Laporan Praktikum

Mata Kuliah Pemrograman Web & Pemrograman Berorientasi Objek

Dosen Pengampu:

Willdan Aprizal Arifin, S.Pd., M.Kom.

Disusun Oleh :

Silvia Isti Lestary (2311883)

Novia Ramadhani (2305968)

PROGRAM STUDI SISTEM INFORMASI KELAUTAN

UNIVERSITAS PENDIDIKAN INDONESIA

2024

Link GitHub

https://github.com/silviaalestary/miniproject_1

PENDAHULUAN

Node.js adalah platform runtime berbasis JavaScript yang dirancang untuk menjalankan kode di sisi server. Dengan kemampuan menangani permintaan secara asinkron dan mendukung skalabilitas, Node.js sangat cocok untuk aplikasi yang memerlukan pemrosesan data real-time, seperti aplikasi pesan instan atau permainan daring. Framework Express.js, yang dibangun di atas Node.js, menawarkan alat sederhana namun kuat untuk pengembangan aplikasi web dan API, termasuk fitur routing HTTP, middleware, dan kemampuan untuk membangun aplikasi dengan cepat.

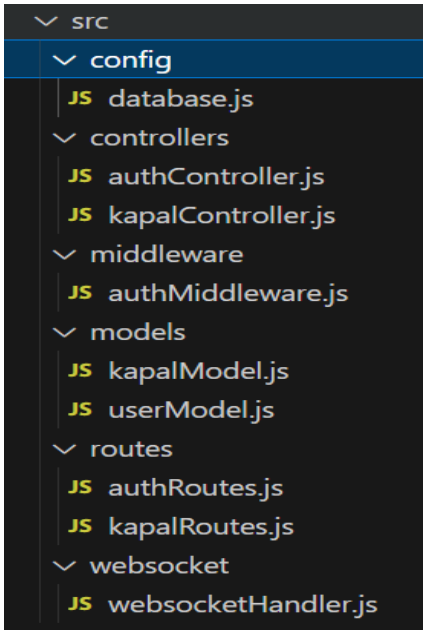
Untuk penyimpanan data, MySQL adalah sistem manajemen basis data relasional yang memungkinkan pengelolaan data dalam tabel terstruktur menggunakan bahasa SQL. MySQL dikenal dengan keamanan yang baik dan dukungannya terhadap transaksi, menjadikannya pilihan utama untuk aplikasi web yang membutuhkan pengelolaan data kompleks.

Dalam era teknologi yang semakin berkembang, kebutuhan akan komunikasi data secara real-time antara klien dan server menjadi semakin penting. Salah satu teknologi yang memungkinkan komunikasi dua arah yang efisien dan cepat adalah WebSocket. WebSocket adalah protokol komunikasi berbasis TCP yang memungkinkan koneksi dua arah (full-duplex) antara klien dan server melalui satu koneksi yang terbuka secara terus-menerus. Berbeda dengan protokol HTTP tradisional, di mana komunikasi bersifat satu arah (klien mengirim permintaan dan server merespons), WebSocket memungkinkan kedua belah pihak untuk mengirim data kapan saja selama koneksi aktif. Pertama kali distandardisasi oleh IETF (Internet Engineering Task Force) dalam RFC 6455 dan diimplementasikan dalam browser modern serta banyak platform server-side seperti Node.js.

1. CREATE DATABASE
   Kami beri nama “mini_project”
2. CREATE TABLE
   CREATE TABLE kapal (
   id_kapal INT PRIMARY KEY AUTO_INCREMENT,
   nama_kapal VARCHAR(100) NOT NULL,
   jenis_kapal VARCHAR(50) NOT NULL,
   kapasitas_muatan INT NOT NULL,
   waktu_terdaftar TIMESTAMP DEFAULT CURRENT_TIMESTAMP
   );

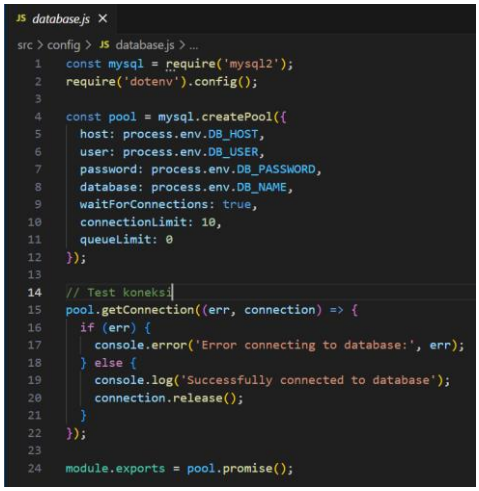
   CREATE TABLE users (
   id_user INT PRIMARY KEY AUTO_INCREMENT,
   username VARCHAR(50) UNIQUE NOT NULL,
   password VARCHAR(100) NOT NULL,
   role ENUM('admin', 'user') NOT NULL
   );

SOURCE CODE



- **CONFIG**
  Folder ini digunakan untuk menyimpan konfigurasi aplikasi

- **Database.js**
  Berisi konfigurasi koneksi ke database, termasuk parameter seperti host, username, password, dan nama database.



```
database.js
src > config > database.js > ...
1   const mysql = require('mysql2');
2   require('dotenv').config();
3
4   const pool = mysql.createPool({
5     host: process.env.DB_HOST,
6     user: process.env.DB_USER,
7     password: process.env.DB_PASSWORD,
8     database: process.env.DB_NAME,
9     waitForConnections: true,
10    connectionLimit: 10,
11    queueLimit: 0
12  });
13
14  // Test koneksi
15  pool.getConnection((err, connection) => {
16    if (err) {
17      console.error('Error connecting to database:', err);
18    } else {
19      console.log('Successfully connected to database');
20      connection.release();
21    }
22  });
23
24  module.exports = pool.promise();
```

- **CONTROLLERS**
  Folder ini berisi file yang menangani logika aplikasi dan interaksi antara model dan view.

- **authContollers.js**
  Mengelola logika otentikasi seperti login, register, dan logout.

```
const UserModel = require('../models/userModel');
const jwt = require('jsonwebtoken');
const bcrypt = require('bcrypt');

class AuthController {
  static async register(req, res) {
    try {
      const { username, password, role } = req.body;

      // Validasi input
      if (!username || !password || !role) {
        return res.status(400).json({ message: 'Semua field harus diisi' });
      }

      // Cek username yang sudah ada
      const existingUser = await UserModel.findByUsername(username);
      if (existingUser) {
        return res.status(400).json({ message: 'Username sudah digunakan' });
      }

      await UserModel.register(username, password, role);
      res.status(201).json({ message: 'Registrasi berhasil' });
    } catch (error) {
      res.status(500).json({ message: error.message });
    }
  }

  static async login(req, res) {
    try {
```

```
      const { username, password } = req.body;

      const user = await UserModel.findByUsername(username);
      if (!user) {
        return res.status(401).json({ message: 'Username atau password salah' });
      }

      const isValidPassword = await bcrypt.compare(password, user.password);
      if (!isValidPassword) {
        return res.status(401).json({ message: 'Username atau password salah' });
      }

      const token = jwt.sign(
        { id: user.id_user, username: user.username, role: user.role },
        process.env.JWT_SECRET,
        { expiresIn: '24h' }
      );

      res.json({ token });
    } catch (error) {
      res.status(500).json({ message: error.message });
    }
  }
}

module.exports = AuthController;
```

- **kapalController.js**
  Mengelola logika terkait data kapal, seperti menambahkan kapal, mengedit, menghapus, dan menampilkan data kapal.

```
const KapalModel = require('../models/kapalModel');
const WebSocket = require('ws');
const jwt = require('jsonwebtoken');

class KapalController {
  // Fungsi helper untuk verifikasi token
  static verifyUserToken(req) {
    const token = req.headers['authorization']?.split(' ')[1];
    if (!token) {
      throw new Error('Token diperlukan');
    }
    return jwt.verify(token, process.env.JWT_SECRET);
  }

  static async getAll(req, res) {
    try {
      // Verifikasi token
```

```
      await KapalController.verifyUserToken(req);

      const kapal = await KapalModel.getAll();
      res.json(kapal);
    } catch (error) {
      if (error.message === 'Token diperlukan' || error.name === 
'JsonWebTokenError') {
        return res.status(401).json({ message: 'Token tidak valid atau 
diperlukan' });
      }
      res.status(500).json({ message: error.message });
    }
  }

  static async getById(req, res) {
    try {
      await KapalController.verifyUserToken(req);

      const kapal = await KapalModel.getById(req.params.id);
      if (!kapal) {
        return res.status(404).json({ message: 'Kapal tidak ditemukan' });
      }
      res.json(kapal);
    } catch (error) {
      if (error.message === 'Token diperlukan' || error.name === 
'JsonWebTokenError') {
        return res.status(401).json({ message: 'Token tidak valid atau 
diperlukan' });
      }
      res.status(500).json({ message: error.message });
    }
  }

  static async create(req, res) {
    try {
      const decoded = await KapalController.verifyUserToken(req);

      // Cek role admin
      if (decoded.role !== 'admin') {
        return res.status(403).json({ message: 'Hanya admin yang dapat membuat 
data' });
      }

      const { nama_kapal, jenis_kapal, kapasitas_muatan } = req.body;

      if (!nama_kapal || !jenis_kapal || !kapasitas_muatan) {
        return res.status(400).json({ message: 'Semua field harus diisi' });
      }
```

```
      if (kapasitas_muatan <= 0) {
        return res.status(400).json({ message: 'Kapasitas muatan harus
positif' });
      }

      const result = await KapalModel.create(req.body);

      // Notifikasi WebSocket
      global.wss.clients.forEach(client => {
        if (client.readyState === WebSocket.OPEN) {
          client.send(JSON.stringify({
            event: 'data_changed',
            message: 'Data kapal baru ditambahkan',
            data: { id_kapal: result.insertId, ...req.body }
          }));
        }
      });

      res.status(201).json({ message: 'Kapal berhasil ditambahkan' });
    } catch (error) {
      if (error.message === 'Token diperlukan' || error.name ===
'JsonWebTokenError') {
        return res.status(401).json({ message: 'Token tidak valid atau
diperlukan' });
      }
      res.status(500).json({ message: error.message });
    }
  }

  static async update(req, res) {
    try {
      const decoded = await KapalController.verifyUserToken(req);

      // Cek role admin
      if (decoded.role !== 'admin') {
        return res.status(403).json({ message: 'Hanya admin yang dapat
mengubah data' });
      }

      const kapal = await KapalModel.getById(req.params.id);
      if (!kapal) {
        return res.status(404).json({ message: 'Kapal tidak ditemukan' });
      }

      await KapalModel.update(req.params.id, req.body);

      // Notifikasi WebSocket
```

```
      global.wss.clients.forEach(client => {
        if (client.readyState === WebSocket.OPEN) {
          client.send(JSON.stringify({
            event: 'data_changed',
            message: 'Data kapal diperbarui',
            data: { id_kapal: req.params.id, ...req.body }
          }));
        }
      });

      res.json({ message: 'Kapal berhasil diperbarui' });
    } catch (error) {
      if (error.message === 'Token diperlukan' || error.name === 'JsonWebTokenError') {
        return res.status(401).json({ message: 'Token tidak valid atau diperlukan' });
      }
      res.status(500).json({ message: error.message });
    }
  }

  static async delete(req, res) {
    try {
      const decoded = await KapalController.verifyUserToken(req);

      // Cek role admin
      if (decoded.role !== 'admin') {
        return res.status(403).json({ message: 'Hanya admin yang dapat menghapus data' });
      }

      const kapal = await KapalModel.getById(req.params.id);
      if (!kapal) {
        return res.status(404).json({ message: 'Kapal tidak ditemukan' });
      }

      await KapalModel.delete(req.params.id);

      // Notifikasi WebSocket
      global.wss.clients.forEach(client => {
        if (client.readyState === WebSocket.OPEN) {
          client.send(JSON.stringify({
            event: 'data_changed',
            message: 'Data kapal dihapus',
            data: { id_kapal: req.params.id }
          }));
        }
      });
```

```
      res.json({ message: 'Kapal berhasil dihapus' });
    } catch (error) {
      if (error.message === 'Token diperlukan' || error.name === 
'JsonWebTokenError') {
        return res.status(401).json({ message: 'Token tidak valid atau 
diperlukan' });
      }
      res.status(500).json({ message: error.message });
    }
  }
}

module.exports = KapalController;
```

➢ **MIDDLEWARE**
Folder ini berisi middleware yang berfungsi sebagai perantara dalam proses permintaan dan respons.

- **authMiddleware.js**
  Middleware untuk memverifikasi otentikasi pengguna, misalnya memeriksa token JWT atau hak akses sebelum mengakses rute tertentu.

```
const jwt = require('jsonwebtoken');

const verifyToken = (req, res, next) => {
  const token = req.headers['authorization']?.split(' ')[1];

  if (!token) {
    return res.status(403).json({ message: 'Token diperlukan untuk 
autentikasi' });
  }

  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    req.user = decoded;
    next();
  } catch (err) {
    return res.status(401).json({ message: 'Token tidak valid' });
  }
};

const isAdmin = (req, res, next) => {
  if (req.user.role !== 'admin') {
    return res.status(403).json({ message: 'Akses ditolak. Hanya admin yang 
diizinkan.' });
  }
  next();
};
```

```
module.exports = { verifyToken, isAdmin };
```

➢ **MODELS**
Folder ini berisi representasi tabel database dalam bentuk model.

- **kapalModels.js**
  Mendefinisikan struktur tabel kapal dan fungsi untuk mengakses atau memanipulasi data kapal di database.

```
const db = require('../config/database');

class KapalModel {
  static async getAll() {
    const [rows] = await db.query('SELECT * FROM kapal');
    return rows;
  }

  static async getById(id) {
    const [rows] = await db.execute(
      'SELECT * FROM kapal WHERE id_kapal = ?',
      [id]
    );
    return rows[0];
  }

  static async create(data) {
    const { nama_kapal, jenis_kapal, kapasitas_muatan } = data;
    const [result] = await db.execute(
      'INSERT INTO kapal (nama_kapal, jenis_kapal, kapasitas_muatan) VALUES
(?, ?, ?)',
      [nama_kapal, jenis_kapal, kapasitas_muatan]
    );
    return result;
  }

  static async update(id, data) {
    const { nama_kapal, jenis_kapal, kapasitas_muatan } = data;
    const [result] = await db.execute(
      'UPDATE kapal SET nama_kapal = ?, jenis_kapal = ?, kapasitas_muatan = ?
WHERE id_kapal = ?',
      [nama_kapal, jenis_kapal, kapasitas_muatan, id]
    );
    return result;
  }

  static async delete(id) {
    const [result] = await db.execute(
      'DELETE FROM kapal WHERE id_kapal = ?',
      [id]
```

```
    );
    return result;
  }
}

module.exports = KapalModel;
```

- **userModels.js**
  Mendefinisikan struktur tabel users dan fungsi terkait otentikasi dan pengelolaan pengguna.

```
const db = require('../config/database');
const bcrypt = require('bcrypt');

class UserModel {
  static async register(username, password, role) {
    const hashedPassword = await bcrypt.hash(password, 10);
    const [result] = await db.execute(
      'INSERT INTO users (username, password, role) VALUES (?, ?, ?)',
      [username, hashedPassword, role]
    );
    return result;
  }

  static async findByUsername(username) {
    const [rows] = await db.execute(
      'SELECT * FROM users WHERE username = ?',
      [username]
    );
    return rows[0];
  }
}

module.exports = UserModel;
```

  - **ROUTES**
    Folder ini menangani definisi rute API untuk menangani permintaan HTTP.

- **authRoutes.js**
  Mendefinisikan rute untuk proses otentikasi seperti login, logout, dan registrasi.

```
const express = require('express');
const router = express.Router();
const AuthController = require('../controllers/authController');
const { verifyToken, isAdmin } = require('../middleware/authMiddleware');

router.post('/register', AuthController.register);
router.post('/login', AuthController.login);

module.exports = router;
```

- **kapalRoutes.js**

Mendefinisikan rute untuk mengelola data kapal seperti menambah, menghapus, atau memperbarui data kapal.

```
const express = require('express');
const router = express.Router();
const KapalController = require('../controllers/kapalController');

router.get('/', KapalController.getAll);
router.get('/:id', KapalController.getById);
router.post('/', KapalController.create);
router.put('/:id', KapalController.update);
router.delete('/:id', KapalController.delete);

module.exports = router;
```

- **WEBSOCKET**
  Folder ini menangani komunikasi real-time menggunakan WebSocket.

- **websocketHandler.js**
  Mengelola koneksi WebSocket, menangani event seperti koneksi, pesan masuk, dan pemutusan koneksi.

```
module.exports = (wss) => {
  global.wss = wss;

  wss.on('connection', (ws) => {
    console.log('Client terhubung ke WebSocket');

    // Kirim pesan selamat datang
    ws.send(JSON.stringify({
      event: 'welcome',
      message: 'Selamat datang di Kapal WebSocket Server',
      data: {
        time: new Date().toISOString(),
        clientCount: wss.clients.size
      }
    }));

    ws.on('message', (message) => {
      try {
        const parsedMessage = JSON.parse(message);
        console.log('Pesan diterima:', parsedMessage);
      } catch (error) {
        console.log('Pesan diterima (raw):', message);
      }
    });

    ws.on('close', () => {
      console.log('Client terputus dari WebSocket');
    });
```

```
    ws.on('error', (error) => {
      console.error('WebSocket error:', error);
    });
  });

  // Fungsi untuk broadcast ke semua client
  wss.broadcast = (data) => {
    wss.clients.forEach(client => {
      if (client.readyState === WebSocket.OPEN) {
        client.send(JSON.stringify(data));
      }
    });
  };
};
```

KESIMPULAN

Penggunaan WebSocket, CRUD, dan API dalam pengelolaan data kapal menawarkan solusi yang efisien dan terintegrasi untuk mengelola informasi secara real-time dan terstruktur. Dengan WebSocket, sistem dapat memberikan notifikasi langsung kepada pengguna ketika ada perubahan data kapal, seperti penambahan, pembaruan, atau penghapusan data, tanpa perlu melakukan permintaan berkala ke server. Sementara itu, operasi CRUD (Create, Read, Update, Delete) memungkinkan pengelolaan data kapal secara sistematis, mulai dari pendaftaran kapal baru, melihat daftar kapal, memperbarui informasi kapal, hingga menghapus data yang sudah tidak diperlukan. Semua operasi ini diakses melalui API (Application Programming Interface), yang bertindak sebagai jembatan komunikasi antara klien dan server, memastikan data dapat diakses dan dimanipulasi dengan aman serta konsisten. Integrasi ketiga teknologi ini menciptakan sistem yang responsif, skalabel, dan mudah dikelola, sehingga sangat cocok digunakan dalam aplikasi pemantauan dan pengelolaan data kapal yang memerlukan pembaruan data secara cepat dan akurat.