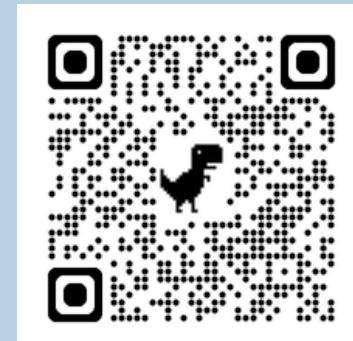


# Web Bluetooth Medical Dashboard

from physical hardware to digital data

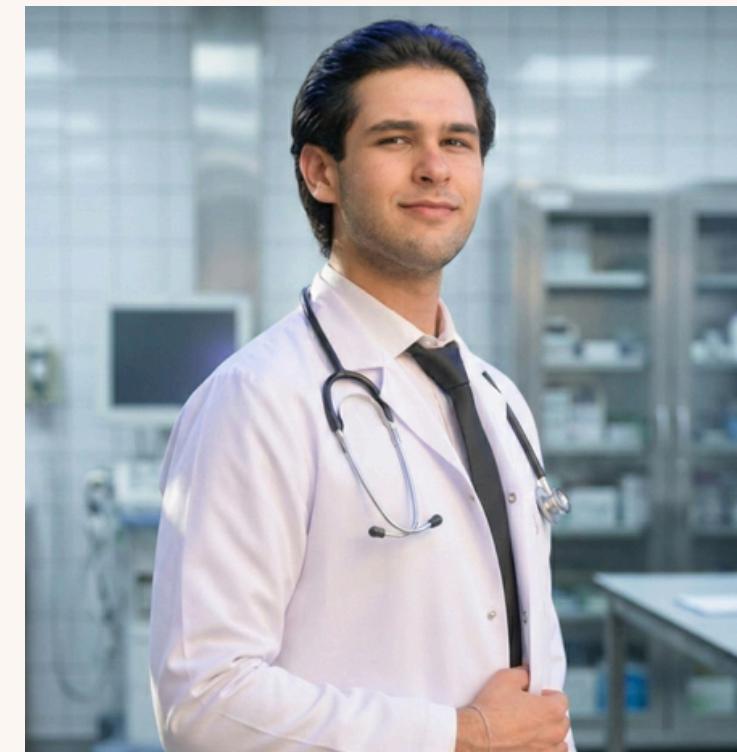
Github repository



# Our medical team: Dedicated professionals



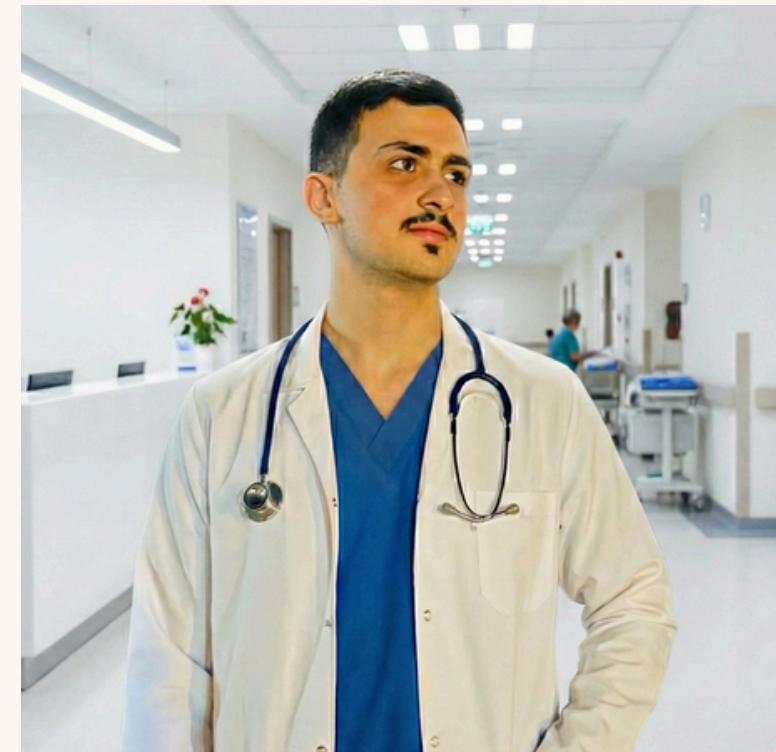
**George Khayat**  
**22232005**



**Elia Ghazal**  
**22330018**



**William Ishak**  
**22232003**



**Bassam Farhat**  
**22330047**

# Problem Statement

| Platform Fragmentation   | Installation Friction  |
|--|--|
| Native apps per OS<br>(iOS/Android/Windows) → more dev work + inconsistent UX. | Users must find, download, install → lower adoption (especially occasional use). |



# Solution: Web Bluetooth API

01.

Cross-platform compatibility.

02.

Zero-installation access via  
URL.

03.

Real-time physiological  
data acquisition.

# System Architecture

## Presentation Layer

- HTML5/Razor views
- CSS3
- jQuery for dynamic UI updates.

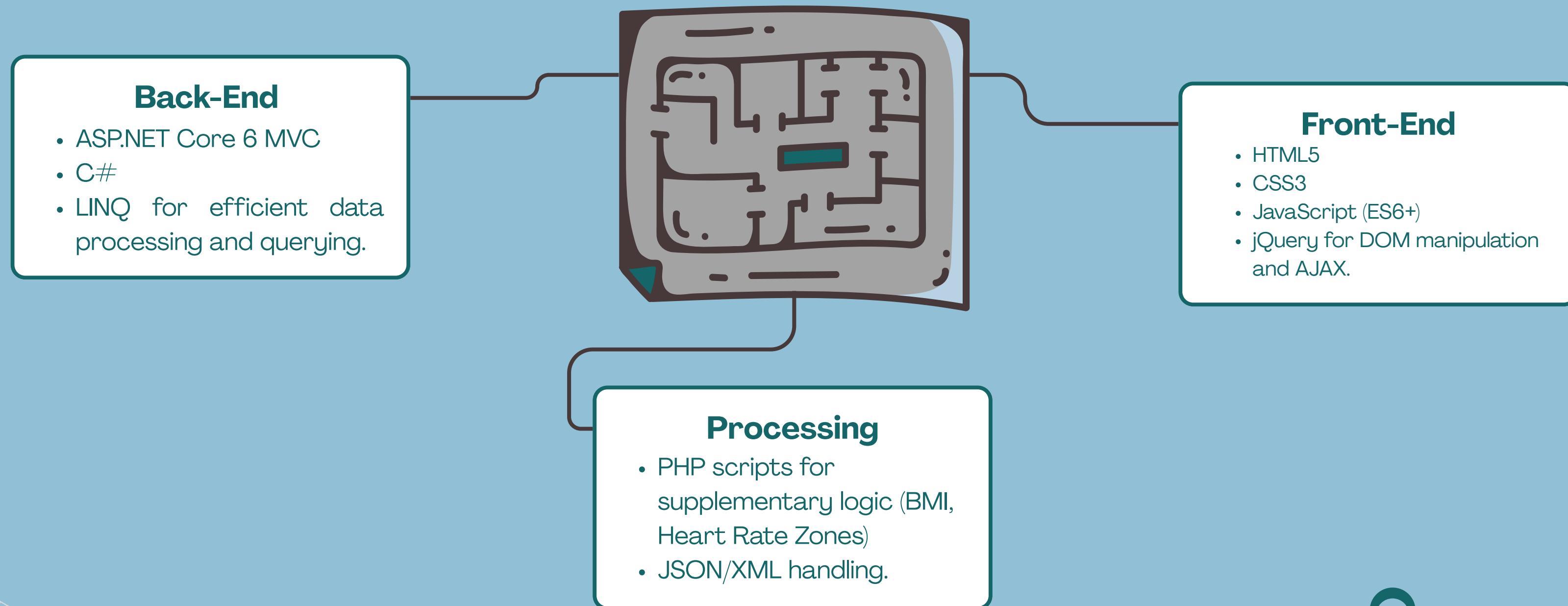
## Application Layer

- ASP.NET Core 6 MVC controller handling logic and routing.

## Data Layer

- In-memory storage processed via LINQ queries and exported via XML/JSON.

# Technology Stack



# Core Feature: Web Bluetooth Integration

## Workflow:

1. User Gesture (Click)
2. `navigator.bluetooth.requestDevice()`
3. Connect to GATT Server
4. Subscribe to Characteristics (e.g., `0x2A37`)

```
// A. User Gesture (Click Event)
$('#connectHeartRate').on('click', async function() {

    // B. Request Device
    const device = await navigator.bluetooth.requestDevice({
        filters: [{ services: ['heart_rate'] }] // Standard Heart Rate Service
    });

    // C. Connect to GATT Server
    const server = await device.gatt.connect();

    // D. Subscribe to Characteristics
    const service = await server.getPrimaryService('heart_rate');
    const characteristic = await service.getCharacteristic('heart_rate_measurement'); // UUID:
0x2A37
    await characteristic.startNotifications();
    characteristic.addEventListener('characteristicvaluechanged', handleNotifications);
});
```

# Data Processing With LINQ

We utilize C# LINQ (Language Integrated Query) for powerful in-memory data manipulation, ensuring type safety and code readability.

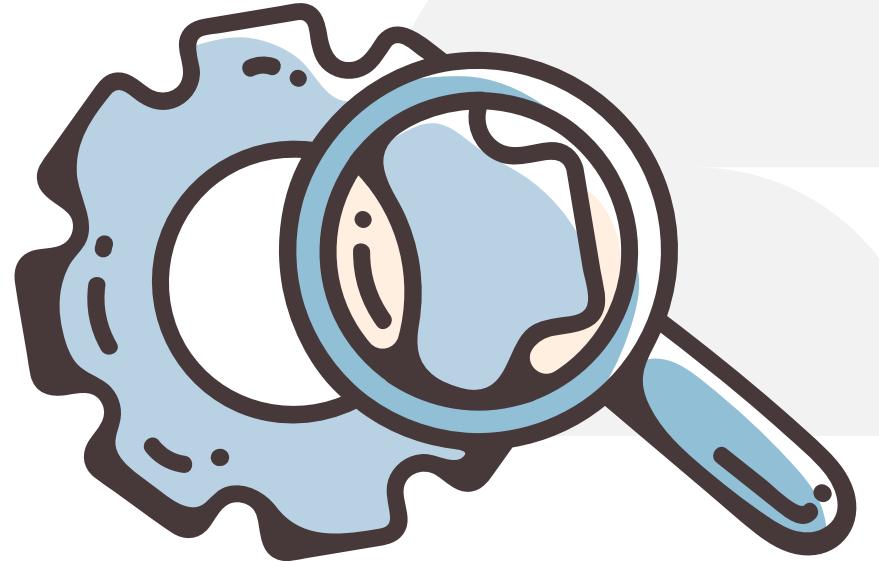
```
// 1. Filtering by Device Type (Where)
public List<HealthReadingDto> GetReadingsByDeviceType(string deviceType)
{
    return _readings
        .Where(r => r.DeviceType.Equals(deviceType,
StringComparison.OrdinalIgnoreCase))
        .ToList();
}

// 2. Real-time Aggregation (GroupBy)
public Dictionary<string, List<HealthReadingDto>> GetReadingsGroupedByDevice()
{
    return _readings
        .GroupBy(r => r.DeviceId)
        .ToDictionary(g => g.Key, g => g.OrderByDescending(r => r.Timestamp).ToList());
}

// 3. Calculating Statistical Averages (Average)
public Dictionary<string, double> GetAveragesByType()
{
    return _readings
        .GroupBy(r => r.DeviceType)
        .Where(g => g.Any())
        .ToDictionary(g => g.Key, g => g.Average(r => r.Value));
}
```

# User Interface & Experience





## Real-Time Feedback

The dashboard employs jQuery animations to provide immediate visual feedback (pulse effects) when new data packets arrive.

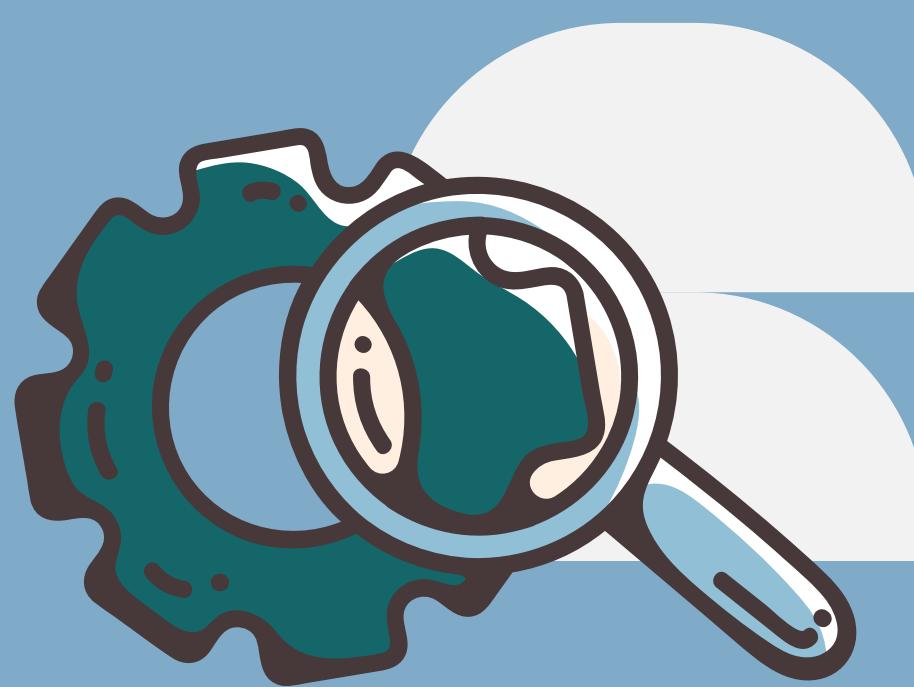
### Define the Pulse Animation (CSS)

```
@keyframes pulse {  
  0% { transform: scale(1); box-shadow: 0 5px 20px rgba(0,0,0,0.2); }  
  50% { transform: scale(1.05); box-shadow: 0 10px 40px rgba(0,0,0,0.3); }  
  100% { transform: scale(1); box-shadow: 0 5px 20px rgba(0,0,0,0.2); }  
  
.pulse-animation {  
  animation: pulse 0.5s ease-in-out;  
}
```

### Trigger it with jQuery when data arrives (JS)

```
function handleHeartRateData(value) {  
  // Update text  
  $('#heartRateValue').text(value);  
  
  // Trigger Animation  
  $('#heartRateCard').addClass('pulse-animation');  
  
  // Remove class after animation finishes so it can be triggered again  
  setTimeout(() => $('#heartRateCard').removeClass('pulse-animation'),  
            500);
```





# Responsive Design

Built with Bootstrap 5 and custom CSS gradients, the interface adapts seamlessly to various screen sizes, maintaining usability across desktop and mobile devices.

## Bootstrap Grid System (HTML)

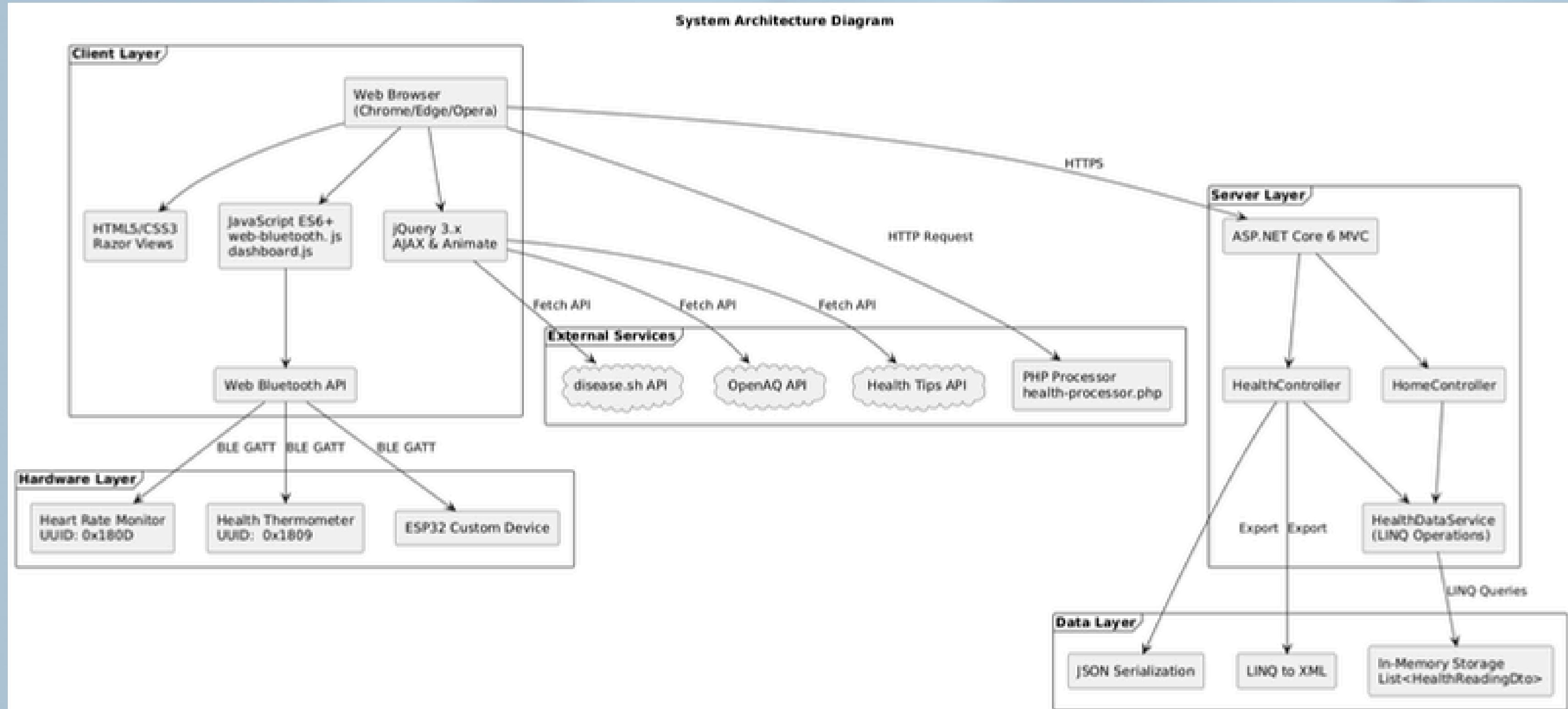
```
<div class="row mb-4">
  <div class="col-12 col-md-3">
    <div class="card shadow stat-
card">      <!-- Content -->
    </div>
  </div>
</div>
```

## Custom Mobile Adjustments (CSS)

```
@media (max-width: 768px) {
  .stat-card .display-3 {
    font-size: 2.5rem; /* Smaller font for mobile */
  }

  .btn-group .btn {
    display: block; /* Stack buttons vertically
    width: 100%; margin-bottom: 10px;
  }
}
```

# How Does The System Work?



# Main Dashboard View

## Medical/Fitness Dashboard

### Web Bluetooth Devices

Connect Heart Rate Monitor Connect Thermometer Connect ESP32 Device Disconnect All

No devices connected. Click a button above to connect via Web Bluetooth.

Heart Rate

Temperature

ESP32 Sensor

Statistics (LINQ Aggregations)

Total Readings: 0

Connected Devices: 0

Last Reading: --

Device Types: 0

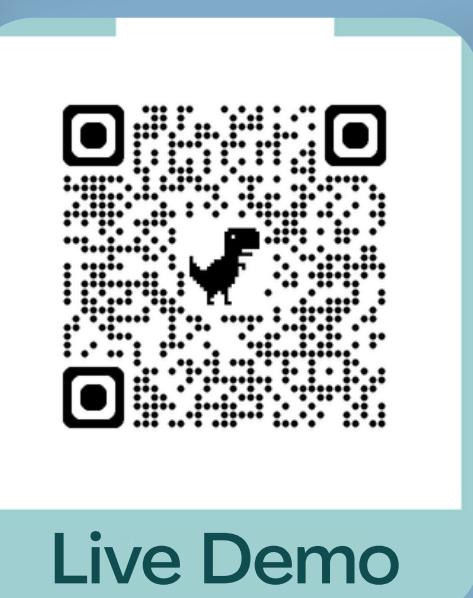
### Recent Readings (Last 20)

| Timestamp | Device Type | Device ID | Value | Unit | Notes |
|-----------|-------------|-----------|-------|------|-------|
|-----------|-------------|-----------|-------|------|-------|

### Export Data

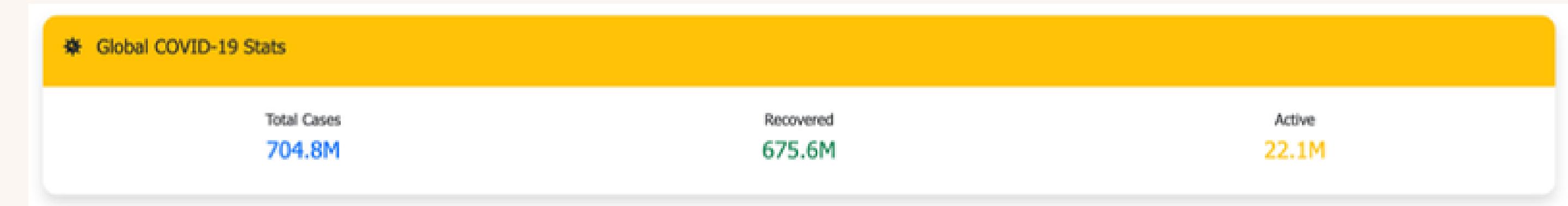
Export as JSON Export as XML View Analytics Clear All Data

© 2026 - Medical/Fitness Health Dashboard - Web Programming Final Project





Live Demo



# Analytics Model & Covid Data Fetch View

## Health Data Management

Total Records

**30**

Devices

**1**

Device Types

**1**

Last Update

22:32:36

# Future Roadmap

## Persistence

- Transition from in-memory storage to a persistent database (SQL Server) for long-term tracking.

## Mobile Apps

- Develop companion native apps using .NET MAUI for broader hardware support.

## Analytics

- Integrate Machine Learning for anomaly detection and health trend prediction.

# Thank you very much!

