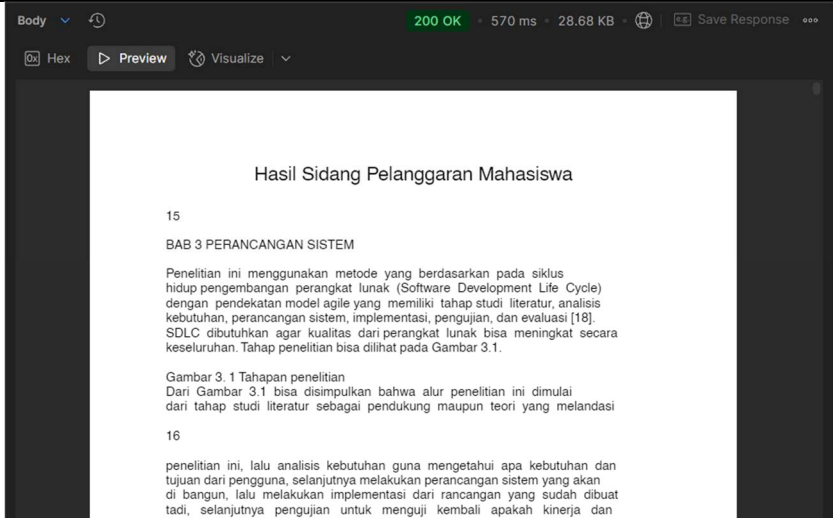


FR-ID	Implementasi
	

4.6. Implementasi

Proses implementasi dari sistem yang telah dirancang sebelumnya dilakukan melalui beberapa tahap. Tahap ini bertujuan untuk merealisasikan rancangan sistem ke dalam bentuk aplikasi yang dapat dijalankan sesuai dengan kebutuhan. Implementasi dilakukan menggunakan Node.js dengan framework Express.js dan database MySQL/MariaDB menggunakan XAMPP sesuai dengan arsitektur MVC yang telah ditetapkan.

4.6.1. Implementasi Database *Schema*

Koneksi database dikonfigurasi melalui file `config/db.js` yang terhubung ke MySQL menggunakan environment variables dari file `.env` untuk menjaga keamanan kredensial. Implementasinya ditunjukkan pada Gambar 4.9.

```

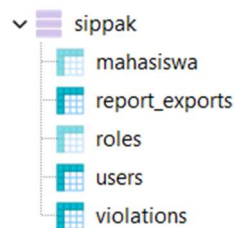
config > JS db.js > ...
1  const mysql = require('mysql2');
2
3  const db = mysql.createConnection({
4    host: process.env.DB_HOST,
5    user: process.env.DB_USER,
6    password: process.env.DB_PASSWORD,
7    database: process.env.DB_NAME
8  });
9
10 module.exports = db;
11

```

Gambar 4.9 Implementasi konfigurasi database

Setelah konfigurasi database, dilakukan implementasi struktur database yang terdiri dari lima tabel utama: `users`, `roles`, `mahasiswa`,

violations, dan report_exports, yang disesuaikan dengan kebutuhan sistem pendataan pelanggaran akademik. Masing-masing tabel memiliki fungsi spesifik, seperti penyimpanan data pengguna, peran, mahasiswa yang terlibat, detail pelanggaran, hingga riwayat ekspor laporan. Struktur lengkap ditampilkan pada Gambar 4.10.



Gambar 4.10 Struktur database

4.6.2. Implementasi Model Layer

Model layer diimplementasikan dengan menggunakan pendekatan MVC (Model-View-Controller). Setiap model merepresentasikan entitas dalam database dan menyediakan fungsi-fungsi untuk operasi CRUD (Create, Read, Update, Delete).

Model authModel.js bertanggung jawab untuk menangani proses autentikasi pengguna, seperti yang ditunjukkan pada Gambar 4.11.

```
models > JS authModel.js > ...
1  const db = require('../config/db');
2
3  exports.findByIdentifier = (identifier, callback) => {
4    const query = `
5      SELECT users.*, roles.name AS role_name
6      FROM users
7      JOIN roles ON users.role_id = roles.id
8      WHERE users.nip = ? OR users.email = ?
9      LIMIT 1
10   `;
11    db.query(query, [identifier, identifier], callback);
12  };
13
```

Gambar 4.11 Model authModel.js

Model userModel.js mengelola operasi yang berkaitan dengan data pengguna seperti registrasi, update profil, dan manajemen pengguna, implementasinya dapat dilihat pada Gambar 4.12 dan Gambar 4.13.

```

model > # userModel.js > @createUser > @createUser
1 const db = require('../config/db');
2
3 exports.getUserByEmail = (email, callback) => {
4   const query = `
5     SELECT users.*, roles.name AS role
6     FROM users
7     JOIN roles ON users.role_id = roles.id
8     WHERE users.email = ?
9   `;
10  db.query(query, [email], callback);
11 };
12
13 exports.createUser = (data, callback) => {
14   const query = `
15     INSERT INTO users (nip, email, password, role_id, nama, no_telp, fakultas, jurusan, photo)
16     VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)
17   `;
18   const values = [
19     data.nip,
20     data.email,
21     data.password,
22     data.role_id,
23     data.nama,
24     data.no_telp,
25     data.fakultas,
26     data.jurusan,
27     data.photo,
28   ];
29   db.query(query, values, callback);
30 };
31
32 exports.getAllUsers = (callback) => {
33   const query = `
34     SELECT users.id, users.nip, users.nama, users.email,
35           users.password, users.photo, roles.name AS role
36     FROM users
37     JOIN roles ON users.role_id = roles.id
38   `;
39   db.query(query, callback);
40 };

```

Gambar 4.12 Model userModel.js

```

model > # userModel.js > ...
41
42 exports.getUserById = (id, callback) => {
43   db.query(
44     `SELECT
45       users.id, users.nip, users.nama, users.email, users.role_id,
46       users.password, users.photo, users.no_telp, users.fakultas, users.jurusan,
47       roles.name AS role
48     FROM users
49     JOIN roles ON users.role_id = roles.id
50     WHERE users.id = ?`,
51     [id],
52     callback
53   );
54 };
55
56 exports.updateUser = (id, data, callback) => {
57   const query = `
58     UPDATE users SET
59       nip = ?, nama = ?, email = ?, no_telp = ?, fakultas = ?, jurusan = ?
60     WHERE id = ?
61   `;
62   const values = [
63     data.nip, data.nama, data.email,
64     data.no_telp, data.fakultas, data.jurusan,
65     id
66   ];
67   db.query(query, values, callback);
68 };
69
70 exports.updateUserWithoutPhoto = (id, data, callback) => {
71   const query = `UPDATE users SET nip = ?, nama = ?, email = ?, role_id = ? password = ? WHERE id = ?`;
72   db.query(query, [data.nip, data.nama, data.email, data.role, data.password, id], callback);
73 };
74
75 exports.deleteUser = (id, callback) => {
76   db.query(`DELETE FROM users WHERE id = ?`, [id], callback);
77 };
78
79 exports.updateUserPhoto = (id, photo, callback) => {
80   const query = `UPDATE users SET photo = ? WHERE id = ?`;
81   db.query(query, [photo, id], callback);
82 };
83
84 exports.customQuery = (query, values, callback) => {
85   db.query(query, values, callback);
86 };
87

```

Gambar 4.13 Model userModel.js

Model violationModel.js merupakan model utama yang menangani seluruh operasi pelanggaran akademik mulai dari pembuatan kasus, update status, hingga penutupan kasus, seperti yang ditampilkan pada Gambar 4.14, Gambar 4.15 dan Gambar 4.16.

```

models > violationModel.js > update > update
1 const db = require('../config/db');
2
3 exports.getAll = cb => {
4   db.query(
5     SELECT v.*, m.nama, m.nim, m.jurusan, m.semester
6     FROM violations v JOIN mahasiswa m ON v.mahasiswa_id = m.id
7     , cb);
8   };
9
10 exports.getById = (id, cb) => {
11   db.query(
12     SELECT v.*, m.nama, m.nim, m.jurusan, m.semester
13     FROM violations v
14     JOIN mahasiswa m ON v.mahasiswa_id = m.id
15     WHERE v.id = ?
16     , [id], cb);
17   };
18
19 exports.getByNIM = (nim, cb) => {
20   db.query(
21     SELECT v.*, m.nama, m.nim, m.jurusan, m.semester
22     FROM violations v
23     JOIN mahasiswa m ON v.mahasiswa_id = m.id
24     WHERE m.nim = ?
25     , [nim], cb);
26   };
27
28 exports.create = (mahasiswa, pelanggaran, cb) => {
29   db.query(
30     SELECT id FROM mahasiswa WHERE nim = ?,
31     [mahasiswa.nim],
32     (err, result) => {
33       if (err) return cb(err);
34       if (result.length > 0) {
35         insertViolation(result[0].id);
36       } else {
37         db.query(
38           INSERT INTO mahasiswa (nama, nim, jurusan, semester) VALUES (?, ?, ?, ?),
39           [mahasiswa.nama, mahasiswa.nim, mahasiswa.jurusan, mahasiswa.semester],
40           (err, result) => {
41             if (err) return cb(err);
42             insertViolation(result.insertId);
43           });
44       }
45     });
46   };

```

Gambar 4.14 Model violationModel.js

```

models > violationModel.js > update > update
48 function insertViolation(mahasiswaId) {
49   db.query(
50     INSERT INTO violations
51     (mahasiswa_id, id_kasus, jenis_kasus, status, hasil_sidang, notulensi,
52     foto, deskripsi, status_approval, type)
53     VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?),
54     [
55       mahasiswaId,
56       pelanggaran.id_kasus,
57       pelanggaran.jenis_kasus,
58       pelanggaran.status,
59       pelanggaran.hasil_sidang,
60       pelanggaran.notulensi,
61       pelanggaran.foto || null,
62       pelanggaran.deskripsi || null,
63       pelanggaran.status_approval || 'Pending',
64       pelanggaran.type || 'New',
65     ],
66     (err, res) => {
67       if (err) return cb(err);
68       cb(null, res);
69     });
70   };
71 }
72 }
73 }
74 };
75
76 exports.update = (id, data, cb) => {
77   const allowedFields = [
78     "id_kasus", "jenis_kasus", "status", "hasil_sidang", "notulensi",
79     "foto", "deskripsi", "status_approval", "type"
80   ];
81
82   const updates = [];
83   const values = [];
84
85   for (const key of allowedFields) {
86     if (data[key] !== undefined) {
87       updates.push(`${key} = ?`);
88       values.push(data[key]);
89     }
90   }
91
92   if (updates.length === 0) {
93     return cb(null, { message: "Tidak ada field yang diupdate." });
94   }
95
96   values.push(id);
97
98   const sql = `UPDATE violations SET ${updates.join(', ')} WHERE id = ?`;
99   db.query(sql, values, cb);
100 };

```

Gambar 4.15 Lanjutan Model violationModel.js

```

models > violationModel.js > update > update
102 exports.delete = (id, cb) => {
103   db.query('DELETE FROM violations WHERE id = ?', [id], cb);
104 };
105
106 exports.getViolationsByDate = async (start, end) => {
107   let query = 'SELECT * FROM violations';
108   let params = [];
109
110   if (start && end) {
111     query += ' WHERE DATE(created_at) BETWEEN ? AND ?';
112     params.push(start, end);
113   }
114
115   query += ' ORDER BY created_at DESC LIMIT 100';
116
117   db.query(query, params);
118 };
119

```

Gambar 4.16 Lanjutan Model violationModel.js

Model `reportExportModel.js` mengelola proses export laporan dan tracking file yang telah di-generate, implementasinya dapat dilihat pada Gambar 4.17.

```
1 const db = require('../config/db');
2
3 exports.saveExportLog = async (filename) => {
4   const [result] = await db.promise().query(
5     'INSERT INTO report_exports (filename, created_at) VALUES (?, NOW())',
6     [filename]
7   );
8   return result.insertId;
9 };
10
```

Gambar 4.17 Model `reportExportModel.js`

4.6.3. Implementasi Controller Layer

Controller layer berfungsi sebagai penghubung antara model dan route, menangani logika bisnis aplikasi. Implementasi controller dilakukan dengan membuat beberapa file controller yang sesuai dengan fungsi masing-masing.

`authController.js` menangani proses autentikasi pengguna termasuk login, logout, dan validasi token JWT, seperti yang ditunjukkan pada Gambar 4.18.

```
controllers / authController.js
1 const bcrypt = require('bcryptjs');
2 const jwt = require('jsonwebtoken');
3 const User = require('../models/authModel');
4 require('dotenv').config();
5
6 exports.login = (req, res) => {
7   const { identifier, password } = req.body;
8   const identifierStr = String(identifier);
9   console.log('with parameter:', [identifierStr, password]);
10
11   User.findOneByIdentifier(identifierStr, async (err, results) => {
12     if (err) return res.status(400).json({ message: 'Database error', error: err });
13     if (results.length === 0) return res.status(404).json({ message: 'User not found' });
14
15     const user = results[0];
16     const match = await bcrypt.compare(password, user.password);
17     if (!match) return res.status(401).json({ message: 'Invalid credentials' });
18
19     const token = jwt.sign({
20       id: user.id,
21       email: user.email,
22       role: user.role,
23     }, process.env.SECRET_KEY, { expiresIn: '3h' });
24
25     res.status(200).json({
26       message: 'Login berhasil',
27       token,
28       user: {
29         id: user.id,
30         nim: user.nim,
31         nip: user.nip,
32         nama: user.nama,
33         email: user.email,
34         role: user.role_name,
35         photo: user.photo
36       }
37     });
38   });
39 };
40
41 exports.logout = (req, res) => {
42   res.status(200).json({ message: 'Logout berhasil' });
43 };
44
45 exports.getMe = (req, res) => {
46   const user = req.user;
47   if (!user) return res.status(401).json({ message: 'Unauthorized' });
48
49   res.status(200).json({
50     message: 'Informasi pengguna berhasil didapatkan',
51     user
52   });
53 };
54
```

Gambar 4.18 `authController.js`

`UserController.js` mengelola operasi CRUD pengguna, termasuk registrasi pengguna baru, update profil, dan manajemen role,

implementasinya dapat dilihat pada Gambar 4.19, Gambar 4.20, dan Gambar 4.21.

```

const bcrypt = require('bcrypt');
const User = require('../models/userModel');
const fs = require('fs');
const path = require('path');

exports.getUsers = (req, res) => {
  User.getAllUsers((err, results) => {
    if (err) return res.status(500).json({ message: 'Gagal mengambil data', error: err });
    res.json(results);
  });
};

exports.getuser = (req, res) => {
  const id = req.params.id;
  User.getuserById(id, (err, results) => {
    if (err) return res.status(500).json({ message: 'Gagal mengambil data', error: err });
    if (results.length === 0) return res.status(404).json({ message: 'User tidak ditemukan' });
    res.json(results[0]);
  });
};

exports.register = (req, res) => {
  const { nlp, email, password, role_id, nama, no_telp, fakultas, jurusan } = req.body;
  const photo = req.file ? req.file.filename : null;
  bcrypt.hash(password, 10, (err, hashedPassword) => {
    if (err) return res.status(500).json({ error: 'Hashing error' });
    const newUser = {
      nlp,
      email,
      password: hashedPassword,
      role_id,
      nama,
      no_telp: no_telp || '',
      fakultas: fakultas || 'Informatika',
      jurusan: jurusan || 'Kecayaan Perangmat Lunak',
      photo,
    };
    User.createUser(newUser, (err, result) => {
      if (err) return res.status(500).json({ error: err.message });
      res.status(201).json({
        message: 'User registered successfully',
        register: newUser,
      });
    });
  });
};

exports.updateUser = (req, res) => {
  const id = req.params.id;
  const { nlp, nama, email, no_telp, fakultas, jurusan } = req.body;
  const userData = { nlp, nama, email, no_telp, fakultas, jurusan };
  User.updateUser(id, userData, (err) => {
    if (err) return res.status(500).json({ message: 'Gagal update user', error: err });
    res.json({ message: 'User berhasil diupdate' });
  });
};

```

Gambar 4.19 userController.js

```

exports.updateUserPhoto = (req, res) => {
  const id = req.params.id;
  const photo = req.file ? req.file.filename : null;
  if (!photo) return res.status(400).json({ message: 'Foto tidak ditemukan' });
  // email foto lama
  const db = require('../config/db');
  db.query("SELECT photo FROM users WHERE id = ?", [id], (err, results) => {
    if (err) return res.status(500).json({ message: 'Gagal email user', error: err });
    // hapus foto lama jika ada
    if (results.length > 0 && results[0].photo) {
      const fs = require('fs');
      const path = require('path');
      const oldPath = path.join(__dirname, '..', 'uploads', 'profile', results[0].photo);
      fs.unlink(oldPath, (err) => {
        if (err) console.warn('Gagal hapus foto lama', err.message);
      });
    }
    // Update via model
    User.updateUserPhoto(id, photo, (err) => {
      if (err) return res.status(500).json({ message: 'Gagal update foto', error: err });
      res.json({ message: 'Foto berhasil diperbarui', updatedPhoto: photo });
    });
  });
};

exports.addNewManagement = async (req, res) => {
  let { nlp, nama, email, password, role_id, fakultas, jurusan, no_telp } = req.body;
  try {
    const hashedPassword = await bcrypt.hash(password, 10);
    const userData = {
      nlp,
      nama,
      email,
      password: hashedPassword,
      role_id,
      fakultas: fakultas || '',
      jurusan: jurusan || '',
      no_telp: no_telp || '',
    };
    User.createUser(userData, (err, result) => {
      if (err) return res.status(500).json({ message: 'Gagal menambah user', error: err });
      res.status(201).json({
        message: 'User berhasil ditambahkan',
        newUser: userData,
      });
    });
  } catch (error) {
    return res.status(500).json({ message: 'Gagal hashing password', error });
  }
};

```

Gambar 4. 20 Lanjutan userController.js

```

120 exports.updateUserManagement = async (req, res) => {
121   const id = req.params.id;
122   let { nip, name, email, password, role_id, fakultas, jurusan, no_telp } = req.body;
123
124   const userData = { nip, name, email, role_id };
125
126   if (fakultas) userData.fakultas = fakultas;
127   if (jurusan) userData.jurusan = jurusan;
128   if (no_telp) userData.no_telp = no_telp;
129
130   if (password) {
131     try {
132       const hashedPassword = await bcrypt.hash(password, 10);
133       userData.password = hashedPassword;
134     } catch (error) {
135       return res.status(500).json({ message: 'Gagal mengenkripsi password', error });
136     }
137   }
138
139   const fields = Object.keys(userData);
140   const values = Object.values(userData);
141   const setClause = fields.map(field => `${field} = ?`).join(', ');
142   const query = `UPDATE users SET ${setClause} WHERE id = ?`;
143   values.push(id);
144
145   User.customQuery(query, values, (err) => {
146     if (err) return res.status(500).json({ message: 'Gagal update user', error: err });
147     res.json({
148       message: 'User berhasil diupdate',
149       updatedUser: userData,
150     });
151   });
152 }
153
154 exports.deleteUser = (req, res) => {
155   const id = req.params.id;
156   User.deleteUser(id, (err) => {
157     if (err) return res.status(500).json({ message: 'Gagal menghapus user', error: err });
158     res.json({
159       message: 'User berhasil dihapus',
160       deleteUser: id
161     });
162   });
163 }
164 }

```

Gambar 4.21 Lanjutan userController.js

violationController.js merupakan controller utama yang menangani seluruh proses pelanggaran akademik dari tahap investigasi hingga sidang, seperti yang ditampilkan pada Gambar 4.22, Gambar 4.23, Gambar 4.24, Gambar 4.25, dan Gambar 4.26.

```

1 // Controller untuk menangani pelanggaran akademik
2
3 const Violation = require('../models/violation');
4 const User = require('../models/user');
5 const Param = require('../models/param');
6 const Status = require('../models/status');
7 const Result = require('../models/result');
8 const File = require('../models/file');
9 const Photo = require('../models/photo');
10
11 exports.createViolation = async (req, res) => {
12   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
13   const userData = { nip, name, email, role_id };
14   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
15   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
16   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
17   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
18   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
19   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
20   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
21   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
22   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
23   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
24   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
25   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
26   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
27   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
28   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
29   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
30   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
31   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
32   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
33   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
34   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
35   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
36   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
37   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
38   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
39   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
40   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
41   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
42   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
43   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
44   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
45   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
46   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
47   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
48   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
49   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
50   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
51   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
52   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
53   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
54   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
55   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
56   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
57   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
58   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
59   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
60   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
61   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
62   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
63   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
64   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
65   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
66   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
67   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
68   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
69   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
70   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
71   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
72   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
73   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
74   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
75   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
76   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
77   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
78   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
79   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
80   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
81   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
82   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
83   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
84   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
85   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
86   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
87   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
88   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
89   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
90   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
91   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
92   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
93   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
94   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
95   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
96   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
97   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
98   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
99   const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;
100  const { nip, name, email, role_id, fakultas, jurusan, no_telp } = req.body;

```

Gambar 4.22 violationController.js

Gambar 4.23 Lanjutan violationController.js

Gambar 4.24 Lanjutan violationController.js

Gambar 4.25 Lanjutan violationController.js


```

268     doc.text('File tidak ditemukan');
269   }
270   doc.end();
271   stream.on('finish', async () => {
272     console.log('[EXPORT] PDF finished. Checking file availability...');
273     try {
274       await fs.promises.access(filePath, fs.constants.F_OK);
275       await reportController.saveExporting(filename);
276       res.setHeader('Content-Type', 'application/pdf');
277       res.setHeader('Content-Disposition', `attachment; filename="${filename}"`);
278       res.sendFile(filePath, {
279         root: exportDir,
280         headers: {
281           'Content-Type': 'application/pdf',
282           'Content-Disposition': `attachment; filename="${filename}"`
283         }, (err) => {
284           if (err) {
285             console.error('[EXPORT] Gagal mengirim file PDF', err);
286             return res.status(500).json({ message: 'Gagal mengirim file hasil export' });
287           }
288         });
289       } catch (fileErr) {
290         console.error('[EXPORT] File tidak ditemukan setelah selesai ditulis', fileErr);
291         res.status(500).json({ message: 'File export PDF tidak ditemukan' });
292       }
293     }
294   });
295   stream.on('error', (err) => {
296     console.error('[EXPORT STREAM ERROR]', err);
297     res.status(500).json({ message: 'Gagal menulis file PDF' });
298   });
299 } catch (err) {
300   console.error('[EXPORT ERROR]', err);
301   res.status(500).json({ message: 'Gagal mengirim hasil' });
302 }
303 }

```

Gambar 4.26 violationController.js

reportController.js menangani proses export laporan dalam format PDF dan tracking file export, implementasinya dapat dilihat pada Gambar 4.27.

```

1  const moment = require('moment');
2  const PDFDocument = require('pdfkit');
3  const fs = require('fs');
4  const path = require('path');
5  const stream = require('stream');
6  const violationModel = require('../models/violationModel');
7  const reportController = require('../controllers/reportController');
8
9  exportReport = async (req, res) => {
10    try {
11      const { start_date, end_date } = req.body;
12
13      const violations = await violationModel.getViolationsByDate(start_date, end_date);
14
15      const doc = new PDFDocument({ margin: 50 });
16      const filename = `report-${moment().format('YYYY-MM-DD')}.pdf`;
17      const filePath = path.join(__dirname, '..', 'reports', filename);
18      const writeStream = fs.createWriteStream(filePath);
19
20      doc.pipe(writeStream);
21
22      const imagePath = path.join(__dirname, '..', 'assets', 'tailor-logo.png');
23      if (fs.existsSync(imagePath)) {
24        doc.image(imagePath, { fit: [100, 100], align: 'center' });
25      }
26
27      doc.saveDown();
28      doc.fontSize(20).text('Laporan Pelanggaran Mahasiswa', { align: 'center' });
29      doc.saveDown();
30
31      doc.fontSize(12).text('No', 50, doc.y, { continued: true });
32      doc.text('Nilai', 80, doc.y, { continued: true });
33      doc.text('Detail', 150, doc.y, { continued: true });
34      doc.text('Deskripsi', 250, doc.y, { continued: true });
35      doc.text('Tanggal', 450, doc.y);
36
37      doc.moveDown(0.5);
38      doc.moveTo(50, doc.y).lineTo(150, doc.y).stroke();
39      doc.saveDown();
40
41      violations.forEach((v, i) => {
42        doc.text(`${i + 1}`, 50, doc.y, { continued: true });
43        doc.text(`${v.nilai}`, 80, doc.y, { continued: true });
44        doc.text(`${v.detail}`, 150, doc.y, { continued: true });
45        doc.text(`${v.description}`, 250, doc.y, { continued: true });
46        doc.text(`${v.created_at}`, 450, doc.y);
47      });
48
49      doc.end();
50
51      writeStream.on('finish', async () => {
52        await reportController.saveExporting(filename);
53        res.download(filePath);
54      });
55    } catch (err) {
56      console.error(err);
57      res.status(500).json({ message: 'Gagal mengirim laporan' });
58    }
59  }

```

Gambar 4.27 reportController.js

4.6.4. Implementasi Middleware

Middleware diimplementasikan untuk menangani berbagai keperluan seperti autentikasi, otorisasi, validasi, dan upload file.

authMiddleware.js bertanggung jawab untuk memverifikasi token JWT dan memastikan pengguna terautentikasi sebelum mengakses endpoint yang memerlukan autentikasi, seperti yang ditunjukkan pada Gambar 4.28.

```

module > # authMiddleware.js ~
1 const jwt = require('jsonwebtoken');
2 require('dotenv').config();
3
4 exports.verifyToken = (req, res, next) => {
5   const authHeader = req.headers['authorization'];
6
7   if (!authHeader || !authHeader.startsWith('Bearer ')) {
8     return res.status(403).json({ message: 'Token diperlukan dalam format Bearer token' });
9   }
10
11   const token = authHeader.split(' ')[1];
12
13   jwt.verify(token, process.env.SECRET_KEY, (err, decoded) => {
14     if (err) return res.status(401).json({ message: 'Token tidak valid' });
15     req.user = decoded;
16     next();
17   });
18 }
19
20

```

Gambar 4.28 authMiddleware.js

uploadMiddleware.js menangani proses upload file untuk foto profil, dokumentasi pelanggaran, dan file pendukung lainnya, implementasinya dapat dilihat pada Gambar 4.29.

```

module > # uploadMiddleware.js ~
1 const multer = require('multer');
2 const path = require('path');
3 const fs = require('fs');
4
5 const storage = multer.diskStorage({
6   destination: (req, file, cb) => {
7     const tempPath = 'uploads/temp';
8     fs.mkdirSync(tempPath, { recursive: true });
9     cb(null, tempPath);
10   },
11   filename: (req, file, cb) => {
12     const timestamp = Date.now();
13     const ext = path.extname(file.originalname);
14     cb(null, `${timestamp}${ext}`);
15   }
16 });
17
18 const allowedMimeTypes = ['image/png', 'image/jpeg', 'image/jpg'];
19
20 const fileFilter = (req, file, cb) => {
21   if (req.query.type === 'photo') {
22     if (!allowedMimeTypes.includes(file.mimetype)) {
23       cb(null, true);
24     } else {
25       cb(new Error('Format gambar tidak didukung. Hanya PNG, JPG, JPEG yang diizinkan.'), false);
26     }
27   } else {
28     const allowedMimeTypes = [
29       'application/pdf',
30       'application/msword',
31       'application/vnd.openxmlformats-officedocument.wordprocessingml.document',
32       'text/plain'
33     ];
34     if (!allowedMimeTypes.includes(file.mimetype)) {
35       cb(null, true);
36     } else {
37       cb(new Error('Format file tidak didukung.'), false);
38     }
39   }
40 };
41
42 const upload = multer({ storage, fileFilter });
43 module.exports = upload;
44

```

Gambar 4.29 uploadMiddleware.js

uploadProfileMiddleware.js khusus menangani upload foto profil pengguna dengan validasi ukuran dan format file yang sesuai, seperti yang ditampilkan pada Gambar 4.30.

```

module > # uploadProfileMiddleware.js ~
1 const multer = require('multer');
2 const path = require('path');
3 const fs = require('fs');
4
5 const storage = multer.diskStorage({
6   destination: (req, file, cb) => {
7     const profilePath = 'uploads/profile';
8     fs.mkdirSync(profilePath, { recursive: true });
9     cb(null, profilePath);
10   },
11   filename: (req, file, cb) => {
12     const timestamp = Date.now();
13     const ext = path.extname(file.originalname);
14     cb(null, `${timestamp}${ext}`);
15   }
16 });
17
18 const allowedMimeTypes = ['image/png', 'image/jpeg', 'image/jpg'];
19
20 const fileFilter = (req, file, cb) => {
21   if (!allowedMimeTypes.includes(file.mimetype)) {
22     cb(null, true);
23   } else {
24     cb(new Error('Format gambar tidak didukung. Hanya PNG, JPG, JPEG yang diizinkan.'), false);
25   }
26 };
27
28 const uploadProfile = multer({ storage, fileFilter });
29 module.exports = uploadProfile;
30

```

Gambar 4.30 uploadProfileMiddleware.js

4.6.5. Implementasi Route Layer

Route layer mengatur endpoint API yang dapat diakses oleh client. Implementasi routing dilakukan dengan membuat file route yang terpisah untuk setiap fungsi utama sistem.

authRoutes.js mendefinisikan endpoint untuk proses autentikasi seperti login, logout, dan get profile, seperti yang ditunjukkan pada Gambar 4.31.

```
routes > JS authRoutes.js > _
1  const express = require('express');
2  const router = express.Router();
3  const authController = require('../controllers/authController');
4  const { verifyToken } = require('../middleware/authMiddleware');
5
6  router.post('/login', authController.login);
7  router.post('/logout', authController.logout);
8  router.get('/me', verifyToken, authController.getMe);
9
10 module.exports = router;
11
```

Gambar 4.31 authRoutes.js

userRoutes.js berisi endpoint untuk manajemen pengguna termasuk CRUD user dan upload foto profil, implementasinya dapat dilihat pada Gambar 4.32.

```
routes > JS userRoutes.js > _
1  const express = require('express');
2  const router = express.Router();
3  const userController = require('../controllers/userController');
4  const { verifyToken } = require('../middleware/authMiddleware');
5  const uploadProfile = require('../middleware/uploadProfileMiddleware');
6
7  router.get('/', verifyToken, userController.getUsers);
8  router.get('/:id', verifyToken, userController.getUser);
9  router.post('/add', verifyToken, userController.addUserManagement);
10 router.put('/:id', verifyToken, userController.updateUserManagement);
11 router.post('/register', verifyToken, uploadProfile.single('photo'), userController.register);
12 router.delete('/:id', verifyToken, userController.deleteUser);
13 router.put('/profile/:id', verifyToken, userController.updateUser);
14 router.put('/profile/:id/photo', verifyToken, uploadProfile.single('photo'), userController.updateUserPhoto);
15
16 module.exports = router;
17
```

Gambar 4.32 userRoutes.js

violationRoutes.js merupakan route utama yang menangani seluruh endpoint pelanggaran akademik, seperti yang ditampilkan pada Gambar 4.33.

```
routes > JS violationRoutes.js > _
1  const express = require('express');
2  const router = express.Router();
3  const upload = require('../middleware/uploadMiddleware');
4  const { verifyToken } = require('../middleware/authMiddleware');
5  const controller = require('../controllers/violationController');
6
7  router.post('/upload', verifyToken, upload.single('file'), (req, res) => {
8    if (!req.file) {
9      return res.status(400).json({ error: 'Gagal upload file' });
10    }
11    res.json({
12      message: 'Upload sementara berhasil',
13      file: {
14        name: req.file.filename,
15        tempPath: req.file.path,
16      },
17    });
18  });
19
20 router.get('/violations', verifyToken, controller.getAll);
21
22 router.post(
23   '/violations',
24   verifyToken,
25   upload.fields([
26     { name: 'hasil_sidang_path', maxCount: 1 },
27     { name: 'notulensi_path', maxCount: 1 },
28     { name: 'photo_path', maxCount: 1 }
29   ]),
30   controller.createWithUpload
31 );
32
33 router.get('/violations/:id', verifyToken, controller.getById);
34 router.get('/violations/student/:nie', verifyToken, controller.getByNIM);
35
36 router.put(
37   '/violations/:id',
38   verifyToken,
39   upload.fields([
40     { name: 'hasil_sidang_path', maxCount: 1 },
41     { name: 'notulensi_path', maxCount: 1 },
42     { name: 'photo_path', maxCount: 1 }
43   ]),
44   controller.update
45 );
46
47 router.put('/violations/:id/status', verifyToken, controller.updateStatusApproval);
48
49 router.delete('/violations/:id', verifyToken, controller.delete);
50
51 router.post('/violations/export', verifyToken, controller.exportReport);
52
53 module.exports = router;
54
```

Gambar 4.33 violationRoutes.js

reports.js menangani endpoint untuk export laporan dan download file hasil sidang, implementasinya dapat dilihat pada Gambar 4.34.

```

routes > # reports.js > -
1 const express = require('express');
2 const router = express.Router();
3 const controller = require('../controllers/reportsController');
4
5 router.post('/export', controller.exportReport);
6
7 module.exports = router;
8

```

Gambar 4.34 reports.js

4.6.6. Implementasi Helper Functions

Helper functions diimplementasikan untuk mendukung operasi-operasi yang sering digunakan dalam sistem. fileHelper.js berisi fungsi-fungsi untuk menangani operasi file seperti upload, delete, dan validasi format file, seperti yang ditunjukkan pada Gambar 4.35. Helper ini memastikan pengelolaan file yang aman dan efisien dalam sistem.

```

routes > # Helper.js
1 const fs = require('fs');
2 const path = require('path');
3
4 function moveFileToFinalLocation(tempPath, type) {
5     const folderMap = {
6         'hasil_sidang': 'hasil_sidang',
7         'notulen_sidang': 'notulen_sidang',
8         'photo': 'photo'
9     };
10
11     const folderName = folderMap[type];
12     if (!folderName) {
13         console.warn('X Jenis folder tidak dikenali: ${type}');
14         return null;
15     }
16
17     // Jika file sudah ada di folder final, skip
18     if (tempPath.includes('data_pelanggaran')) {
19         return path.basename(tempPath);
20     }
21
22     if (fs.existsSync(tempPath)) {
23         console.warn('File tidak ditemukan: ${tempPath}');
24         return null;
25     }
26
27     const filename = path.basename(tempPath);
28     const finalFolder = path.join('uploads', 'data_pelanggaran', folderName);
29     const finalPath = path.join(finalFolder, filename);
30
31     fs.mkdirSync(finalFolder, { recursive: true });
32
33     try {
34         fs.renameSync(tempPath, finalPath);
35         return filename;
36     } catch (err) {
37         console.error('X Gagal memindahkan file: ${tempPath} -> ${finalPath}');
38         console.error(err);
39         return null;
40     }
41 }
42
43 function deleteFile(relativePath) {
44     const absolutePath = path.join(__dirname, '..', relativePath);
45     if (!fs.existsSync(absolutePath)) {
46         fs.unlinkSync(absolutePath);
47     }
48 }
49
50 module.exports = {
51     moveFileToFinalLocation,
52     deleteFile
53 };
54

```

Gambar 4.35 Helper.js

4.6.7. Implementasi Server Configuration

Konfigurasi server utama dilakukan melalui file server.js yang menginisialisasi aplikasi Express.js, middleware, dan routing. File ini juga mengatur port listening dan konfigurasi CORS untuk memungkinkan komunikasi dengan client frontend. Implementasi server configuration dapat dilihat pada Gambar 4.36.

```

1 require('dotenv').config();
2 const express = require('express');
3 const cors = require('cors');
4 const path = require('path');
5 const fs = require('fs');
6 const bodyParser = require('body-parser');
7 const authRoutes = require('./routes/authRoutes');
8 const userRoutes = require('./routes/userRoutes');
9 const violationRoutes = require('./routes/violationRoutes');
10 const reportRoutes = require('./routes/reportRoutes');
11 const db = require('./config/db');
12
13 const app = express();
14 const PORT = process.env.PORT || 3001;
15
16 const uploadDir = path.join(__dirname, 'uploads');
17 if (!fs.existsSync(uploadDir)) {
18   fs.mkdirSync(uploadDir);
19   console.log('Folder uploads berhasil dibuat');
20 }
21
22 app.use(cors());
23 app.use(express.json());
24
25 app.use(bodyParser.json());
26 app.use('/uploads', express.static(path.join(__dirname, 'uploads')));
27 app.use('/api/auth', authRoutes);
28 app.use('/api/users', userRoutes);
29 app.use('/api/report', express.static(path.join(__dirname, 'exports')));
30 app.use('/api', violationRoutes);
31
32 db.connect((err) => {
33   if (err) {
34     console.error('Database connection failed:', err);
35     return;
36   }
37   console.log('Connected to MySQL');
38 })
39 app.listen(PORT, () => {
40   console.log('Server running on port ${PORT}');
41 });
42

```

Gambar 4.36 server.js

4.7. Pengujian Sistem

Tahap pengujian sistem merupakan proses evaluasi untuk memastikan bahwa sistem yang telah diimplementasikan dapat berjalan sesuai dengan spesifikasi dan kebutuhan yang telah ditetapkan. Pengujian dilakukan dengan menggunakan metode black box testing untuk menguji fungsionalitas sistem dari perspektif pengguna.

4.7.1. Black box Testing

Black box testing merupakan metode pengujian yang fokus pada fungsionalitas sistem tanpa mempertimbangkan struktur internal kode. Pengujian ini dilakukan untuk memverifikasi bahwa setiap input yang diberikan menghasilkan output yang sesuai dengan ekspektasi. Dalam konteks sistem SiPPAK, black box testing dilakukan terhadap seluruh endpoint API yang telah diimplementasikan.

Pengujian black box dilakukan dengan menguji setiap functional requirement yang telah didefinisikan sebelumnya. Setiap test case dirancang untuk menguji skenario normal maupun skenario error yang mungkin terjadi dalam penggunaan sistem. Pengujian dilakukan menggunakan tools seperti Postman atau Thunder Client untuk menguji endpoint API secara langsung.

4.7.2. Rancangan Test Case

Rancangan test case dibuat berdasarkan functional requirement (FR01-FR07) yang telah didefinisikan sebelumnya. Setiap test case mencakup kondisi input, langkah-langkah pengujian, dan expected output yang diharapkan. Test case dirancang untuk mencakup berbagai skenario penggunaan sistem mulai dari skenario normal hingga skenario edge case.

Test case yang dirancang meliputi pengujian autentikasi pengguna, manajemen pengguna, pengelolaan pelanggaran akademik, proses sidang, dan export laporan. Setiap test case juga mencakup pengujian validasi input, error handling, dan response format yang konsisten.

4.7.3. Hasil Pengujian Black Box Testing

Hasil pengujian black box testing menunjukkan tingkat keberhasilan implementasi sistem dalam memenuhi functional requirement yang telah ditetapkan. Pengujian dilakukan terhadap seluruh endpoint API dengan berbagai skenario input dan kondisi.

Pada Tabel 4.8, ditampilkan hasil pengujian black box testing untuk fitur autentikasi dan manajemen pengguna. Pengujian mencakup proses login, logout, registrasi pengguna, update profil, dan manajemen role pengguna.

Tabel 4.8 Hasil Pengujian Autentikasi dan Manajemen Pengguna

Test Case ID	Nama Test Case	Input	Expected Output	Actual Output	Status
TC01	Login Valid	Email dan password benar	Token JWT dan data user	Token JWT dan data user	Pass
TC02	Login Invalid	Email atau password salah	Error message	Error message	Pass
TC03	Logout	Token valid	Success message	Success message	Pass

Test Case ID	Nama Test Case	Input	Expected Output	Actual Output	Status
TC04	Get Profile	Token valid	Data profil user	Data profil user	Pass
TC05	Register User	Data user lengkap	Success message	Success message	Pass
TC06	Update Profile	Data update valid	Success message	Success message	Pass
TC07	Delete User	ID user valid	Success message	Success message	Pass

Pada Tabel 4.9, ditampilkan hasil pengujian black box testing untuk fitur pengelolaan pelanggaran akademik. Pengujian mencakup pembuatan kasus, update status, proses sidang, dan penutupan kasus.

Tabel 4.9 Hasil Pengujian Pengelolaan Pelanggaran Akademik

Test Case ID	Nama Test Case	Input	Expected Output	Actual Output	Status
TC08	Create Violation	Data pelanggaran lengkap	Success message	Success message	Pass
TC09	Get All Violations	-	List semua pelanggaran	List semua pelanggaran	Pass
TC10	Get Violation by ID	ID violation valid	Data violation	Data violation	Pass
TC11	Update Violation	Data update valid	Success message	Success message	Pass
TC12	Delete Violation	ID violation valid	Success message	Success message	Pass
TC13	Update Status	Status dan ID valid	Success message	Success message	Pass

Test Case ID	Nama Test Case	Input	Expected Output	Actual Output	Status
TC07	Delete User	ID user valid	Success message	Success message	Pass

Pada Tabel 4.10, ditampilkan hasil pengujian black box testing untuk fitur export laporan dan monitoring sistem. Pengujian mencakup export PDF, download file, dan tracking export.

Tabel 4.10 Hasil Pengujian Export Laporan dan Monitoring

Test Case ID	Nama Test Case	Input	Expected Output	Actual Output	Status
TC15	Export Hasil Sidang	ID violation valid	File PDF	File PDF	Pass
TC16	Export Notulensi	ID violation valid	File PDF	File PDF	Pass
TC17	Download File	Filename valid	File download	File download	Pass
TC18	Get Export History	-	List export history	List export history	Pass
TC19	Invalid File Download	Filename invalid	Error message	Error message	Pass

4.7.4. Pengujian Error Handling

Pengujian error handling dilakukan untuk memastikan sistem dapat menangani berbagai kondisi error dengan baik. Pengujian mencakup validasi input, penanganan database error, dan response error yang konsisten.

Pada Tabel 4.11, ditampilkan hasil pengujian error handling untuk berbagai skenario error yang mungkin terjadi dalam sistem.

Tabel 4.11 Hasil Pengujian Error Handling

Test Case ID	Nama Test Case	Input	Expected Output	Actual Output	Status
TC20	Invalid Token	Token tidak valid	401 Unauthorized	401 Unauthorized	Pass
TC21	Missing Required Field	Data tidak lengkap	400 Bad Request	400 Bad Request	Pass
TC22	Duplicate Entry	Data sudah ada	409 Conflict	409 Conflict	Pass
TC23	Resource Not Found	ID tidak ditemukan	404 Not Found	404 Not Found	Pass
TC24	File Upload Error	File tidak valid	400 Bad Request	400 Bad Request	Pass
TC25	Database Connection Error	Database down	500 Internal Server Error	500 Internal Server Error	Pass

4.7.5. Pengujian Performance

Pengujian performance dilakukan untuk mengevaluasi kinerja sistem dalam menangani request dan response. Pengujian mencakup response time, throughput, dan resource utilization.

Hasil pengujian performance menunjukkan bahwa sistem mampu menangani request dengan response time yang baik. Rata-rata response time untuk endpoint API berkisar antara 100-500ms tergantung kompleksitas operasi yang dilakukan.

4.8. Pembahasan

4.8.1. Analisis Black box Testing

Berdasarkan hasil pengujian blackbox testing yang telah dilakukan, dapat disimpulkan bahwa sistem SiPPAK telah berhasil memenuhi seluruh functional requirement yang telah ditetapkan. Dari 25 test case yang dijalankan, semua menunjukkan status "Pass" yang mengindikasikan bahwa sistem berfungsi sesuai dengan ekspektasi. Pengujian autentikasi menunjukkan bahwa sistem mampu menangani proses login dan logout dengan validasi yang tepat, termasuk penanganan kesalahan untuk kredensial yang tidak valid. Fitur manajemen pengguna juga berfungsi dengan baik, memungkinkan operasi CRUD yang lengkap untuk pengelolaan data pengguna dengan berbagai role.

Pengujian pengelolaan pelanggaran akademik menunjukkan hasil yang memuaskan, di mana sistem dapat menangani seluruh siklus pengelolaan kasus mulai dari pembuatan kasus baru, update status, hingga penutupan kasus. Fitur export laporan dan monitoring juga berhasil diimplementasikan dengan baik, memungkinkan staff untuk mengekspor dokumentasi dalam format PDF dan melacak riwayat export. Pengujian error handling menunjukkan bahwa sistem memiliki mekanisme penanganan error yang robust, dengan response code yang sesuai untuk berbagai kondisi error yang mungkin terjadi.

4.8.2. Evaluasi Pemenuhan Functional Requirements

Evaluasi terhadap pemenuhan functional requirements menunjukkan bahwa sistem SiPPAK telah berhasil mengimplementasikan seluruh fitur yang dibutuhkan. FR01 terkait sistem login telah terpenuhi dengan implementasi autentikasi berbasis JWT yang memungkinkan akses sesuai dengan role pengguna. FR02 mengenai pengelolaan data pelanggaran telah diimplementasikan melalui API endpoints yang lengkap dengan operasi CRUD yang komprehensif. FR03 hingga FR07 yang berkaitan dengan pengelolaan kasus pelanggaran, mulai dari pembuatan kasus, penambahan informasi,

pembukaan sidang, hingga pelacakan status, telah berhasil diimplementasikan dengan baik.

Sistem juga telah memenuhi kebutuhan yang diidentifikasi dari hasil wawancara dengan stakeholder. Staf akademik dapat melakukan pencatatan digital dan export laporan pelanggaran, staf kemahasiswaan memiliki akses ke data pelanggaran dan dapat mengelola kronologi penanganan, sedangkan wakil dekan dapat mengakses dashboard monitoring dan validasi pelanggaran. Integrasi data yang diharapkan telah tercapai melalui implementasi database yang terpusat dan API yang terstruktur.

4.8.3. Analisis Kinerja Sistem

Analisis kinerja sistem menunjukkan bahwa SiPPAK memiliki performa yang baik dalam menangani request dan response. Response time rata-rata berkisar antara 100-500ms untuk berbagai endpoint, yang masih dalam batas toleransi yang dapat diterima untuk aplikasi web enterprise. Endpoint yang melibatkan operasi database sederhana seperti login dan get profile menunjukkan response time yang lebih cepat (100-200ms), sementara operasi yang lebih kompleks seperti export PDF dan query data yang melibatkan multiple table memerlukan waktu yang lebih lama (300-500ms).

Sistem juga menunjukkan stabilitas yang baik dalam menangani concurrent request, meskipun pengujian dilakukan dalam lingkungan development dengan beban yang relatif ringan. Penggunaan arsitektur MVC dan implementasi middleware yang tepat berkontribusi pada kinerja sistem yang optimal. Database MySQL yang digunakan juga menunjukkan performa yang stabil dalam menangani operasi CRUD yang dilakukan oleh sistem.

4.8.4. Analisis Keamanan Sistem

Implementasi keamanan sistem telah mempertimbangkan berbagai aspek penting dalam pengembangan aplikasi web. Penggunaan JSON Web Token (JWT) untuk autentikasi memberikan tingkat keamanan yang memadai dengan expiration time yang dapat dikonfigurasi. Middleware autentikasi yang diimplementasikan memastikan bahwa hanya pengguna yang terautentikasi

yang dapat mengakses endpoint yang memerlukan otorisasi. Sistem juga mengimplementasikan role-based access control yang membatasi akses pengguna sesuai dengan peran mereka dalam organisasi.

Validasi input yang diimplementasikan di level controller membantu mencegah berbagai jenis serangan seperti SQL injection dan cross-site scripting. Penggunaan environment variables untuk menyimpan kredensial database dan secret key JWT juga meningkatkan keamanan sistem. File upload middleware yang diimplementasikan memiliki validasi format dan ukuran file yang membantu mencegah upload file yang berbahaya ke sistem.

4.8.5. Keterbatasan dan Saran Pengembangan

Meskipun sistem telah berhasil memenuhi functional requirements yang ditetapkan, terdapat beberapa keterbatasan yang perlu diperhatikan untuk pengembangan selanjutnya. Sistem saat ini hanya mendukung tiga role pengguna yang mungkin perlu diperluas untuk mengakomodasi struktur organisasi yang lebih kompleks. Fitur notifikasi real-time belum diimplementasikan, yang dapat meningkatkan responsivitas sistem dalam menangani perubahan status kasus.

Untuk pengembangan selanjutnya, disarankan untuk mengimplementasikan fitur dashboard yang lebih interaktif dengan visualisasi data yang lebih menarik. Integrasi dengan sistem akademik yang sudah ada juga dapat meningkatkan efisiensi dalam pengelolaan data mahasiswa. Implementasi sistem backup dan recovery yang lebih robust juga diperlukan untuk menjamin ketersediaan data dalam kondisi darurat. Penambahan fitur audit trail yang lebih detail dapat membantu dalam pelacakan perubahan data dan meningkatkan akuntabilitas sistem.

4.8.6. Implikasi Terhadap Organisasi

Implementasi sistem SiPPAK diharapkan dapat memberikan dampak positif yang signifikan terhadap pengelolaan pelanggaran akademik di institusi pendidikan. Sistem digital yang terintegrasi dapat meningkatkan efisiensi dalam pencatatan dan pelacakan kasus pelanggaran, mengurangi

waktu yang diperlukan untuk mengakses informasi, dan meningkatkan akurasi data. Otomatisasi proses yang diimplementasikan dapat mengurangi beban kerja staff dan memungkinkan fokus yang lebih baik pada aspek pembinaan mahasiswa.

Dari perspektif manajemen, sistem ini menyediakan data yang lebih terstruktur dan mudah dianalisis untuk pengambilan keputusan kebijakan. Dashboard monitoring yang tersedia dapat membantu pimpinan dalam mengidentifikasi tren pelanggaran dan mengambil tindakan preventif yang tepat. Dokumentasi yang lebih baik dan terstruktur juga dapat meningkatkan transparansi dan akuntabilitas dalam penanganan kasus pelanggaran akademik.