**Struktur Direktori**

Pastikan struktur direktori proyek Anda seperti berikut:

bash

Salin kode

/project

├── /src

│ ├── /controllers

│ │ └── queueController.js

│ ├── /models

│ │ └── queue.js

│ ├── /routes

│ │ ├── adminRoutes.js

│ │ └── queueRoutes.js

│ ├── /view

│ │ ├── patient.html

│ │ └── poli.html

│ ├── server.js

│ └── .env

└── package.json

**Langkah-langkah**

**1. Instalasi Dependensi**

Pastikan Anda telah menginstal express, mongoose, body-parser, socket.io, dan dotenv dengan menggunakan perintah:

bash

Salin kode

npm install express mongoose body-parser socket.io dotenv

**2. File .env**

Pastikan file .env Anda mengandung konfigurasi berikut:

plaintext

Salin kode

# Konfigurasi MongoDB

MONGODB\_URI=mongodb://localhost:27017/poli-antrian

# Port server

PORT=3000

# Kunci rahasia untuk pengaturan lainnya

SECRET\_KEY=secretpass123

# Kunci API untuk layanan eksternal (jika diperlukan)

API\_KEY=apikey123

**3. File server.js**

Ini adalah kode lengkap untuk server.js. Kode ini akan menyediakan server Express untuk menangani permintaan, menghubungkan ke MongoDB, dan mengaktifkan Socket.io untuk komunikasi real-time.

javascript

Salin kode

const express = require('express');

const mongoose = require('mongoose');

const bodyParser = require('body-parser');

const socketio = require('socket.io');

const http = require('http');

require('dotenv').config();

const adminRoutes = require('./routes/adminRoutes');

const queueRoutes = require('./routes/queueRoutes');

const app = express();

const server = http.createServer(app);

const io = socketio(server);

// Middleware

app.use(bodyParser.json());

app.use(bodyParser.urlencoded({ extended: true }));

// Serve static files from 'src/view' directory

app.use(express.static('src/view'));

// MongoDB connection

mongoose.connect(process.env.MONGODB\_URI, {

useNewUrlParser: true,

useUnifiedTopology: true

})

.then(() => {

console.log('Connected to MongoDB');

})

.catch(err => {

console.error('MongoDB connection error:', err.message);

process.exit(1); // Exit the process on connection error

});

// Socket.io connection

app.set('io', io);

// Routes

app.use('/api/admin', adminRoutes);

app.use('/api/queue', queueRoutes);

// Server start

const PORT = process.env.PORT || 3000;

server.listen(PORT, () => {

console.log(`Server running on port ${PORT}`);

});

**4. Akses File HTML**

Pastikan untuk mengakses file HTML menggunakan URL yang sesuai:

* http://localhost:3000/patient.html
* http://localhost:3000/poli.html

**Penjelasan Tambahan**

* **Middleware express.static**: Baris app.use(express.static('src/view')); bertujuan untuk menyediakan akses ke file statis (termasuk file HTML) yang berada di dalam direktori src/view. Ini memungkinkan Anda mengakses patient.html dan poli.html dengan menyertakan nama file tersebut dalam URL tanpa menyertakan path lengkap dari direktori root.
* **MongoDB Connection**: Kode mongoose.connect(process.env.MONGODB\_URI, { useNewUrlParser: true, useUnifiedTopology: true }) bertanggung jawab untuk menghubungkan aplikasi Anda ke server MongoDB yang sudah berjalan. Pastikan MongoDB berjalan di localhost pada port 27017 sesuai dengan nilai yang Anda set dalam file .env.
* **Server Start**: Server Express dijalankan pada port yang diatur dalam variabel lingkungan PORT. Jika variabel PORT tidak tersedia dalam file .env, maka server akan berjalan di port default 3000.

Dengan langkah-langkah di atas, Anda seharusnya dapat mengakses patient.html dan poli.html dengan benar melalui server Express yang sudah dikonfigurasi dengan baik. Pastikan Anda telah menjalankan MongoDB dan server Express dengan benar sesuai dengan instruksi yang telah dijelaskan sebelumnya.

3.5

lebih lengkap lagi termasuk html

ChatGPT

Baik, saya akan menunjukkan bagaimana struktur lengkap dari proyek Anda, termasuk file HTML (patient.html dan poli.html), serta bagaimana mereka berinteraksi dengan server Node.js dan MongoDB.

**Struktur Direktori**

Pastikan struktur direktori proyek Anda seperti berikut:

bash

Salin kode

/project

├── /src

│ ├── /controllers

│ │ └── queueController.js

│ ├── /models

│ │ └── queue.js

│ ├── /routes

│ │ ├── adminRoutes.js

│ │ └── queueRoutes.js

│ ├── /view

│ │ ├── patient.html

│ │ └── poli.html

│ ├── server.js

│ └── .env

└── package.json

**File server.js**

Ini adalah file utama server Node.js yang menggunakan Express dan Socket.io untuk menangani permintaan, serta mongoose untuk terhubung ke MongoDB.

javascript

Salin kode

// src/server.js

const express = require('express');

const mongoose = require('mongoose');

const bodyParser = require('body-parser');

const socketio = require('socket.io');

const http = require('http');

require('dotenv').config();

const adminRoutes = require('./routes/adminRoutes');

const queueRoutes = require('./routes/queueRoutes');

const app = express();

const server = http.createServer(app);

const io = socketio(server);

// Middleware

app.use(bodyParser.json());

app.use(bodyParser.urlencoded({ extended: true }));

// Serve static files from 'src/view' directory

app.use(express.static('src/view'));

// MongoDB connection

mongoose.connect(process.env.MONGODB\_URI, {

useNewUrlParser: true,

useUnifiedTopology: true

})

.then(() => {

console.log('Connected to MongoDB');

})

.catch(err => {

console.error('MongoDB connection error:', err.message);

process.exit(1); // Exit the process on connection error

});

// Socket.io connection

app.set('io', io);

// Routes

app.use('/api/admin', adminRoutes);

app.use('/api/queue', queueRoutes);

// Server start

const PORT = process.env.PORT || 3000;

server.listen(PORT, () => {

console.log(`Server running on port ${PORT}`);

});

**File patient.html**

Berikut adalah contoh sederhana dari file patient.html yang menampilkan nomor antrian dari poli tertentu:

html

Salin kode

<!-- src/view/patient.html -->

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Patient Page</title>

<link rel="stylesheet" href="/styles.css">

<script src="/socket.io/socket.io.js"></script>

</head>

<body>

<h1>Patient Page</h1>

<div>

<h2 id="poliName">Poli Gigi</h2>

<h3>Current Queue Number: <span id="currentNumber">0</span></h3>

</div>

<script>

const socket = io();

const poliName = 'Poli Gigi'; // Ganti dengan nama poli yang sesuai

async function fetchQueue() {

try {

const response = await fetch(`/api/queue/${poliName}`);

if (!response.ok) {

throw new Error('Failed to fetch queue');

}

const queue = await response.json();

document.getElementById('currentNumber').innerText = queue.currentNumber;

} catch (error) {

console.error('Error fetching queue:', error.message);

}

}

socket.on('queueUpdated', () => {

fetchQueue();

});

fetchQueue();

</script>

</body>

</html>

**File poli.html**

Berikut adalah contoh sederhana dari file poli.html yang menampilkan nomor antrian dari poli tertentu dan mengizinkan penanganan operasi berikutnya dan sebelumnya:

html

Salin kode

<!-- src/view/poli.html -->

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Poli Page</title>

<link rel="stylesheet" href="/styles.css">

<script src="/socket.io/socket.io.js"></script>

</head>

<body>

<h1>Poli Page</h1>

<div>

<h2 id="poliName">Poli Gigi</h2>

<h3>Current Queue Number: <span id="currentNumber">0</span></h3>

<button onclick="previous()">Previous</button>

<button onclick="next()">Next</button>

</div>

<script>

const socket = io();

const poliName = 'Poli Gigi'; // Ganti dengan nama poli yang sesuai

async function fetchQueue() {

try {

const response = await fetch(`/api/queue/${poliName}`);

if (!response.ok) {

throw new Error('Failed to fetch queue');

}

const queue = await response.json();

document.getElementById('currentNumber').innerText = queue.currentNumber;

} catch (error) {

console.error('Error fetching queue:', error.message);

}

}

async function next() {

try {

const response = await fetch('/api/queue/next', {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify({ poliName })

});

if (!response.ok) {

throw new Error('Failed to update queue');

}

socket.emit('updateQueue');

fetchQueue();

} catch (error) {

console.error('Error updating queue:', error.message);

}

}

async function previous() {

try {

const response = await fetch('/api/queue/previous', {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify({ poliName })

});

if (!response.ok) {

throw new Error('Failed to update queue');

}

socket.emit('updateQueue');

fetchQueue();

} catch (error) {

console.error('Error updating queue:', error.message);

}

}

socket.on('queueUpdated', () => {

fetchQueue();

});

fetchQueue();

</script>

</body>

</html>

**Catatan Tambahan**

* Pastikan Anda telah mengatur CSS atau styling lainnya di file styles.css di direktori src/view sesuai kebutuhan.
* Pastikan bahwa server MongoDB telah berjalan di localhost pada port 27017 sebelum Anda menjalankan server Node.js.
* Gunakan npm run dev untuk menjalankan server Node.js dengan menggunakan Nodemon, yang akan secara otomatis memulai ulang server setiap kali terjadi perubahan dalam file.

Dengan langkah-langkah ini, Anda seharusnya dapat menjalankan server Node.js dengan Express, mengakses halaman patient.html dan poli.html dari browser, dan berinteraksi dengan data antrian yang dikelola melalui MongoDB dengan dukungan Socket.io untuk pembaruan real-time. Pastikan untuk memeriksa konsol browser untuk melihat jika ada kesalahan dalam permintaan HTTP atau koneksi Socket.io.