService.getAll()
    ]);
    setFacilities(facilitiesRes.data);
    setIndicators(indicatorsRes.data);
  };
  fetchData();
}, []);

const handleSubmit = async (e) => {
  e.preventDefault();
  try {
    await dataValueService.submit(formData);
    alert('Data submitted successfully!');
    // Reset form
  } catch (error) {
    alert('Error submitting data: ' + error.message);
  }
};

return (
  <form onSubmit={handleSubmit}>
    <h2>Data Entry</h2>
    { /* Form fields */ }
    <button type="submit">Submit Data</button>
  </form>
);
}

export default DataEntryForm;

```

```
git add .
git commit -m "Create data entry form component"
git push origin main
```

Step 6.6

Create Dashboard with Charts

File: `frontend/client/src/pages/Dashboard.jsx`

Use `recharts` library to create:

- Line chart showing indicator trends over time
- Bar chart for regional comparisons
- Summary cards with key metrics

```
git add .
git commit -m "Create dashboard with data visualization"
git push origin main
```

Step 6.7

Create Routing & Navigation

File: `frontend/client/src/App.js`

```
import { BrowserRouter as Router, Routes, Route, Link } from 'react-router-dom';
import Dashboard from './pages/Dashboard';
import FacilityList from './components/facilities/FacilityList';
import DataEntryForm from './components/dataentry/DataEntryForm';

function App() {
  return (
    <Router>
      <nav>
        <ul>
          <li><Link to="/">Dashboard</Link></li>
          <li><Link to="/facilities">Facilities</Link></li>
          <li><Link to="/data-entry">Data Entry</Link></li>
        </ul>
      </nav>

      <Routes>
        <Route path="/" element={<Dashboard />} />
        <Route path="/facilities" element={<FacilityList />} />
        <Route path="/data-entry" element={<DataEntryForm />} />
      </Routes>
    </Router>
  );
}
```

```
}
```

```
export default App;
```

```
git add .  
git commit -m "Implement routing and navigation"  
git push origin main
```

PHASE 7: Advanced Features

Step 7.1

Add Filtering & Search

Backend: Add query parameters to controllers

Frontend: Add search bars and filter dropdowns

```
git add .  
git commit -m "Implement filtering and search functionality"  
git push origin main
```

Step 7.2

Add Pagination

Backend: Use Spring Data's `Pageable`

Frontend: Add pagination controls

```
git add .  
git commit -m "Implement pagination for large datasets"  
git push origin main
```

Step 7.3

Add Data Export

Backend: Create CSV export endpoint

Frontend: Add export button

```
git add .  
git commit -m "Add CSV data export feature"  
git push origin main
```

Step 7.4

Add Data Validation Rules

Backend: Custom validators for data values

Frontend: Real-time validation feedback

```
git add .
git commit -m "Implement comprehensive data validation"
git push origin main
```

PHASE 8: Testing

Step 8.1

Backend Unit Tests

File: `backend/src/test/java/com/healthmetrics/tracker/service/FacilityServiceTest.java`

```
@SpringBootTest
class FacilityServiceTest {

    @Autowired
    private FacilityService facilityService;

    @MockBean
    private FacilityRepository facilityRepository;

    @Test
    void testGetAllFacilities() {
        // Arrange
        List<Facility> mockFacilities = Arrays.asList(
            new Facility(/* ... */)
        );
        when(facilityRepository.findAll()).thenReturn(mockFacilities);

        // Act
        List<FacilityDTO> result = facilityService.getAllFacilities();

        // Assert
        assertEquals(1, result.size());
        verify(facilityRepository, times(1)).findAll();
    }
}
```

Write tests for all services.

```
git add .
git commit -m "Add backend unit tests for services"
git push origin main
```

Step 8.2

Backend Integration Tests

Test controllers with MockMvc.

```
git add .
git commit -m "Add backend integration tests"
git push origin main
```

Step 8.3

Frontend Component Tests

Use React Testing Library.

```
git add .
git commit -m "Add frontend component tests"
git push origin main
```

PHASE 9: Documentation & Deployment Prep

Step 9.1

Complete README

Update main `README.md` with:

- Detailed project description
- Complete setup instructions
- Architecture diagram
- API documentation
- Screenshots
- Technologies explained

```
git add .
git commit -m "Complete comprehensive README documentation"
git push origin main
```

Step 9.2

Create API Documentation

File: `docs/API.md` with all endpoints documented

Consider adding Swagger/OpenAPI:

```
// Add dependency: springdoc-openapi-starter-webmvc-ui
// Access at: http://localhost:8080/swagger-ui.html
```

```
git add .
git commit -m "Add comprehensive API documentation"
git push origin main
```

Step 9.3

Create Architecture Documentation

File: `docs/ARCHITECTURE.md` explaining:

- System architecture
- Database schema
- Design decisions
- Technology choices

```
git add .
git commit -m "Add architecture documentation"
git push origin main
```

Step 9.4

Add Deployment Instructions

File: `docs/DEPLOYMENT.md` for:

- Docker containerization
- Environment configuration
- Production considerations

Create `docker-compose.yml`:

```
version: '3.8'
services:
  postgres:
    image: postgres:16
    environment:
      POSTGRES_DB: health_metrics_db
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: password
    ports:
      - "5432:5432"

  backend:
    build: ./backend
    ports:
      - "8080:8080"
    depends_on:
      - postgres

  frontend:
    build: ./frontend/client
    ports:
      - "3000:3000"
```

```
depends_on:  
  - backend
```

```
git add .  
git commit -m "Add Docker deployment configuration"  
git push origin main
```

Step 9.5

Create Screenshots & Demo

- Take screenshots of all major features
- Create animated GIFs of key workflows
- Add to README

```
git add .  
git commit -m "Add project screenshots and demo materials"  
git push origin main
```

PHASE 10: Polish & Best Practices

Step 10.1

Code Quality

- Run code formatters (Prettier, Checkstyle)
- Fix all linting warnings
- Ensure consistent naming conventions

```
git add .  
git commit -m "Apply code formatting and quality improvements"  
git push origin main
```

Step 10.2

Security Enhancements

- Add input sanitization
- Implement CORS properly
- Add rate limiting
- Security headers

```
git add .  
git commit -m "Implement security best practices"  
git push origin main
```

Step 10.3

Performance Optimization

- Add database indexes
- Implement caching where appropriate
- Optimize queries
- Frontend lazy loading

```
git add .  
git commit -m "Optimize application performance"  
git push origin main
```

Step 10.4

Error Handling Enhancement

- Comprehensive error messages
- Proper HTTP status codes
- User-friendly error pages

```
git add .  
git commit -m "Enhance error handling across application"  
git push origin main
```

PHASE 11: Final Touches

Step 11.1

Create **CONTRIBUTING.md**

Guidelines for future contributions.

```
git add .  
git commit -m "Add contributing guidelines"  
git push origin main
```

Step 11.2

Add **LICENSE**

Choose and add appropriate license (MIT recommended).

```
git add .  
git commit -m "Add MIT license"  
git push origin main
```

Step 11.3

Create Comprehensive Test Data

Ensure database has realistic, diverse test data.

```
git add .
git commit -m "Add comprehensive test dataset"
git push origin main
```

Step 11.4

Final README Polish

- Add badges (build status, license)
- Add table of contents
- Add "Future Enhancements" section
- Add contact information

```
git add .
git commit -m "Final README polish and enhancements"
git push origin main
```

Step 11.5

Create GitHub Release

- Tag version v1.0.0
- Create release notes
- Document all features

```
git tag -a v1.0.0 -m "Initial release - Health Metrics Tracker"
git push origin v1.0.0
```

Success Metrics

After completion, your project will demonstrate:

- ✓ **Java/Spring Boot Mastery:** RESTful APIs, JPA, dependency injection
- ✓ **Database Skills:** PostgreSQL with complex queries and relationships
- ✓ **React Development:** Component architecture, state management, API integration
- ✓ **Full-Stack Integration:** Complete end-to-end application
- ✓ **Domain Knowledge:** Health information systems (DHIS2-relevant)
- ✓ **Best Practices:** Testing, documentation, Git workflow
- ✓ **Production Ready:** Error handling, validation, deployment config