4. IMPLEMENTASI SISTEM

Pada bab ini membahas mengenai implementasi sistem berdasarkan analisis kebutuhan dan desain sistem yang telah dibuat pada bab sebelumnya. Sistem dibuat dengan menggunakan bahasa pemrograman HTML, *JavaScript*, dan menggunakan *database* MySQL. Beberapa *tools,* yang digunakan antara lain: XAMPP, *browser* (Google Chrome), dan Visual Studio Code sebagai aplikasi *text editor*.

4.1. Pengaturan Koneksi Database

Konfigurasi koneksi program dengan *database* dapat dilihat pada Segmen Program 4.1.

Segmen Program 4.1. Konfigurasi Koneksi Program dengan *Database*

```
const mysql = require("mysql");

const pool = mysql.createPool({
  connectionLimit: 100, // default 10
  host: process.env.DB_HOST,
  password: process.env.DB_PASSWORD,
  user: process.env.DB_USER,
});

const connection = () => {
  return new Promise((resolve, reject) => {
    pool.getConnection((err, connection) => {
      if (err) reject(err);
      const query = (sql, binding) => {
        return new Promise((resolve, reject) => {
          connection.query(sql, binding, (err, result) => {
            if (err) reject(err);
            resolve(result);
          });
        });
      };
      const release = () => {
        return new Promise((resolve, reject) => {
          if (err) reject(err);
          resolve(connection.release());
        });
      };
      resolve({ query, release });
    });
  });
```

```
};

const query = (sql, binding) => {
  return new Promise((resolve, reject) => {
    pool.query(sql, binding, (err, result, fields) => {
      if (err) reject(err);
      resolve(result);
    });
  });
};

module.exports = { pool, connection, query };
```

4.2. Implementasi Program

Implementasi program merupakan tahap penerapan rancangan sistem sesuai dengan desain yang telah dibuat pada bab sebelumnya menjadi sebuah program. Pembuatan program dilakukan secara bertahap, mulai dari pembuatan menu *login* sampai dengan laporan.

4.2.1. *Login*

Segmen Program 4.2. *Login*

```
router.post("/login", async (req, res) => {
  const connection = await mysql.connection();

  await connection.query(`USE ${process.env.DB_NAME}`);

  // check company code
  let query = await connection.query(`
    SELECT * FROM company WHERE code = '${req.body.company_code}' AND is_deleted = 0
  `);

  // company not found
  if (query.length <= 0) {
    connection.release();
    return res.json({ message: "User not found", status: 500 });
  }

  await connection.query(`USE ${process.env.DB_NAME}_${req.body.company_code}`);

  // check username
  query = await connection.query(`
    SELECT * FROM user WHERE username = '${req.body.username}' AND is_deleted = 0
  `);

  // username not found
  if (query.length <= 0) {
    connection.release();
```

```
  return res.json({ message: "User not found", status: 500 });
}

// wrong password
if (!bcrypt.compareSync(req.body.password, query[0].password)) {
  connection.release();
  return res.json({ message: "User not found", status: 500 });
}

// success
const accessToken = generateAccessToken(
  Object.assign({ company_code: req.body.company_code }, query[0])
);
const refreshToken = generateRefreshToken(
  Object.assign({ company_code: req.body.company_code }, query[0])
);
let accessTokenExpiresIn = new Date();
accessTokenExpiresIn.setSeconds(
  accessTokenExpiresIn.getSeconds() +
    parseInt(process.env.ACCESS_TOKEN_EXPIRES_IN)
);
let refreshTokenExpiresIn = new Date();
refreshTokenExpiresIn.setSeconds(
  refreshTokenExpiresIn.getSeconds() +
    parseInt(process.env.REFRESH_TOKEN_EXPIRES_IN)
);

// set cookie
res.cookie("access-token", accessToken, {
  expires: accessTokenExpiresIn,
  httpOnly: true,
});

res.cookie("refresh-token", refreshToken, {
  expires: refreshTokenExpiresIn,
  httpOnly: true,
});

// get user role json
let user_role_json = await connection.query(
  `SELECT * FROM user_role_json WHERE id = ${query[0].role_id}`
);
connection.release();

return res.json({
  message: "Login success",
  payload: {
    company_code: req.body.company_code,
    id: query[0].id,
```

```
    username: query[0].username,
    role_id: query[0].role_id,
    role_name: user_role_json[0].name,
    role_json: user_role_json[0].json,
   },
   status: 200,
  });
});
```

Sistem akan mengecek *username* di *database* terlebih dahulu untuk mengetahui apakah ada atau tidak. Jika ada, sistem akan mengecek *password*. Jika *username* dan *password* benar, maka sistem akan membuat *token* lalu mengirim data pengguna. *Source code* dapat dilihat pada Segmen Program 4.2.

4.2.2. Menu Log Presensi

Segmen Program 4.3. *Add* Log Presensi

```
async (req, res) => {
  const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
  const connection = await mysql.connection();
  await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);

  await connection.query(`
    INSERT INTO attendance_log VALUES (
      null,
      ${req.body.user_id},
      STR_TO_DATE('${req.body.date}', '%Y-%m-%d %H:%i:%s'),
      0
    )
  `);

  connection.release();

  return res.json({
    message: "Add attendance log success",
    status: 200,
  });
 }
```

Ketika pengguna menekan tombol cek log, maka sistem akan memasukkan data waktu tersebut ke tabel log presensi. *Source code* dapat dilihat pada Segmen Program 4.3.

Segmen Program 4.4. *Get* Log Presensi

```
async (req, res) => {
  const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
```

```
  const connection = await mysql.connection();
  await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);

  let query = await connection.query(`
    SELECT al.*, u.username
    FROM attendance_log al
    LEFT JOIN user u
    ON al.user_id = u.id
    WHERE al.is_deleted = 0
  `);

  connection.release();

  return res.json({
    payload: query,
    status: 200,
  });
 }
```

Sistem mengirim data waktu dari *database* ke tabel log presensi yang berada di *website*. *Source code* dapat dilihat pada Segmen Program 4.4.

Segmen Program 4.5. *Delete* Log Presensi

```
async (req, res) => {
  const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
  const connection = await mysql.connection();
  await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);

  await connection.query(
    `UPDATE attendance_log SET is_deleted = 1 WHERE id = ${req.params.id}`
  );

  connection.release();

  return res.json({
    message: "Delete attendance log success",
    status: 200,
  });
 }
```

Sistem akan menghapus data yang berada di tabel log presensi dengan cara memperbarui (*soft delete*) data sesuai dengan ID data yang dihapus. *Source code* dapat dilihat pada Segmen Program 4.5.

4.2.3. Menu *Master Data* Perusahaan

Segmen Program 4.6. *Get Holiday*

```
async (req, res) => {
  // update path
  const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
  holidayPath = `public/uploads/${currentUser}/${holidayTableName}`;

  // check if template exist
  if (!isFoundFile(holidayPath, holidayTemplateFilename)) {
    return res.json({ status: 204 });
  }

  let filterModel = req.body.filterModel;
  let pageIndex = req.body.pageIndex;
  let rowsPerPage = req.body.rowsPerPage;
  let sortModel = req.body.sortModel;

  // get template column data
  const templateData = await getExcelTemplateData(
    holidayPath + "/" + holidayTemplateFilename,
    worksheetTemplateColumnTitleIndex,
    worksheetTemplateRowRuleIndex,
    worksheetTemplateRowTitleIndex
  );

  // add ID column
  templateData.columns.unshift("ID");
  templateData.fields.unshift("holiday_id");
  templateData.rules.unshift("id");

  let filterModelString = convertFilterModelToSQL(filterModel);
  let sortModelString = convertSortModelToSQL(sortModel);

  // fetch data
  const connection = await mysql.connection();
  await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);

  let query = await connection.query(`
    SELECT *
    FROM ${holidayTableName}
    ${filterModelString}
    ${sortModelString}
    LIMIT ${rowsPerPage}
    OFFSET ${pageIndex * rowsPerPage}
  `);

  // get filtered row count
```

```
  let rowCount = await connection.query(`
    SELECT COUNT(*) AS row_count
    FROM ${holidayTableName}
    ${filterModelString}
    ${sortModelString}
  `);

  connection.release();

  return res.json({
    payload: { query: query, rowCount: rowCount[0].row_count },
    status: 200,
    templateData: templateData,
  });
 }
```

Sistem akan mengecek apakah *template* ada. Jika ada, sistem akan mengambil data kolom, menambahkan kolom ID, mengambil data tersebut untuk diberikan ke pengguna, lalu mengirim data tersebut ke *website*. *Source code* dapat dilihat pada Segmen Program 4.6.

Segmen Program 4.7. *Import Template Holiday*

```
async (req, res) => {
  if (req.fileValidationError) {
    return res.json({
      message: req.fileValidationError,
      status: 500,
    });
  }

  // get uploaded template filename
  const filename = req.file.originalname;

  // check is template valid
  const isInvalidTemplate = await isInvalidExcelTemplateData(
    modules.HOLIDAY.key,
    tempPath + "/" + filename,
    worksheetTemplateColumnTitleIndex,
    worksheetTemplateRowRuleIndex,
    worksheetTemplateRowTitleIndex
  );

  if (isInvalidTemplate) {
    // delete uploaded template at temp folder
    fs.unlinkSync(tempPath + "/" + filename);

    return res.json({
      message: isInvalidTemplate,
```

```
    status: 500,
  });
}

// if template valid
// get template column data
const templateData = await getExcelTemplateData(
  tempPath + "/" + filename,
  worksheetTemplateColumnTitleIndex,
  worksheetTemplateRowRuleIndex,
  worksheetTemplateRowTitleIndex
);

// update path
const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
holidayPath = `public/uploads/${currentUser}/${holidayTableName}`;

// create if directory not exist
const dir = holidayPath;

if (!fs.existsSync(dir)) {
  fs.mkdirSync(dir, { recursive: true });
}

// copy template to correct path
fs.copyFileSync(
  tempPath + "/" + filename,
  holidayPath + "/" + holidayTemplateFilename
);

// delete uploaded template at temp folder
fs.unlinkSync(tempPath + "/" + filename);

// delete file data
if (fs.existsSync(holidayPath + "/" + holidayFilename)) {
  fs.unlinkSync(holidayPath + "/" + holidayFilename);
}

// generate SQL syntax
let sqlString = "";

for (let i = 0; i < templateData.columns.length; i++) {
  const field = templateData.fields[i];
  const dataType = convertRuleToSQL(templateData.rules[i]);

  sqlString += `${field} ${dataType},\n`;
}

const connection = await mysql.connection();
```

```
  await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);

  // drop table if exist
  await connection.query(`DROP TABLE IF EXISTS ${holidayTableName}`);

  // create table
  await connection.query(
    `CREATE TABLE IF NOT EXISTS ${holidayTableName} (
      ${holidayTableName}_id int NOT NULL AUTO_INCREMENT,
      ${sqlString}
      PRIMARY KEY (${holidayTableName}_id)
    )`
  );

  connection.release();

  return res.json({
    message: "Import template success",
    status: 200,
  });
 }
```

Sistem akan mengecek apakah format *template* valid. Jika format *template* tersebut valid, sistem akan mengambil data-data kolom tersebut untuk ditambahkan dalam bentuk SQL. *Source code* dapat dilihat pada Segmen Program 4.7.

Segmen Program 4.8. *Import Data Holiday*

```
async (req, res) => {
  if (req.fileValidationError) {
    return res.json({
      message: req.fileValidationError,
      status: 500,
    });
  }

  // update path
  const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
  holidayPath = `public/uploads/${currentUser}/${holidayTableName}`;

  // get uploaded data filename
  const filename = req.file.originalname;

  // check if template not exist
  if (!isFoundFile(holidayPath, holidayTemplateFilename)) {
    // delete uploaded file at temp folder
    fs.unlinkSync(tempPath + "/" + filename);
```

```
  return res.json({ message: "Template not found", status: 204 });
}

// get template column data
const templateData = await getExcelTemplateData(
  holidayPath + "/" + holidayTemplateFilename,
  worksheetTemplateColumnTitleIndex,
  worksheetTemplateRowRuleIndex,
  worksheetTemplateRowTitleIndex
);

// check is data valid
const isInvalidData = await isInvalidExcelData(
  templateData,
  tempPath + "/" + filename,
  worksheetDataColumnTitleIndex,
  worksheetDataRowDataIndex,
  worksheetDataRowTitleIndex
);

if (isInvalidData) {
  // delete uploaded template at temp folder
  fs.unlinkSync(tempPath + "/" + filename);

  return res.json({
    message: isInvalidData,
    status: 500,
  });
}

// if file valid
// get data
const data = await getExcelData(
  tempPath + "/" + filename,
  worksheetDataRowDataIndex
);

// create if directory not exist
const dir = holidayPath;

if (!fs.existsSync(dir)) {
  fs.mkdirSync(dir, { recursive: true });
}

// copy file to correct path
fs.copyFileSync(
  tempPath + "/" + filename,
  holidayPath + "/" + holidayFilename
);
```

```
// delete uploaded file at temp folder
fs.unlinkSync(tempPath + "/" + filename);

const connection = await mysql.connection();
await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);
await connection.query("START TRANSACTION");
// if replace data
if (req.query.isReplace === "true") {
  // empty table
  await connection.query(`TRUNCATE TABLE ${holidayTableName}`);
}

let sqlFieldString = "";
// init field string
for (let i = 0; i < templateData.columns.length; i++) {
  const field = templateData.fields[i];
  sqlFieldString += `${field}, `;
}

// remove comma
sqlFieldString = `${holidayTableName}_id, ` + sqlFieldString.slice(0, -2);

let sqlValueString = "";

// init value string
for (let i = 0; i < data.length; i++) {
  const row = data[i];
  sqlValueString = "";

  for (let j = 0; j < row.length; j++) {
    const cellValue = row[j];
    const dataType = convertRuleToSQL(templateData.rules[j]);

    // if cell value not null
    if (cellValue) {
      if (dataType === "DATE") {
        sqlValueString += `STR_TO_DATE('${dateFns.format(
          excelDateToJSDate(cellValue),
          "dd-MM-yyyy"
        )}', '%d-%m-%Y'), `;
      } else if (dataType === "DOUBLE") {
        sqlValueString += `${cellValue}, `;
      } else {
        sqlValueString += `'${cellValue}', `;
      }
    } else {
      if (dataType === "DATE") {
        sqlValueString += `NULL, `;
```

```
        } else if (dataType === "DOUBLE") {
          sqlValueString += `0, `;
        } else {
          sqlValueString += `'', `;
        }
       }
     }

     // remove comma
     sqlValueString = `null, ` + sqlValueString.slice(0, -2);
     // transaction
     try {
       await connection.query(
         `INSERT INTO ${holidayTableName} (${sqlFieldString})
         VALUES (${sqlValueString})`
       );
     } catch (error) {
       await connection.query("ROLLBACK");
       return res.json({
         message: "Import data failed",
         status: 400,
       });
     }
   }
   connection.query("COMMIT");
   connection.release();

   return res.json({
     message: "Import data success",
     status: 200,
   });
 }
```

Sistem akan mengecek apakah *template* ada. Jika ada, sistem akan mengambil data kolom, lalu sistem akan mengecek apakah data yang diberikan valid. Jika valid, kosongkan tabel sesuai data untuk diisi ulang, setelah itu sistem akan mengisi *database* sesuai data yang diberikan. *Source code* dapat dilihat pada Segmen Program 4.8.

Segmen Program 4.9. *Export Template Holiday*

```
async (req, res) => {
   // update path
   const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
   holidayPath = `public/uploads/${currentUser}/${holidayTableName}`;

   if (!isFoundFile(holidayPath, holidayTemplateFilename)) {
     return res.json({ message: "Template not found", status: 204 });
   }
```

```
  return res.download(
    holidayPath + "/" + holidayTemplateFilename,
    holidayTemplateFilename,
    (err) => {
      if (err) {
        res.json({ message: "Could not download the file", status: 500 });
      }
    }
  );
}
```

Sistem akan mengecek apakah data *template* hari libur ada. Jika ada, sistem akan mengirim data *template* hari libur. *Source code* dapat dilihat pada Segmen Program 4.9.

Segmen Program 4.10. *Export Data Holiday*

```
async (req, res) => {
  // update path
  const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
  holidayPath = `public/uploads/${currentUser}/${holidayTableName}`;

  let filterModel = req.body.filterModel;
  let columnVisibilityModel = req.body.columnVisibilityModel;
  let sortModel = req.body.sortModel;

  // get template column data
  const templateData = await getExcelTemplateData(
    holidayPath + "/" + holidayTemplateFilename,
    worksheetTemplateColumnTitleIndex,
    worksheetTemplateRowRuleIndex,
    worksheetTemplateRowTitleIndex
  );

  let columnVisibilityModelString = convertColumnVisibilityModelToSQL(
    columnVisibilityModel,
    templateData
  );
  let filterModelString = convertFilterModelToSQL(filterModel);
  let sortModelString = convertSortModelToSQL(sortModel);
  let templateDataAccordingRule =
    getExcelTemplateDataAccordingColumnVisibilityModel(
      columnVisibilityModel,
      templateData
    );

  const connection = await mysql.connection();
  await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);
```

```
  let query = await connection.query(`
    SELECT ${columnVisibilityModelString}
    FROM ${holidayTableName}
    ${filterModelString}
    ${sortModelString}
  `);

  connection.release();

  // if file exist
  if (isFoundFile(holidayPath, holidayFilename)) {
    // delete uploaded file
    fs.unlinkSync(holidayPath + "/" + holidayFilename);
  }

  // export file
  await exportExcelData(
    query,
    holidayPath + "/" + holidayFilename,
    templateDataAccordingRule,
    worksheetDataRowDataIndex
  );

  return res.download(
    holidayPath + "/" + holidayFilename,
    holidayFilename,
    (err) => {
      if (err) {
        res.json({ message: "Could not download the file", status: 500 });
      }
    }
  );
}
```

Sistem akan mengambil data *template* hari libur, lalu merangkai data sesuai *filter* yang digunakan, dan mengurutkan sesuai *sort* yang dipilih, setelah itu sistem akan mengirim data hari libur ke pengguna. *Source code* dapat dilihat pada Segmen Program 4.10.

Segmen Program 4.11. *Create Holiday*

```
async (req, res) => {
  // update path
  const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
  holidayPath = `public/uploads/${currentUser}/${holidayTableName}`;

  let newHolidayData = req.body.newHolidayData;

  // get template column data
```

```
const templateData = await getExcelTemplateData(
  holidayPath + "/" + holidayTemplateFilename,
  worksheetTemplateColumnTitleIndex,
  worksheetTemplateRowRuleIndex,
  worksheetTemplateRowTitleIndex
);

const isInvalidData = await isInvalidInsertData(
  templateData,
  newHolidayData
);

if (isInvalidData) {
  return res.json({
    message: isInvalidData,
    status: 500,
  });
}

// generate SQL syntax
let newHolidayDataValues = Object.values(newHolidayData);

// remove id field
newHolidayDataValues.shift();

let sqlFieldString = "";

// init field string
for (let i = 0; i < templateData.columns.length; i++) {
  const field = templateData.fields[i];
  sqlFieldString += `${field}, `;
}

// remove comma
sqlFieldString = `${holidayTableName}_id, ` + sqlFieldString.slice(0, -2);

let sqlValueString = "";

// init value string
for (let i = 0; i < newHolidayDataValues.length; i++) {
  const newHolidayDataValue = newHolidayDataValues[i];
  const dataType = convertRuleToSQL(templateData.rules[i]);

  if (newHolidayDataValue) {
    if (dataType === "DATE") {
      sqlValueString += `STR_TO_DATE('${dateFns.format(
        dateFns.parseISO(newHolidayDataValue),
        "dd-MM-yyyy"
      )}', '%d-%m-%Y'), `;
```

```
      } else if (dataType === "DOUBLE") {
        sqlValueString += `${newHolidayDataValue}, `;
      } else {
        sqlValueString += `'${newHolidayDataValue}', `;
      }
    } else {
      if (dataType === "DATE") {
        sqlValueString += `NULL, `;
      } else if (dataType === "DOUBLE") {
        sqlValueString += `0, `;
      } else {
        sqlValueString += `'', `;
      }
    }
  }

  // remove comma
  sqlValueString = `null, ` + sqlValueString.slice(0, -2);

  const connection = await mysql.connection();
  await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);
  await connection.query(`
    INSERT INTO ${holidayTableName} (${sqlFieldString})
    VALUES (
    ${sqlValueString}
    )
  `);
  connection.release();

  return res.json({
    message: "Create holiday success",
    status: 200,
  });
}
```

Sistem akan mengambil data kolom pada *template*, lalu sistem akan mengecek apakah data yang baru adalah valid. Jika data tersebut valid, sistem akan memasukkan data hari libur pada *database*. *Source code* dapat dilihat pada Segmen Program 4.11.

Segmen Program 4.12. *Update Holiday*

```
async (req, res) => {
  // update path
  const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
  holidayPath = `public/uploads/${currentUser}/${holidayTableName}`;

  let newHolidayData = req.body.newHolidayData;
```

```
// get template column data
const templateData = await getExcelTemplateData(
  holidayPath + "/" + holidayTemplateFilename,
  worksheetTemplateColumnTitleIndex,
  worksheetTemplateRowRuleIndex,
  worksheetTemplateRowTitleIndex
);

const isInvalidData = await isInvalidUpdateData(
  templateData,
  newHolidayData
);

if (isInvalidData) {
  return res.json({
    message: isInvalidData,
    status: 500,
  });
}

// generate SQL syntax
let newHolidayDataValues = Object.values(newHolidayData);

// remove id field
newHolidayDataValues.shift();

// generate SQL syntax
let sqlString = "";

for (let i = 0; i < newHolidayDataValues.length; i++) {
  const newHolidayDataValue = newHolidayDataValues[i];
  const dataType = convertRuleToSQL(templateData.rules[i]);

  sqlString += `${templateData.fields[i]} = `;

  if (dataType === "DATE") {
    if (newHolidayDataValue) {
      sqlString += `STR_TO_DATE('${dateFns.format(
        dateFns.parseISO(newHolidayDataValue),
        "dd-MM-yyyy"
      )}', '%d-%m-%Y'), `;
    } else {
      sqlString += `NULL, `;
    }
  } else if (dataType === "DOUBLE") {
    sqlString += `${newHolidayDataValue}, `;
  } else {
    sqlString += `'${newHolidayDataValue}', `;
  }
```

```
  }

  // remove comma
  sqlString = sqlString.slice(0, -2);

  const connection = await mysql.connection();
  await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);
  await connection.query(`
    UPDATE ${holidayTableName}
    SET ${sqlString}
    WHERE ${holidayTableName}_id = ${newHolidayData.holiday_id}
  `);
  connection.release();

  return res.json({
    message: "Update holiday success",
    status: 200,
  });
 }
```

Sistem akan mengambil data kolom pada *template*, lalu sistem akan mengecek apakah data yang baru adalah valid. Jika data tersebut valid, sistem akan memperbarui data pada *database*. *Source code* dapat dilihat pada Segmen Program 4.12.

Segmen Program 4.13. *Delete Holiday*

```
async (req, res) => {
  // update path
  const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
  holidayPath = `public/uploads/${currentUser}/${holidayTableName}`;

  let selectedHolidayID = req.body.selectedHolidayID;
  let sqlValueString = "";

  for (let i = 0; i < selectedHolidayID.length; i++) {
    sqlValueString += `${selectedHolidayID[i]}, `;
  }

  // remove comma
  sqlValueString = sqlValueString.slice(0, -2);

  const connection = await mysql.connection();
  await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);
  await connection.query(`
    DELETE FROM ${holidayTableName}
    WHERE ${holidayTableName}_id IN (${sqlValueString})
  `);
  connection.release();
```

```
  return res.json({
    message: "Delete holiday success",
    status: 200,
  });
 }
```

Sistem akan menghapus data yang terdapat pada *database* sesuai dengan ID yang dikirimkan pengguna dari *website*. *Source code* dapat dilihat pada Segmen Program 4.13.

Segmen Program 4.14. *Get Data Period*

```
async (req, res) => {
  // update path
  const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
  periodPath = `public/uploads/${currentUser}/${periodTableName}`;

  // check if template exist
  if (!isFoundFile(periodPath, periodTemplateFilename)) {
    return res.json({ status: 204 });
  }

  let filterModel = req.body.filterModel;
  let pageIndex = req.body.pageIndex;
  let rowsPerPage = req.body.rowsPerPage;
  let sortModel = req.body.sortModel;

  // get template column data
  const templateData = await getExcelTemplateData(
    periodPath + "/" + periodTemplateFilename,
    worksheetTemplateColumnTitleIndex,
    worksheetTemplateRowRuleIndex,
    worksheetTemplateRowTitleIndex
  );

  // add ID column
  templateData.columns.unshift("ID");
  templateData.fields.unshift("period_id");
  templateData.rules.unshift("id");

  let filterModelString = convertFilterModelToSQL(filterModel);
  let sortModelString = convertSortModelToSQL(sortModel);

  // fetch data
  const connection = await mysql.connection();
  await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);

  let query = await connection.query(`
    SELECT *
```

```
    FROM ${periodTableName}
    ${filterModelString}
    ${sortModelString}
    LIMIT ${rowsPerPage}
    OFFSET ${pageIndex * rowsPerPage}
  `);

  // get filtered row count
  let rowCount = await connection.query(`
    SELECT COUNT(*) AS row_count
    FROM ${periodTableName}
    ${filterModelString}
    ${sortModelString}
  `);

  connection.release();

  return res.json({
    payload: { query: query, rowCount: rowCount[0].row_count },
    status: 200,
    templateData: templateData,
  });
 }
```

Sistem akan mengecek apakah *template* ada. Jika ada, sistem akan mengambil data kolom, menambahkan kolom ID, mengambil data tersebut untuk diberikan ke pengguna, lalu mengirim data tersebut ke *website*. Source code dapat dilihat pada Segmen Program 4.14.

Segmen Program 4.15. *Import Template Period*

```
async (req, res) => {
  if (req.fileValidationError) {
    return res.json({
      message: req.fileValidationError,
      status: 500,
    });
  }

  // get uploaded template filename
  const filename = req.file.originalname;

  // check is template valid
  const isInvalidTemplate = await isInvalidExcelTemplateData(
    modules.PERIOD.key,
    tempPath + "/" + filename,
    worksheetTemplateColumnTitleIndex,
    worksheetTemplateRowRuleIndex,
    worksheetTemplateRowTitleIndex
```

```
);

if (isInvalidTemplate) {
  // delete uploaded template at temp folder
  fs.unlinkSync(tempPath + "/" + filename);

  return res.json({
    message: isInvalidTemplate,
    status: 500,
  });
}

// if template valid
// get template column data
const templateData = await getExcelTemplateData(
  tempPath + "/" + filename,
  worksheetTemplateColumnTitleIndex,
  worksheetTemplateRowRuleIndex,
  worksheetTemplateRowTitleIndex
);

// update path
const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
periodPath = `public/uploads/${currentUser}/${periodTableName}`;

// create if directory not exist
const dir = periodPath;

if (!fs.existsSync(dir)) {
  fs.mkdirSync(dir, { recursive: true });
}

// copy template to correct path
fs.copyFileSync(
  tempPath + "/" + filename,
  periodPath + "/" + periodTemplateFilename
);

// delete uploaded template at temp folder
fs.unlinkSync(tempPath + "/" + filename);

// delete file data
if (fs.existsSync(periodPath + "/" + periodFilename)) {
  fs.unlinkSync(periodPath + "/" + periodFilename);
}

// generate SQL syntax
let sqlString = "";
```

```
  for (let i = 0; i < templateData.columns.length; i++) {
    const field = templateData.fields[i];
    const dataType = convertRuleToSQL(templateData.rules[i]);

    sqlString += `${field} ${dataType},\n`;
  }

  const connection = await mysql.connection();
  await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);

  // drop table if exist
  await connection.query(`DROP TABLE IF EXISTS ${periodTableName}`);

  // create table
  await connection.query(
    `CREATE TABLE IF NOT EXISTS ${periodTableName} (
      ${periodTableName}_id int NOT NULL AUTO_INCREMENT,
      ${sqlString}
      PRIMARY KEY (${periodTableName}_id)
    )`
  );

  connection.release();

  return res.json({
    message: "Import template success",
    status: 200,
  });
 }
```

Sistem akan mengecek apakah format *template* valid. Jika format *template* tersebut valid, sistem akan mengambil data-data kolom tersebut untuk ditambahkan dalam bentuk SQL. *Source code* dapat dilihat pada Segmen Program 4.15.

Segmen Program 4.16. *Import Data Period*

```
async (req, res) => {
  if (req.fileValidationError) {
    return res.json({
      message: req.fileValidationError,
      status: 500,
    });
  }

  // update path
  const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
  periodPath = `public/uploads/${currentUser}/${periodTableName}`;
```

```
// get uploaded data filename
const filename = req.file.originalname;

// check if template not exist
if (!isFoundFile(periodPath, periodTemplateFilename)) {
  // delete uploaded file at temp folder
  fs.unlinkSync(tempPath + "/" + filename);

  return res.json({ message: "Template not found", status: 204 });
}

// get template column data
const templateData = await getExcelTemplateData(
  periodPath + "/" + periodTemplateFilename,
  worksheetTemplateColumnTitleIndex,
  worksheetTemplateRowRuleIndex,
  worksheetTemplateRowTitleIndex
);

// check is data valid
const isInvalidData = await isInvalidExcelData(
  templateData,
  tempPath + "/" + filename,
  worksheetDataColumnTitleIndex,
  worksheetDataRowDataIndex,
  worksheetDataRowTitleIndex
);

if (isInvalidData) {
  // delete uploaded template at temp folder
  fs.unlinkSync(tempPath + "/" + filename);

  return res.json({
    message: isInvalidData,
    status: 500,
  });
}

// if file valid
// get data
const data = await getExcelData(
  tempPath + "/" + filename,
  worksheetDataRowDataIndex
);

// create if directory not exist
const dir = periodPath;

if (!fs.existsSync(dir)) {
```

```
  fs.mkdirSync(dir, { recursive: true });
}

// copy file to correct path
fs.copyFileSync(
  tempPath + "/" + filename,
  periodPath + "/" + periodFilename
);

// delete uploaded file at temp folder
fs.unlinkSync(tempPath + "/" + filename);

const connection = await mysql.connection();
await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);
await connection.query("START TRANSACTION");
// if replace data
if (req.query.isReplace === "true") {
  // empty table
  await connection.query(`TRUNCATE TABLE ${periodTableName}`);
}

let sqlFieldString = "";
// init field string
for (let i = 0; i < templateData.columns.length; i++) {
  const field = templateData.fields[i];
  sqlFieldString += `${field}, `;
}

// remove comma
sqlFieldString = `${periodTableName}_id, ` + sqlFieldString.slice(0, -2);

let sqlValueString = "";

// init value string
for (let i = 0; i < data.length; i++) {
  const row = data[i];
  sqlValueString = "";

  for (let j = 0; j < row.length; j++) {
    const cellValue = row[j];
    const dataType = convertRuleToSQL(templateData.rules[j]);

    // if cell value not null
    if (cellValue) {
      if (dataType === "DATE") {
        sqlValueString += `STR_TO_DATE('${dateFns.format(
          excelDateToJSDate(cellValue),
          "dd-MM-yyyy"
        )}', '%d-%m-%Y'), `;
```

```
        } else if (dataType === "DOUBLE") {
          sqlValueString += `${cellValue}, `;
        } else {
          sqlValueString += `'${cellValue}', `;
        }
      } else {
        if (dataType === "DATE") {
          sqlValueString += `NULL, `;
        } else if (dataType === "DOUBLE") {
          sqlValueString += `0, `;
        } else {
          sqlValueString += `'', `;
        }
      }
    }

    // remove comma
    sqlValueString = `null, ` + sqlValueString.slice(0, -2);
    // transaction
    try {
      await connection.query(
        `INSERT INTO ${periodTableName} (${sqlFieldString})
        VALUES (${sqlValueString})`
      );
    } catch (error) {
      await connection.query("ROLLBACK");
      return res.json({
        message: "Import data failed",
        status: 400,
      });
    }
  }

  connection.query("COMMIT");
  connection.release();

  return res.json({
    message: "Import data success",
    status: 200,
  });
}
```

Sistem akan mengecek apakah *template* ada. Jika ada, sistem akan mengambil data kolom, lalu sistem akan mengecek apakah data yang diberikan valid. Jika valid, kosongkan tabel sesuai data untuk diisi ulang, setelah itu sistem akan mengisi *database* sesuai data yang diberikan. *Source code* dapat dilihat pada Segmen Program 4.16.

Segmen Program 4.17. *Export Template Period*

```
async (req, res) => {
   // update path
   const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
   periodPath = `public/uploads/${currentUser}/${periodTableName}`;

   if (!isFoundFile(periodPath, periodTemplateFilename)) {
     return res.json({ message: "Template not found", status: 204 });
   }

   return res.download(
     periodPath + "/" + periodTemplateFilename,
     periodTemplateFilename,
     (err) => {
       if (err) {
         res.json({ message: "Could not download the file", status: 500 });
       }
     }
   );
 }
```

Sistem akan mengecek apakah data *template* periode *cut-off* ada. Jika ada, sistem akan mengirim data *template* periode *cut-off*. *Source code* dapat dilihat pada Segmen Program 4.17.

Segmen Program 4.18. *Export Data Period*

```
async (req, res) => {
   // update path
   const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
   periodPath = `public/uploads/${currentUser}/${periodTableName}`;

   let filterModel = req.body.filterModel;
   let columnVisibilityModel = req.body.columnVisibilityModel;
   let sortModel = req.body.sortModel;

   // get template column data
   const templateData = await getExcelTemplateData(
     periodPath + "/" + periodTemplateFilename,
     worksheetTemplateColumnTitleIndex,
     worksheetTemplateRowRuleIndex,
     worksheetTemplateRowTitleIndex
   );

   let columnVisibilityModelString = convertColumnVisibilityModelToSQL(
     columnVisibilityModel,
     templateData
   );
   let filterModelString = convertFilterModelToSQL(filterModel);
```

```
  let sortModelString = convertSortModelToSQL(sortModel);
  let templateDataAccordingRule =
   getExcelTemplateDataAccordingColumnVisibilityModel(
    columnVisibilityModel,
    templateData
   );

  const connection = await mysql.connection();
  await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);

  let query = await connection.query(`
   SELECT ${columnVisibilityModelString}
   FROM ${periodTableName}
   ${filterModelString}
   ${sortModelString}
  `);

  connection.release();

  // if file exist
  if (isFoundFile(periodPath, periodFilename)) {
   // delete uploaded file
   fs.unlinkSync(periodPath + "/" + periodFilename);
  }

  // export file
  await exportExcelData(
   query,
   periodPath + "/" + periodFilename,
   templateDataAccordingRule,
   worksheetDataRowDataIndex
  );

  return res.download(
   periodPath + "/" + periodFilename,
   periodFilename,
   (err) => {
    if (err) {
     res.json({ message: "Could not download the file", status: 500 });
    }
   }
  );
 }
```

Sistem akan mengambil data *template* periode *cut-off*, lalu merangkai data sesuai *filter* yang digunakan, dan mengurutkan sesuai *sort* yang dipilih, setelah itu sistem akan mengirim data periode *cut-off ke* pengguna. *Source code* dapat dilihat pada Segmen Program 4.18.

Segmen Program 4.19. *Create Period*

```
async (req, res) => {
  // update path
  const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
  periodPath = `public/uploads/${currentUser}/${periodTableName}`;

  let newPeriodData = req.body.newPeriodData;

  // get template column data
  const templateData = await getExcelTemplateData(
    periodPath + "/" + periodTemplateFilename,
    worksheetTemplateColumnTitleIndex,
    worksheetTemplateRowRuleIndex,
    worksheetTemplateRowTitleIndex
  );

  const isInvalidData = await isInvalidInsertData(
    templateData,
    newPeriodData
  );

  if (isInvalidData) {
    return res.json({
      message: isInvalidData,
      status: 500,
    });
  }

  // generate SQL syntax
  let newPeriodDataValues = Object.values(newPeriodData);

  // remove id field
  newPeriodDataValues.shift();

  let sqlFieldString = "";

  // init field string
  for (let i = 0; i < templateData.columns.length; i++) {
    const field = templateData.fields[i];
    sqlFieldString += `${field}, `;
  }

  // remove comma
  sqlFieldString = `${periodTableName}_id, ` + sqlFieldString.slice(0, -2);

  let sqlValueString = "";

  // init value string
  for (let i = 0; i < newPeriodDataValues.length; i++) {
```

```
    const newPeriodDataValue = newPeriodDataValues[i];
    const dataType = convertRuleToSQL(templateData.rules[i]);

    if (newPeriodDataValue) {
      if (dataType === "DATE") {
        sqlValueString += `STR_TO_DATE('${dateFns.format(
          dateFns.parseISO(newPeriodDataValue),
          "dd-MM-yyyy"
        )}', '%d-%m-%Y'), `;
      } else if (dataType === "DOUBLE") {
        sqlValueString += `${newPeriodDataValue}, `;
      } else {
        sqlValueString += `'${newPeriodDataValue}', `;
      }
    } else {
      if (dataType === "DATE") {
        sqlValueString += `NULL, `;
      } else if (dataType === "DOUBLE") {
        sqlValueString += `0, `;
      } else {
        sqlValueString += `'', `;
      }
    }
  }

  // remove comma
  sqlValueString = `null, ` + sqlValueString.slice(0, -2);

  const connection = await mysql.connection();
  await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);
  await connection.query(`
    INSERT INTO ${periodTableName} (${sqlFieldString})
    VALUES (
    ${sqlValueString}
    )
  `);
  connection.release();

  return res.json({
    message: "Create period success",
    status: 200,
  });
}
```

Sistem akan mengambil data kolom pada *template*, lalu sistem akan mengecek apakah data yang baru adalah valid. Jika data tersebut valid, sistem akan memasukkan data periode *cut-off* pada *database*. *Source code* dapat dilihat pada Segmen Program 4.19.

Segmen Program 4.20. *Update Period*

```
async (req, res) => {
  // update path
  const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
  periodPath = `public/uploads/${currentUser}/${periodTableName}`;

  let newPeriodData = req.body.newPeriodData;

  // get template column data
  const templateData = await getExcelTemplateData(
    periodPath + "/" + periodTemplateFilename,
    worksheetTemplateColumnTitleIndex,
    worksheetTemplateRowRuleIndex,
    worksheetTemplateRowTitleIndex
  );

  const isInvalidData = await isInvalidUpdateData(
    templateData,
    newPeriodData
  );

  if (isInvalidData) {
    return res.json({
      message: isInvalidData,
      status: 500,
    });
  }

  // generate SQL syntax
  let newPeriodDataValues = Object.values(newPeriodData);

  // remove id field
  newPeriodDataValues.shift();

  // generate SQL syntax
  let sqlString = "";

  for (let i = 0; i < newPeriodDataValues.length; i++) {
    const newPeriodDataValue = newPeriodDataValues[i];
    const dataType = convertRuleToSQL(templateData.rules[i]);

    sqlString += `${templateData.fields[i]} = `;

    if (dataType === "DATE") {
      if (newPeriodDataValue) {
        sqlString += `STR_TO_DATE('${dateFns.format(
          dateFns.parseISO(newPeriodDataValue),
          "dd-MM-yyyy"
        )}', '%d-%m-%Y'), `;
```

```
      } else {
        sqlString += `NULL, `;
      }
    } else if (dataType === "DOUBLE") {
      sqlString += `${newPeriodDataValue}, `;
    } else {
      sqlString += `'${newPeriodDataValue}', `;
    }
  }

  // remove comma
  sqlString = sqlString.slice(0, -2);

  const connection = await mysql.connection();
  await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);
  await connection.query(`
    UPDATE ${periodTableName}
    SET ${sqlString}
    WHERE ${periodTableName}_id = ${newPeriodData.period_id}
  `);
  connection.release();

  return res.json({
    message: "Update period success",
    status: 200,
  });
 }
```

Sistem akan mengambil data kolom pada *template*, lalu sistem akan mengecek apakah data yang baru adalah valid. Jika data tersebut valid, sistem akan memperbarui data pada *database*. *Source code* dapat dilihat pada Segmen Program 4.20.

Segmen Program 4.21. *Delete Period*

```
async (req, res) => {
  // update path
  const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
  periodPath = `public/uploads/${currentUser}/${periodTableName}`;

  let selectedPeriodID = req.body.selectedPeriodID;
  let sqlValueString = "";

  for (let i = 0; i < selectedPeriodID.length; i++) {
    sqlValueString += `${selectedPeriodID[i]}, `;
  }

  // remove comma
  sqlValueString = sqlValueString.slice(0, -2);
```

```
  const connection = await mysql.connection();
  await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);
  await connection.query(`
    DELETE FROM ${periodTableName}
    WHERE ${periodTableName}_id IN (${sqlValueString})
  `);
  connection.release();

  return res.json({
    message: "Delete period success",
    status: 200,
  });
 }
```

Sistem akan menghapus data yang terdapat pada *database* sesuai dengan ID yang dikirimkan pengguna dari *website*. *Source code* dapat dilihat pada Segmen Program 4.21.

Segmen Program 4.22. *Get Workday*

```
async (req, res) => {
  // update path
  const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
  workdayPath = `public/uploads/${currentUser}/${workdayTableName}`;

  // check if template exist
  if (!isFoundFile(workdayPath, workdayTemplateFilename)) {
    return res.json({ status: 204 });
  }

  let filterModel = req.body.filterModel;
  let pageIndex = req.body.pageIndex;
  let rowsPerPage = req.body.rowsPerPage;
  let sortModel = req.body.sortModel;

  // get template column data
  const templateData = await getExcelTemplateData(
    workdayPath + "/" + workdayTemplateFilename,
    worksheetTemplateColumnTitleIndex,
    worksheetTemplateRowRuleIndex,
    worksheetTemplateRowTitleIndex
  );

  // add ID column
  templateData.columns.unshift("ID");
  templateData.fields.unshift("workday_id");
  templateData.rules.unshift("id");

  let filterModelString = convertFilterModelToSQL(filterModel);
```

```
  let sortModelString = convertSortModelToSQL(sortModel);

  // fetch data
  const connection = await mysql.connection();
  await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);

  let query = await connection.query(`
    SELECT *
    FROM ${workdayTableName}
    ${filterModelString}
    ${sortModelString}
    LIMIT ${rowsPerPage}
    OFFSET ${pageIndex * rowsPerPage}
  `);

  // get filtered row count
  let rowCount = await connection.query(`
    SELECT COUNT(*) AS row_count
    FROM ${workdayTableName}
    ${filterModelString}
    ${sortModelString}
  `);

  connection.release();

  return res.json({
    payload: { query: query, rowCount: rowCount[0].row_count },
    status: 200,
    templateData: templateData,
  });
 }
```

Sistem akan mengecek apakah *template* ada. Jika ada, sistem akan mengambil data kolom, menambahkan kolom ID, mengambil data tersebut untuk diberikan ke pengguna, lalu mengirim data tersebut ke *website*. *Source code* dapat dilihat pada Segmen Program 4.22.

Segmen Program 4.23. *Import Template Workday*

```
async (req, res) => {
  if (req.fileValidationError) {
    return res.json({
      message: req.fileValidationError,
      status: 500,
    });
  }

  // get uploaded template filename
  const filename = req.file.originalname;
```

```
// check is template valid
const isInvalidTemplate = await isInvalidExcelTemplateData(
  modules.WORKDAY.key,
  tempPath + "/" + filename,
  worksheetTemplateColumnTitleIndex,
  worksheetTemplateRowRuleIndex,
  worksheetTemplateRowTitleIndex
);

if (isInvalidTemplate) {
  // delete uploaded template at temp folder
  fs.unlinkSync(tempPath + "/" + filename);

  return res.json({
    message: isInvalidTemplate,
    status: 500,
  });
}

// if template valid
// get template column data
const templateData = await getExcelTemplateData(
  tempPath + "/" + filename,
  worksheetTemplateColumnTitleIndex,
  worksheetTemplateRowRuleIndex,
  worksheetTemplateRowTitleIndex
);

// update path
const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
workdayPath = `public/uploads/${currentUser}/${workdayTableName}`;

// create if directory not exist
const dir = workdayPath;

if (!fs.existsSync(dir)) {
  fs.mkdirSync(dir, { recursive: true });
}

// copy template to correct path
fs.copyFileSync(
  tempPath + "/" + filename,
  workdayPath + "/" + workdayTemplateFilename
);

// delete uploaded template at temp folder
fs.unlinkSync(tempPath + "/" + filename);
```

```
  // delete file data
  if (fs.existsSync(workdayPath + "/" + workdayFilename)) {
    fs.unlinkSync(workdayPath + "/" + workdayFilename);
  }

  // generate SQL syntax
  let sqlString = "";

  for (let i = 0; i < templateData.columns.length; i++) {
    const field = templateData.fields[i];
    const dataType = convertRuleToSQL(templateData.rules[i]);

    sqlString += `${field} ${dataType},\n`;
  }

  const connection = await mysql.connection();
  await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);

  // drop table if exist
  await connection.query(`DROP TABLE IF EXISTS ${workdayTableName}`);

  // create table
  await connection.query(
    `CREATE TABLE IF NOT EXISTS ${workdayTableName} (
      ${workdayTableName}_id int NOT NULL AUTO_INCREMENT,
      ${sqlString}
      PRIMARY KEY (${workdayTableName}_id)
    )`
  );

  connection.release();

  return res.json({
    message: "Import template success",
    status: 200,
  });
 }
```

Sistem akan mengecek apakah format *template* valid. Jika format *template* tersebut valid, sistem akan mengambil data-data kolom tersebut untuk ditambahkan dalam bentuk SQL. *Source code* dapat dilihat pada Segmen Program 4.23.

Segmen Program 4.24. *Import Data Workday*

```
async (req, res) => {
  if (req.fileValidationError) {
    return res.json({
      message: req.fileValidationError,
```

```
    status: 500,
  });
}

// update path
const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
workdayPath = `public/uploads/${currentUser}/${workdayTableName}`;

// get uploaded data filename
const filename = req.file.originalname;

// check if template not exist
if (!isFoundFile(workdayPath, workdayTemplateFilename)) {
  // delete uploaded file at temp folder
  fs.unlinkSync(tempPath + "/" + filename);

  return res.json({ message: "Template not found", status: 204 });
}

// get template column data
const templateData = await getExcelTemplateData(
  workdayPath + "/" + workdayTemplateFilename,
  worksheetTemplateColumnTitleIndex,
  worksheetTemplateRowRuleIndex,
  worksheetTemplateRowTitleIndex
);

// check is data valid
const isInvalidData = await isInvalidExcelData(
  templateData,
  tempPath + "/" + filename,
  worksheetDataColumnTitleIndex,
  worksheetDataRowDataIndex,
  worksheetDataRowTitleIndex
);

if (isInvalidData) {
  // delete uploaded template at temp folder
  fs.unlinkSync(tempPath + "/" + filename);

  return res.json({
    message: isInvalidData,
    status: 500,
  });
}

// if file valid
// get data
const data = await getExcelData(
```

```
  tempPath + "/" + filename,
  worksheetDataRowDataIndex
);

// create if directory not exist
const dir = workdayPath;

if (!fs.existsSync(dir)) {
  fs.mkdirSync(dir, { recursive: true });
}

// copy file to correct path
fs.copyFileSync(
  tempPath + "/" + filename,
  workdayPath + "/" + workdayFilename
);

// delete uploaded file at temp folder
fs.unlinkSync(tempPath + "/" + filename);

const connection = await mysql.connection();
await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);
await connection.query("START TRANSACTION");
// if replace data
if (req.query.isReplace === "true") {
  // empty table
  await connection.query(`TRUNCATE TABLE ${workdayTableName}`);
}

let sqlFieldString = "";
// init field string
for (let i = 0; i < templateData.columns.length; i++) {
  const field = templateData.fields[i];
  sqlFieldString += `${field}, `;
}

// remove comma
sqlFieldString = `${workdayTableName}_id, ` + sqlFieldString.slice(0, -2);

let sqlValueString = "";

// init value string
for (let i = 0; i < data.length; i++) {
  const row = data[i];
  sqlValueString = "";

  for (let j = 0; j < row.length; j++) {
    const cellValue = row[j];
    const dataType = convertRuleToSQL(templateData.rules[j]);
```

```
      // if cell value not null
      if (cellValue) {
        if (dataType === "DATE") {
          sqlValueString += `STR_TO_DATE('${dateFns.format(
            excelDateToJSDate(cellValue),
            "dd-MM-yyyy"
          )}', '%d-%m-%Y'), `;
        } else if (dataType === "DOUBLE") {
          sqlValueString += `${cellValue}, `;
        } else {
          sqlValueString += `'${cellValue}', `;
        }
      } else {
        if (dataType === "DATE") {
          sqlValueString += `NULL, `;
        } else if (dataType === "DOUBLE") {
          sqlValueString += `0, `;
        } else {
          sqlValueString += `'', `;
        }
      }
    }

    // remove comma
    sqlValueString = `null, ` + sqlValueString.slice(0, -2);
    // transaction
    try {
      await connection.query(
        `INSERT INTO ${workdayTableName} (${sqlFieldString})
        VALUES (${sqlValueString})`
      );
    } catch (error) {
      await connection.query("ROLLBACK");
      return res.json({
        message: "Import data failed",
        status: 400,
      });
    }
  }

  connection.query("COMMIT");
  connection.release();

  return res.json({
    message: "Import data success",
    status: 200,
  });
}
```

Sistem akan mengecek apakah *template* ada. Jika ada, sistem akan mengambil data kolom, lalu sistem akan mengecek apakah data yang diberikan valid. Jika valid, kosongkan tabel sesuai data untuk diisi ulang, setelah itu sistem akan mengisi *database* sesuai data yang diberikan. *Source code* dapat dilihat pada Segmen Program 4.24.

Segmen Program 4.25. *Export Template Workday*

```
async (req, res) => {
  // update path
  const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
  workdayPath = `public/uploads/${currentUser}/${workdayTableName}`;

  if (!isFoundFile(workdayPath, workdayTemplateFilename)) {
    return res.json({ message: "Template not found", status: 204 });
  }

  return res.download(
    workdayPath + "/" + workdayTemplateFilename,
    workdayTemplateFilename,
    (err) => {
      if (err) {
        res.json({ message: "Could not download the file", status: 500 });
      }
    }
  );
}
```

Sistem akan mengecek apakah data *template* hari kerja ada. Jika ada, sistem akan mengirim data *template* hari kerja. *Source code* dapat dilihat pada Segmen Program 4.25.

Segmen Program 4.26. *Export Data Workday*

```
async (req, res) => {
  // update path
  const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
  workdayPath = `public/uploads/${currentUser}/${workdayTableName}`;

  let filterModel = req.body.filterModel;
  let columnVisibilityModel = req.body.columnVisibilityModel;
  let sortModel = req.body.sortModel;

  // get template column data
  const templateData = await getExcelTemplateData(
    workdayPath + "/" + workdayTemplateFilename,
    worksheetTemplateColumnTitleIndex,
    worksheetTemplateRowRuleIndex,
```

```
  worksheetTemplateRowTitleIndex
);

let columnVisibilityModelString = convertColumnVisibilityModelToSQL(
  columnVisibilityModel,
  templateData
);
let filterModelString = convertFilterModelToSQL(filterModel);
let sortModelString = convertSortModelToSQL(sortModel);
let templateDataAccordingRule =
  getExcelTemplateDataAccordingColumnVisibilityModel(
    columnVisibilityModel,
    templateData
  );

const connection = await mysql.connection();
await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);

let query = await connection.query(`
  SELECT ${columnVisibilityModelString}
  FROM ${workdayTableName}
  ${filterModelString}
  ${sortModelString}
`);

connection.release();

// if file exist
if (isFoundFile(workdayPath, workdayFilename)) {
  // delete uploaded file
  fs.unlinkSync(workdayPath + "/" + workdayFilename);
}

// export file
await exportExcelData(
  query,
  workdayPath + "/" + workdayFilename,
  templateDataAccordingRule,
  worksheetDataRowDataIndex
);

return res.download(
  workdayPath + "/" + workdayFilename,
  workdayFilename,
  (err) => {
    if (err) {
      res.json({ message: "Could not download the file", status: 500 });
    }
  }
```

```
  );
 }
```

Sistem akan mengambil data *template* hari kerja, lalu merangkai data sesuai *filter* yang digunakan, dan mengurutkan sesuai *sort* yang dipilih, setelah itu sistem akan mengirim data hari kerja ke pengguna. *Source code* dapat dilihat pada Segmen Program 4.26.

Segmen Program 4.27. *Create Workday*

```
async (req, res) => {
  // update path
  const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
  workdayPath = `public/uploads/${currentUser}/${workdayTableName}`;

  let newWorkdayData = req.body.newWorkdayData;

  // get template column data
  const templateData = await getExcelTemplateData(
    workdayPath + "/" + workdayTemplateFilename,
    worksheetTemplateColumnTitleIndex,
    worksheetTemplateRowRuleIndex,
    worksheetTemplateRowTitleIndex
  );

  const isInvalidData = await isInvalidInsertData(
    templateData,
    newWorkdayData
  );

  if (isInvalidData) {
    return res.json({
      message: isInvalidData,
      status: 500,
    });
  }

  // generate SQL syntax
  let newWorkdayDataValues = Object.values(newWorkdayData);

  // remove id field
  newWorkdayDataValues.shift();

  let sqlFieldString = "";

  // init field string
  for (let i = 0; i < templateData.columns.length; i++) {
    const field = templateData.fields[i];
    sqlFieldString += `${field}, `;
```

```
}

// remove comma
sqlFieldString = `${workdayTableName}_id, ` + sqlFieldString.slice(0, -2);

let sqlValueString = "";

// init value string
for (let i = 0; i < newWorkdayDataValues.length; i++) {
  const newWorkdayDataValue = newWorkdayDataValues[i];
  const dataType = convertRuleToSQL(templateData.rules[i]);

  if (newWorkdayDataValue) {
    if (dataType === "DATE") {
      sqlValueString += `STR_TO_DATE('${dateFns.format(
        dateFns.parseISO(newWorkdayDataValue),
        "dd-MM-yyyy"
      )}', '%d-%m-%Y'), `;
    } else if (dataType === "DOUBLE") {
      sqlValueString += `${newWorkdayDataValue}, `;
    } else {
      sqlValueString += `'${newWorkdayDataValue}', `;
    }
  } else {
    if (dataType === "DATE") {
      sqlValueString += `NULL, `;
    } else if (dataType === "DOUBLE") {
      sqlValueString += `0, `;
    } else {
      sqlValueString += `'', `;
    }
  }
}

// remove comma
sqlValueString = `null, ` + sqlValueString.slice(0, -2);

const connection = await mysql.connection();
await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);
await connection.query(`
  INSERT INTO ${workdayTableName} (${sqlFieldString})
  VALUES (
  ${sqlValueString}
  )
`);
connection.release();

return res.json({
  message: "Create workday success",
```

```
    status: 200,
   });
  }
```

Sistem akan mengambil data kolom pada *template*, lalu sistem akan mengecek apakah data yang baru adalah valid. Jika data tersebut valid, sistem akan memasukkan data hari kerja pada *database*. *Source code* dapat dilihat pada Segmen Program 4.27.

Segmen Program 4.28. *Update Workday*

```
async (req, res) => {
   // update path
   const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
   workdayPath = `public/uploads/${currentUser}/${workdayTableName}`;

   let newWorkdayData = req.body.newWorkdayData;

   // get template column data
   const templateData = await getExcelTemplateData(
    workdayPath + "/" + workdayTemplateFilename,
    worksheetTemplateColumnTitleIndex,
    worksheetTemplateRowRuleIndex,
    worksheetTemplateRowTitleIndex
   );

   const isInvalidData = await isInvalidUpdateData(
    templateData,
    newWorkdayData
   );

   if (isInvalidData) {
    return res.json({
     message: isInvalidData,
     status: 500,
    });
   }

   // generate SQL syntax
   let newWorkdayDataValues = Object.values(newWorkdayData);

   // remove id field
   newWorkdayDataValues.shift();

   // generate SQL syntax
   let sqlString = "";

   for (let i = 0; i < newWorkdayDataValues.length; i++) {
    const newWorkdayDataValue = newWorkdayDataValues[i];
```

```
    const dataType = convertRuleToSQL(templateData.rules[i]);

    sqlString += `${templateData.fields[i]} = `;

    if (dataType === "DATE") {
      if (newWorkdayDataValue) {
        sqlString += `STR_TO_DATE('${dateFns.format(
          dateFns.parseISO(newWorkdayDataValue),
          "dd-MM-yyyy"
        )}', '%d-%m-%Y'), `;
      } else {
        sqlString += `NULL, `;
      }
    } else if (dataType === "DOUBLE") {
      sqlString += `${newWorkdayDataValue}, `;
    } else {
      sqlString += `'${newWorkdayDataValue}', `;
    }
  }

  // remove comma
  sqlString = sqlString.slice(0, -2);

  const connection = await mysql.connection();
  await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);
  await connection.query(`
    UPDATE ${workdayTableName}
    SET ${sqlString}
    WHERE ${workdayTableName}_id = ${newWorkdayData.workday_id}
  `);
  connection.release();

  return res.json({
    message: "Update workday success",
    status: 200,
  });
 }
```

Sistem akan mengambil data kolom pada *template*, lalu sistem akan mengecek apakah data yang baru adalah valid. Jika data tersebut valid, sistem akan memperbarui data pada *database*. *Source code* dapat dilihat pada Segmen Program 4.28.

Segmen Program 4.29. *Delete Workday*

```
async (req, res) => {
  // update path
  const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
  workdayPath = `public/uploads/${currentUser}/${workdayTableName}`;
```

```
  let selectedWorkdayID = req.body.selectedWorkdayID;
  let sqlValueString = "";

  for (let i = 0; i < selectedWorkdayID.length; i++) {
    sqlValueString += `${selectedWorkdayID[i]}, `;
  }

  // remove comma
  sqlValueString = sqlValueString.slice(0, -2);

  const connection = await mysql.connection();
  await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);
  await connection.query(`
    DELETE FROM ${workdayTableName}
    WHERE ${workdayTableName}_id IN (${sqlValueString})
  `);
  connection.release();

  return res.json({
    message: "Delete workday success",
    status: 200,
  });
 }
```

Sistem akan menghapus data yang terdapat pada *database* sesuai dengan ID yang dikirimkan pengguna dari *website*. *Source code* dapat dilihat pada Segmen Program 4.29.

4.2.4. Menu *Master Data* Karyawan

Segmen Program 4.30. *Get Identity Card*

```
async (req, res) => {
  // update path
  const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
  identityCardPath = `public/uploads/${currentUser}/${identityCardTableName}`;

  // check if template exist
  if (!isFoundFile(identityCardPath, identityCardTemplateFilename)) {
    return res.json({ status: 204 });
  }

  let filterModel = req.body.filterModel;
  let pageIndex = req.body.pageIndex;
  let rowsPerPage = req.body.rowsPerPage;
  let sortModel = req.body.sortModel;

  // get template column data
```

```
  const templateData = await getExcelTemplateData(
    identityCardPath + "/" + identityCardTemplateFilename,
    worksheetTemplateColumnTitleIndex,
    worksheetTemplateRowRuleIndex,
    worksheetTemplateRowTitleIndex
  );

  // add ID column
  templateData.columns.unshift("ID");
  templateData.fields.unshift("identity_card_id");
  templateData.rules.unshift("id");

  let filterModelString = convertFilterModelToSQL(filterModel);
  let sortModelString = convertSortModelToSQL(sortModel);

  // fetch data
  const connection = await mysql.connection();
  await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);

  let query = await connection.query(`
    SELECT *
    FROM ${identityCardTableName}
    ${filterModelString}
    ${sortModelString}
    LIMIT ${rowsPerPage}
    OFFSET ${pageIndex * rowsPerPage}
  `);

  // get filtered row count
  let rowCount = await connection.query(`
    SELECT COUNT(*) AS row_count
    FROM ${identityCardTableName}
    ${filterModelString}
    ${sortModelString}
  `);

  connection.release();

  return res.json({
    payload: { query: query, rowCount: rowCount[0].row_count },
    status: 200,
    templateData: templateData,
  });
}
```

Sistem akan mengecek apakah *template* ada. Jika ada, sistem akan mengambil data kolom, menambahkan kolom ID, mengambil data tersebut untuk diberikan ke pengguna, lalu mengirim data tersebut ke *website*. *Source code* dapat dilihat pada Segmen Program 4.30.

Segmen Program 4.31. *Import Template Identity Card*

```
async (req, res) => {
  if (req.fileValidationError) {
    return res.json({
      message: req.fileValidationError,
      status: 500,
    });
  }

  // get uploaded template filename
  const filename = req.file.originalname;

  // check is template valid
  const isInvalidTemplate = await isInvalidExcelTemplateData(
    modules.IDENTITY_CARD.key,
    tempPath + "/" + filename,
    worksheetTemplateColumnTitleIndex,
    worksheetTemplateRowRuleIndex,
    worksheetTemplateRowTitleIndex
  );

  if (isInvalidTemplate) {
    // delete uploaded template at temp folder
    fs.unlinkSync(tempPath + "/" + filename);

    return res.json({
      message: isInvalidTemplate,
      status: 500,
    });
  }

  // if template valid
  // get template column data
  const templateData = await getExcelTemplateData(
    tempPath + "/" + filename,
    worksheetTemplateColumnTitleIndex,
    worksheetTemplateRowRuleIndex,
    worksheetTemplateRowTitleIndex
  );

  // update path
  const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
  identityCardPath = `public/uploads/${currentUser}/${identityCardTableName}`;

  // create if directory not exist
  const dir = identityCardPath;

  if (!fs.existsSync(dir)) {
    fs.mkdirSync(dir, { recursive: true });
```

```
  }

  // copy template to correct path
  fs.copyFileSync(
    tempPath + "/" + filename,
    identityCardPath + "/" + identityCardTemplateFilename
  );

  // delete uploaded template at temp folder
  fs.unlinkSync(tempPath + "/" + filename);

  // delete file data
  if (fs.existsSync(identityCardPath + "/" + identityCardFilename)) {
    fs.unlinkSync(identityCardPath + "/" + identityCardFilename);
  }

  // generate SQL syntax
  let sqlString = "";

  for (let i = 0; i < templateData.columns.length; i++) {
    const field = templateData.fields[i];
    const dataType = convertRuleToSQL(templateData.rules[i]);

    sqlString += `${field} ${dataType},\n`;
  }

  const connection = await mysql.connection();
  await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);

  // drop table if exist
  await connection.query(`DROP TABLE IF EXISTS ${identityCardTableName}`);

  // create table
  await connection.query(
    `CREATE TABLE IF NOT EXISTS ${identityCardTableName} (
      ${identityCardTableName}_id int NOT NULL AUTO_INCREMENT,
      ${sqlString}
      PRIMARY KEY (${identityCardTableName}_id)
    )`
  );

  connection.release();

  return res.json({
    message: "Import template success",
    status: 200,
  });
}
```

Sistem akan mengecek apakah format *template* valid. Jika format *template* tersebut valid, sistem akan mengambil data-data kolom tersebut untuk ditambahkan dalam bentuk SQL. *Source code* dapat dilihat pada Segmen Program 4.31.

Segmen Program 4.32. *Import Data Identity Card*

```
                                async (req, res) => {
  if (req.fileValidationError) {
    return res.json({
      message: req.fileValidationError,
      status: 500,
    });
  }

  // update path
  const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
  identityCardPath = `public/uploads/${currentUser}/${identityCardTableName}`;

  // get uploaded data filename
  const filename = req.file.originalname;

  // check if template not exist
  if (!isFoundFile(identityCardPath, identityCardTemplateFilename)) {
    // delete uploaded file at temp folder
    fs.unlinkSync(tempPath + "/" + filename);

    return res.json({ message: "Template not found", status: 204 });
  }

  // get template column data
  const templateData = await getExcelTemplateData(
    identityCardPath + "/" + identityCardTemplateFilename,
    worksheetTemplateColumnTitleIndex,
    worksheetTemplateRowRuleIndex,
    worksheetTemplateRowTitleIndex
  );

  // check is data valid
  const isInvalidData = await isInvalidExcelData(
    templateData,
    tempPath + "/" + filename,
    worksheetDataColumnTitleIndex,
    worksheetDataRowDataIndex,
    worksheetDataRowTitleIndex
  );

  if (isInvalidData) {
    // delete uploaded template at temp folder
```

```
  fs.unlinkSync(tempPath + "/" + filename);

  return res.json({
    message: isInvalidData,
    status: 500,
  });
}

// if file valid
// get data
const data = await getExcelData(
  tempPath + "/" + filename,
  worksheetDataRowDataIndex
);

// create if directory not exist
const dir = identityCardPath;

if (!fs.existsSync(dir)) {
  fs.mkdirSync(dir, { recursive: true });
}

// copy file to correct path
fs.copyFileSync(
  tempPath + "/" + filename,
  identityCardPath + "/" + identityCardFilename
);

// delete uploaded file at temp folder
fs.unlinkSync(tempPath + "/" + filename);

const connection = await mysql.connection();
await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);
await connection.query("START TRANSACTION");
// if replace data
if (req.query.isReplace === "true") {
  // empty table
  await connection.query(`TRUNCATE TABLE ${identityCardTableName}`);
}

let sqlFieldString = "";
// init field string
for (let i = 0; i < templateData.columns.length; i++) {
  const field = templateData.fields[i];
  sqlFieldString += `${field}, `;
}

// remove comma
sqlFieldString =
```

```
    `${identityCardTableName}_id, ` + sqlFieldString.slice(0, -2);

  let sqlValueString = "";

  // init value string
  for (let i = 0; i < data.length; i++) {
    const row = data[i];
    sqlValueString = "";

    for (let j = 0; j < row.length; j++) {
      const cellValue = row[j];
      const dataType = convertRuleToSQL(templateData.rules[j]);

      // if cell value not null
      if (cellValue) {
        if (dataType === "DATE") {
          sqlValueString += `STR_TO_DATE('${dateFns.format(
            excelDateToJSDate(cellValue),
            "dd-MM-yyyy"
          )}', '%d-%m-%Y'), `;
        } else if (dataType === "DOUBLE") {
          sqlValueString += `${cellValue}, `;
        } else {
          sqlValueString += `'${cellValue}', `;
        }
      } else {
        if (dataType === "DATE") {
          sqlValueString += `NULL, `;
        } else if (dataType === "DOUBLE") {
          sqlValueString += `0, `;
        } else {
          sqlValueString += `'', `;
        }
      }
    }

    // remove comma
    sqlValueString = `null, ` + sqlValueString.slice(0, -2);
    // transaction
    try {
      await connection.query(
        `INSERT INTO ${identityCardTableName} (${sqlFieldString})
        VALUES (${sqlValueString})`
      );
    } catch (error) {
      await connection.query("ROLLBACK");
      return res.json({
        message: "Import data failed",
        status: 400,
```

```
      });
    }
  }
  connection.query("COMMIT");
  connection.release();

  return res.json({
    message: "Import data success",
    status: 200,
  });
}
```

Sistem akan mengecek apakah *template* ada. Jika ada, sistem akan mengambil data kolom, lalu sistem akan mengecek apakah data yang diberikan valid. Jika valid, kosongkan tabel sesuai data untuk diisi ulang, setelah itu sistem akan mengisi *database* sesuai data yang diberikan. *Source code* dapat dilihat pada Segmen Program 4.32.

Segmen Program 4.33. *Export Template Identity Card*

```
async (req, res) => {
  // update path
  const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
  identityCardPath = `public/uploads/${currentUser}/${identityCardTableName}`;

  if (!isFoundFile(identityCardPath, identityCardTemplateFilename)) {
    return res.json({ message: "Template not found", status: 204 });
  }

  return res.download(
    identityCardPath + "/" + identityCardTemplateFilename,
    identityCardTemplateFilename,
    (err) => {
      if (err) {
        res.json({ message: "Could not download the file", status: 500 });
      }
    }
  );
}
```

Sistem akan mengecek apakah data *template* kartu identitas ada. Jika ada, sistem akan mengirim data *template* kartu identitas. *Source code* dapat dilihat pada Segmen Program 4.33

Segmen Program 4.34. *Export Data Identity Card*

```
async (req, res) => {
  // update path
  const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
```

```
identityCardPath = `public/uploads/${currentUser}/${identityCardTableName}`;

let filterModel = req.body.filterModel;
let columnVisibilityModel = req.body.columnVisibilityModel;
let sortModel = req.body.sortModel;

// get template column data
const templateData = await getExcelTemplateData(
  identityCardPath + "/" + identityCardTemplateFilename,
  worksheetTemplateColumnTitleIndex,
  worksheetTemplateRowRuleIndex,
  worksheetTemplateRowTitleIndex
);

let columnVisibilityModelString = convertColumnVisibilityModelToSQL(
  columnVisibilityModel,
  templateData
);
let filterModelString = convertFilterModelToSQL(filterModel);
let sortModelString = convertSortModelToSQL(sortModel);
let templateDataAccordingRule =
  getExcelTemplateDataAccordingColumnVisibilityModel(
    columnVisibilityModel,
    templateData
  );

const connection = await mysql.connection();
await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);

let query = await connection.query(`
  SELECT ${columnVisibilityModelString}
  FROM ${identityCardTableName}
  ${filterModelString}
  ${sortModelString}
`);

connection.release();

// if file exist
if (isFoundFile(identityCardPath, identityCardFilename)) {
  // delete uploaded file
  fs.unlinkSync(identityCardPath + "/" + identityCardFilename);
}

// export file
await exportExcelData(
  query,
  identityCardPath + "/" + identityCardFilename,
  templateDataAccordingRule,
```

```
    worksheetDataRowDataIndex
  );

  return res.download(
    identityCardPath + "/" + identityCardFilename,
    identityCardFilename,
    (err) => {
      if (err) {
        res.json({ message: "Could not download the file", status: 500 });
      }
    }
  );
}
```

Sistem akan mengambil data *template* kartu identitas, lalu merangkai data sesuai *filter* yang digunakan, dan mengurutkan sesuai *sort* yang dipilih, setelah itu sistem akan mengirim data kartu identitas ke pengguna. *Source code* dapat dilihat pada Segmen Program 4.34.

Segmen Program 4.35. *Create Identity Card*

```
async (req, res) => {
  // update path
  const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
  identityCardPath = `public/uploads/${currentUser}/${identityCardTableName}`;

  let newIdentityCardData = req.body.newIdentityCardData;

  // get template column data
  const templateData = await getExcelTemplateData(
    identityCardPath + "/" + identityCardTemplateFilename,
    worksheetTemplateColumnTitleIndex,
    worksheetTemplateRowRuleIndex,
    worksheetTemplateRowTitleIndex
  );

  const isInvalidData = await isInvalidInsertData(
    templateData,
    newIdentityCardData
  );

  if (isInvalidData) {
    return res.json({
      message: isInvalidData,
      status: 500,
    });
  }

  // generate SQL syntax
```

```
let newIdentityCardDataValues = Object.values(newIdentityCardData);

// remove id field
newIdentityCardDataValues.shift();

let sqlFieldString = "";

// init field string
for (let i = 0; i < templateData.columns.length; i++) {
 const field = templateData.fields[i];
 sqlFieldString += `${field}, `;
}

// remove comma
sqlFieldString =
 `${identityCardTableName}_id, ` + sqlFieldString.slice(0, -2);

let sqlValueString = "";

// init value string
for (let i = 0; i < newIdentityCardDataValues.length; i++) {
 const newIdentityCardDataValue = newIdentityCardDataValues[i];
 const dataType = convertRuleToSQL(templateData.rules[i]);

 if (newIdentityCardDataValue) {
  if (dataType === "DATE") {
   sqlValueString += `STR_TO_DATE('${dateFns.format(
    dateFns.parseISO(newIdentityCardDataValue),
    "dd-MM-yyyy"
   )}', '%d-%m-%Y'), `;
  } else if (dataType === "DOUBLE") {
   sqlValueString += `${newIdentityCardDataValue}, `;
  } else {
   sqlValueString += `'${newIdentityCardDataValue}', `;
  }
 } else {
  if (dataType === "DATE") {
   sqlValueString += `NULL, `;
  } else if (dataType === "DOUBLE") {
   sqlValueString += `0, `;
  } else {
   sqlValueString += `'', `;
  }
 }
}

// remove comma
sqlValueString = `null, ` + sqlValueString.slice(0, -2);
```

```
  const connection = await mysql.connection();
  await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);
  await connection.query(`
    INSERT INTO ${identityCardTableName} (${sqlFieldString})
    VALUES (
    ${sqlValueString}
    )
  `);
  connection.release();

  return res.json({
    message: "Create identity card success",
    status: 200,
  });
 }
```

Sistem akan mengambil data kolom pada *template*, lalu sistem akan mengecek apakah data yang baru adalah valid. Jika data tersebut valid, sistem akan memasukkan data kartu identitas pada *database*. *Source code* dapat dilihat pada Segmen Program 4.35.

Segmen Program 4.36. *Update Identity Card*

```
async (req, res) => {
  // update path
  const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
  identityCardPath = `public/uploads/${currentUser}/${identityCardTableName}`;

  let newIdentityCardData = req.body.newIdentityCardData;

  // get template column data
  const templateData = await getExcelTemplateData(
    identityCardPath + "/" + identityCardTemplateFilename,
    worksheetTemplateColumnTitleIndex,
    worksheetTemplateRowRuleIndex,
    worksheetTemplateRowTitleIndex
  );

  const isInvalidData = await isInvalidUpdateData(
    templateData,
    newIdentityCardData
  );

  if (isInvalidData) {
    return res.json({
      message: isInvalidData,
      status: 500,
    });
  }
```

```
  // generate SQL syntax
  let newIdentityCardDataValues = Object.values(newIdentityCardData);

  // remove id field
  newIdentityCardDataValues.shift();

  // generate SQL syntax
  let sqlString = "";

  for (let i = 0; i < newIdentityCardDataValues.length; i++) {
    const newIdentityCardDataValue = newIdentityCardDataValues[i];
    const dataType = convertRuleToSQL(templateData.rules[i]);

    sqlString += `${templateData.fields[i]} = `;

    if (dataType === "DATE") {
      if (newIdentityCardDataValue) {
        sqlString += `STR_TO_DATE('${dateFns.format(
          dateFns.parseISO(newIdentityCardDataValue),
          "dd-MM-yyyy"
        )}', '%d-%m-%Y'), `;
      } else {
        sqlString += `NULL, `;
      }
    } else if (dataType === "DOUBLE") {
      sqlString += `${newIdentityCardDataValue}, `;
    } else {
      sqlString += `'${newIdentityCardDataValue}', `;
    }
  }

  // remove comma
  sqlString = sqlString.slice(0, -2);

  const connection = await mysql.connection();
  await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);
  await connection.query(`
    UPDATE ${identityCardTableName}
    SET ${sqlString}
    WHERE ${identityCardTableName}_id = ${newIdentityCardData.identity_card_id}
  `);
  connection.release();

  return res.json({
    message: "Update identity card success",
    status: 200,
  });
}
```

Sistem akan mengambil data kolom pada *template*, lalu sistem akan mengecek apakah data yang baru adalah valid. Jika data tersebut valid, sistem akan memperbarui data pada *database*. *Source code* dapat dilihat pada Segmen Program 4.36.

Segmen Program 4.37. *Delete Identity Card*

```
async (req, res) => {
  // update path
  const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
  identityCardPath = `public/uploads/${currentUser}/${identityCardTableName}`;

  let selectedIdentityCardID = req.body.selectedIdentityCardID;
  let sqlValueString = "";

  for (let i = 0; i < selectedIdentityCardID.length; i++) {
    sqlValueString += `${selectedIdentityCardID[i]}, `;
  }

  // remove comma
  sqlValueString = sqlValueString.slice(0, -2);

  const connection = await mysql.connection();
  await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);
  await connection.query(`
    DELETE FROM ${identityCardTableName}
    WHERE ${identityCardTableName}_id IN (${sqlValueString})
  `);
  connection.release();

  return res.json({
    message: "Delete identity card success",
    status: 200,
  });
 }
```

Sistem akan menghapus data yang terdapat pada *database* sesuai dengan ID yang dikirimkan pengguna dari *website*. *Source code* dapat dilihat pada Segmen Program 4.37.

Segmen Program 4.38. *Get Work Agreement*

```
async (req, res) => {
  // update path
  const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
  workAgreementPath = `public/uploads/${currentUser}/${workAgreementTableName}`;

  // check if template exist
```

```
if (!isFoundFile(workAgreementPath, workAgreementTemplateFilename)) {
  return res.json({ status: 204 });
}

let filterModel = req.body.filterModel;
let pageIndex = req.body.pageIndex;
let rowsPerPage = req.body.rowsPerPage;
let sortModel = req.body.sortModel;

// get template column data
const templateData = await getExcelTemplateData(
  workAgreementPath + "/" + workAgreementTemplateFilename,
  worksheetTemplateColumnTitleIndex,
  worksheetTemplateRowRuleIndex,
  worksheetTemplateRowTitleIndex
);

// add ID column
templateData.columns.unshift("ID");
templateData.fields.unshift("work_agreement_id");
templateData.rules.unshift("id");

let filterModelString = convertFilterModelToSQL(filterModel);
let sortModelString = convertSortModelToSQL(sortModel);

// fetch data
const connection = await mysql.connection();
await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);

let query = await connection.query(`
  SELECT *
  FROM ${workAgreementTableName}
  ${filterModelString}
  ${sortModelString}
  LIMIT ${rowsPerPage}
  OFFSET ${pageIndex * rowsPerPage}
`);

// get filtered row count
let rowCount = await connection.query(`
  SELECT COUNT(*) AS row_count
  FROM ${workAgreementTableName}
  ${filterModelString}
  ${sortModelString}
`);

connection.release();

return res.json({
```

```
    payload: { query: query, rowCount: rowCount[0].row_count },
    status: 200,
    templateData: templateData,
   });
  }
```

Sistem akan mengecek apakah *template* ada. Jika ada, sistem akan mengambil data kolom, menambahkan kolom ID, mengambil data tersebut untuk diberikan ke pengguna, lalu mengirim data tersebut ke *website*. *Source code* dapat dilihat pada Segmen Program 4.38.

Segmen Program 4.39. *Import Template Work Agreement*

```
async (req, res) => {
   if (req.fileValidationError) {
    return res.json({
     message: req.fileValidationError,
     status: 500,
    });
   }

   // get uploaded template filename
   const filename = req.file.originalname;

   // check is template valid
   const isInvalidTemplate = await isInvalidExcelTemplateData(
    modules.WORK_AGREEMENT.key,
    tempPath + "/" + filename,
    worksheetTemplateColumnTitleIndex,
    worksheetTemplateRowRuleIndex,
    worksheetTemplateRowTitleIndex
   );

   if (isInvalidTemplate) {
    // delete uploaded template at temp folder
    fs.unlinkSync(tempPath + "/" + filename);

    return res.json({
     message: isInvalidTemplate,
     status: 500,
    });
   }

   // if template valid
   // get template column data
   const templateData = await getExcelTemplateData(
    tempPath + "/" + filename,
    worksheetTemplateColumnTitleIndex,
    worksheetTemplateRowRuleIndex,
```

```
  worksheetTemplateRowTitleIndex
);

// update path
const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
workAgreementPath = `public/uploads/${currentUser}/${workAgreementTableName}`;

// create if directory not exist
const dir = workAgreementPath;

if (!fs.existsSync(dir)) {
  fs.mkdirSync(dir, { recursive: true });
}

// copy template to correct path
fs.copyFileSync(
  tempPath + "/" + filename,
  workAgreementPath + "/" + workAgreementTemplateFilename
);

// delete uploaded template at temp folder
fs.unlinkSync(tempPath + "/" + filename);

// delete file data
if (fs.existsSync(workAgreementPath + "/" + workAgreementFilename)) {
  fs.unlinkSync(workAgreementPath + "/" + workAgreementFilename);
}

// generate SQL syntax
let sqlString = "";

for (let i = 0; i < templateData.columns.length; i++) {
  const field = templateData.fields[i];
  const dataType = convertRuleToSQL(templateData.rules[i]);

  sqlString += `${field} ${dataType},\n`;
}

const connection = await mysql.connection();
await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);

// drop table if exist
await connection.query(`DROP TABLE IF EXISTS ${workAgreementTableName}`);

// create table
await connection.query(
  `CREATE TABLE IF NOT EXISTS ${workAgreementTableName} (
    ${workAgreementTableName}_id int NOT NULL AUTO_INCREMENT,
    ${sqlString}
```

```
      PRIMARY KEY (${workAgreementTableName}_id)
    )`
  );

  connection.release();

  return res.json({
    message: "Import template success",
    status: 200,
  });
 }
```

Sistem akan mengecek apakah format *template* valid. Jika format *template* tersebut valid, sistem akan mengambil data-data kolom tersebut untuk ditambahkan dalam bentuk SQL. *Source code* dapat dilihat pada Segmen Program 4.39.

Segmen Program 4.40. *Import Data Work Agreement*

```
async (req, res) => {
  if (req.fileValidationError) {
    return res.json({
      message: req.fileValidationError,
      status: 500,
    });
  }

  // update path
  const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
  workAgreementPath = `public/uploads/${currentUser}/${workAgreementTableName}`;

  // get uploaded data filename
  const filename = req.file.originalname;

  // check if template not exist
  if (!isFoundFile(workAgreementPath, workAgreementTemplateFilename)) {
    // delete uploaded file at temp folder
    fs.unlinkSync(tempPath + "/" + filename);

    return res.json({ message: "Template not found", status: 204 });
  }

  // get template column data
  const templateData = await getExcelTemplateData(
    workAgreementPath + "/" + workAgreementTemplateFilename,
    worksheetTemplateColumnTitleIndex,
    worksheetTemplateRowRuleIndex,
    worksheetTemplateRowTitleIndex
  );
```

```
// check is data valid
const isInvalidData = await isInvalidExcelData(
  templateData,
  tempPath + "/" + filename,
  worksheetDataColumnTitleIndex,
  worksheetDataRowDataIndex,
  worksheetDataRowTitleIndex
);

if (isInvalidData) {
  // delete uploaded template at temp folder
  fs.unlinkSync(tempPath + "/" + filename);

  return res.json({
    message: isInvalidData,
    status: 500,
  });
}

// if file valid
// get data
const data = await getExcelData(
  tempPath + "/" + filename,
  worksheetDataRowDataIndex
);

// create if directory not exist
const dir = workAgreementPath;

if (!fs.existsSync(dir)) {
  fs.mkdirSync(dir, { recursive: true });
}

// copy file to correct path
fs.copyFileSync(
  tempPath + "/" + filename,
  workAgreementPath + "/" + workAgreementFilename
);

// delete uploaded file at temp folder
fs.unlinkSync(tempPath + "/" + filename);

const connection = await mysql.connection();
await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);
await connection.query("START TRANSACTION");
// if replace data
if (req.query.isReplace === "true") {
  // empty table
```

```
  await connection.query(`TRUNCATE TABLE ${workAgreementTableName}`);
}

let sqlFieldString = "";
// init field string
for (let i = 0; i < templateData.columns.length; i++) {
  const field = templateData.fields[i];
  sqlFieldString += `${field}, `;
}

// remove comma
sqlFieldString =
  `${workAgreementTableName}_id, ` + sqlFieldString.slice(0, -2);

let sqlValueString = "";

// init value string
for (let i = 0; i < data.length; i++) {
  const row = data[i];
  sqlValueString = "";

  for (let j = 0; j < row.length; j++) {
    const cellValue = row[j];
    const dataType = convertRuleToSQL(templateData.rules[j]);

    // if cell value not null
    if (cellValue) {
      if (dataType === "DATE") {
        sqlValueString += `STR_TO_DATE('${dateFns.format(
          excelDateToJSDate(cellValue),
          "dd-MM-yyyy"
        )}', '%d-%m-%Y'), `;
      } else if (dataType === "DOUBLE") {
        sqlValueString += `${cellValue}, `;
      } else {
        sqlValueString += `'${cellValue}', `;
      }
    } else {
      if (dataType === "DATE") {
        sqlValueString += `NULL, `;
      } else if (dataType === "DOUBLE") {
        sqlValueString += `0, `;
      } else {
        sqlValueString += `'', `;
      }
    }
  }

  // remove comma
```

```
    sqlValueString = `null, ` + sqlValueString.slice(0, -2);
    // transaction
    try {
      await connection.query(
        `INSERT INTO ${workAgreementTableName} (${sqlFieldString})
        VALUES (${sqlValueString})`
      );
    } catch (error) {
      await connection.query("ROLLBACK");
      return res.json({
        message: "Import data failed",
        status: 400,
      });
    }
  }

  connection.query("COMMIT");
  connection.release();

  return res.json({
    message: "Import data success",
    status: 200,
  });
 }
```

Sistem akan mengecek apakah *template* ada. Jika ada, sistem akan mengambil data kolom, lalu sistem akan mengecek apakah data yang diberikan valid. Jika valid, kosongkan tabel sesuai data untuk diisi ulang, setelah itu sistem akan mengisi *database* sesuai data yang diberikan. *Source code* dapat dilihat pada Segmen Program 4.40.

Segmen Program 4.41. *Export Template Work Agreement*

```
async (req, res) => {
  // update path
  const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
  workAgreementPath = `public/uploads/${currentUser}/${workAgreementTableName}`;

  if (!isFoundFile(workAgreementPath, workAgreementTemplateFilename)) {
    return res.json({ message: "Template not found", status: 204 });
  }

  return res.download(
    workAgreementPath + "/" + workAgreementTemplateFilename,
    workAgreementTemplateFilename,
    (err) => {
      if (err) {
        res.json({ message: "Could not download the file", status: 500 });
      }
```

```
    }
  );
 }
```

Sistem akan mengecek apakah data *template* kesepakatan kerja ada. Jika ada, sistem akan mengirim data *template* kesepakatan kerja. *Source code* dapat dilihat pada Segmen Program 4.41.

Segmen Program 4.42. *Export Data Work Agreement*

```
async (req, res) => {
  // update path
  const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
  workAgreementPath = `public/uploads/${currentUser}/${workAgreementTableName}`;

  let filterModel = req.body.filterModel;
  let columnVisibilityModel = req.body.columnVisibilityModel;
  let sortModel = req.body.sortModel;

  // get template column data
  const templateData = await getExcelTemplateData(
    workAgreementPath + "/" + workAgreementTemplateFilename,
    worksheetTemplateColumnTitleIndex,
    worksheetTemplateRowRuleIndex,
    worksheetTemplateRowTitleIndex
  );

  let columnVisibilityModelString = convertColumnVisibilityModelToSQL(
    columnVisibilityModel,
    templateData
  );
  let filterModelString = convertFilterModelToSQL(filterModel);
  let sortModelString = convertSortModelToSQL(sortModel);
  let templateDataAccordingRule =
    getExcelTemplateDataAccordingColumnVisibilityModel(
      columnVisibilityModel,
      templateData
    );

  const connection = await mysql.connection();
  await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);

  let query = await connection.query(`
    SELECT ${columnVisibilityModelString}
    FROM ${workAgreementTableName}
    ${filterModelString}
    ${sortModelString}
  `);
```

```
  connection.release();

  // if file exist
  if (isFoundFile(workAgreementPath, workAgreementFilename)) {
    // delete uploaded file
    fs.unlinkSync(workAgreementPath + "/" + workAgreementFilename);
  }

  // export file
  await exportExcelData(
    query,
    workAgreementPath + "/" + workAgreementFilename,
    templateDataAccordingRule,
    worksheetDataRowDataIndex
  );

  return res.download(
    workAgreementPath + "/" + workAgreementFilename,
    workAgreementFilename,
    (err) => {
      if (err) {
        res.json({ message: "Could not download the file", status: 500 });
      }
    }
  );
}
```

Sistem akan mengambil data *template* kesepakatan kerja, lalu merangkai data sesuai *filter* yang digunakan, dan mengurutkan sesuai *sort* yang dipilih, setelah itu sistem akan mengirim data kesepakatan kerja ke pengguna. *Source code* dapat dilihat pada Segmen Program 4.42

Segmen Program 4.43. *Create Work Agreement*

```
async (req, res) => {
  // update path
  const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
  workAgreementPath = `public/uploads/${currentUser}/${workAgreementTableName}`;

  let newWorkAgreementData = req.body.newWorkAgreementData;

  // get template column data
  const templateData = await getExcelTemplateData(
    workAgreementPath + "/" + workAgreementTemplateFilename,
    worksheetTemplateColumnTitleIndex,
    worksheetTemplateRowRuleIndex,
    worksheetTemplateRowTitleIndex
```

```
);

const isInvalidData = await isInvalidInsertData(
  templateData,
  newWorkAgreementData
);

if (isInvalidData) {
  return res.json({
    message: isInvalidData,
    status: 500,
  });
}

// generate SQL syntax
let newWorkAgreementDataValues = Object.values(newWorkAgreementData);

// remove id field
newWorkAgreementDataValues.shift();

let sqlFieldString = "";

// init field string
for (let i = 0; i < templateData.columns.length; i++) {
  const field = templateData.fields[i];
  sqlFieldString += `${field}, `;
}

// remove comma
sqlFieldString =
  `${workAgreementTableName}_id, ` + sqlFieldString.slice(0, -2);

let sqlValueString = "";

// init value string
for (let i = 0; i < newWorkAgreementDataValues.length; i++) {
  const newWorkAgreementDataValue = newWorkAgreementDataValues[i];
  const dataType = convertRuleToSQL(templateData.rules[i]);

  if (newWorkAgreementDataValue) {
    if (dataType === "DATE") {
      sqlValueString += `STR_TO_DATE('${dateFns.format(
        dateFns.parseISO(newWorkAgreementDataValue),
        "dd-MM-yyyy"
      )}', '%d-%m-%Y'), `;
    } else if (dataType === "DOUBLE") {
      sqlValueString += `${newWorkAgreementDataValue}, `;
    } else {
      sqlValueString += `'${newWorkAgreementDataValue}', `;
```

```
      }
    } else {
      if (dataType === "DATE") {
        sqlValueString += `NULL, `;
      } else if (dataType === "DOUBLE") {
        sqlValueString += `0, `;
      } else {
        sqlValueString += `'', `;
      }
    }
  }

  // remove comma
  sqlValueString = `null, ` + sqlValueString.slice(0, -2);

  const connection = await mysql.connection();
  await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);
  await connection.query(`
    INSERT INTO ${workAgreementTableName} (${sqlFieldString})
    VALUES (
    ${sqlValueString}
    )
  `);
  connection.release();

  return res.json({
    message: "Create work agreement success",
    status: 200,
  });
 }
```

Sistem akan mengambil data kolom pada *template*, lalu sistem akan mengecek apakah data yang baru adalah valid. Jika data tersebut valid, sistem akan memasukkan data kesepakatan kerja pada *database*. *Source code* dapat dilihat pada Segmen Program 4.43.

Segmen Program 4.44. *Update Work Agreement*

```
async (req, res) => {
  // update path
  const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
  workAgreementPath = `public/uploads/${currentUser}/${workAgreementTableName}`;

  let newWorkAgreementData = req.body.newWorkAgreementData;

  // get template column data
  const templateData = await getExcelTemplateData(
    workAgreementPath + "/" + workAgreementTemplateFilename,
    worksheetTemplateColumnTitleIndex,
```

```
  worksheetTemplateRowRuleIndex,
  worksheetTemplateRowTitleIndex
);

const isInvalidData = await isInvalidUpdateData(
  templateData,
  newWorkAgreementData
);

if (isInvalidData) {
  return res.json({
    message: isInvalidData,
    status: 500,
  });
}

// generate SQL syntax
let newWorkAgreementDataValues = Object.values(newWorkAgreementData);

// remove id field
newWorkAgreementDataValues.shift();

// generate SQL syntax
let sqlString = "";

for (let i = 0; i < newWorkAgreementDataValues.length; i++) {
  const newWorkAgreementDataValue = newWorkAgreementDataValues[i];
  const dataType = convertRuleToSQL(templateData.rules[i]);

  sqlString += `${templateData.fields[i]} = `;

  if (dataType === "DATE") {
    if (newWorkAgreementDataValue) {
      sqlString += `STR_TO_DATE('${dateFns.format(
        dateFns.parseISO(newWorkAgreementDataValue),
        "dd-MM-yyyy"
      )}', '%d-%m-%Y'), `;
    } else {
      sqlString += `NULL, `;
    }
  } else if (dataType === "DOUBLE") {
    sqlString += `${newWorkAgreementDataValue}, `;
  } else {
    sqlString += `'${newWorkAgreementDataValue}', `;
  }
}

// remove comma
sqlString = sqlString.slice(0, -2);
```

```
  const connection = await mysql.connection();
  await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);
  await connection.query(`
    UPDATE ${workAgreementTableName}
    SET ${sqlString}
    WHERE ${workAgreementTableName}_id =
${newWorkAgreementData.work_agreement_id}
  `);
  connection.release();

  return res.json({
    message: "Update work agreement success",
    status: 200,
  });
 }
```

Sistem akan mengambil data kolom pada *template*, lalu sistem akan mengecek apakah data yang baru adalah valid. Jika data tersebut valid, sistem akan memperbarui data pada *database*. *Source code* dapat dilihat pada Segmen Program 4.44.

Segmen Program 4.45. *Delete Work Agreement*

```
async (req, res) => {
  // update path
  const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
  workAgreementPath = `public/uploads/${currentUser}/${workAgreementTableName}`;

  let selectedWorkAgreementID = req.body.selectedWorkAgreementID;
  let sqlValueString = "";

  for (let i = 0; i < selectedWorkAgreementID.length; i++) {
    sqlValueString += `${selectedWorkAgreementID[i]}, `;
  }

  // remove comma
  sqlValueString = sqlValueString.slice(0, -2);

  const connection = await mysql.connection();
  await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);
  await connection.query(`
    DELETE FROM ${workAgreementTableName}
    WHERE ${workAgreementTableName}_id IN (${sqlValueString})
  `);
  connection.release();

  return res.json({
    message: "Delete work agreement success",
```

```
    status: 200,
   });
  }
```

Sistem akan menghapus data yang terdapat pada *database* sesuai dengan ID yang dikirimkan pengguna dari *website*. *Source code* dapat dilihat pada Segmen Program 4.45

Segmen Program 4.46. *Get Wage Agreement*

```
async (req, res) => {
   // update path
   const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
   wageAgreementPath = `public/uploads/${currentUser}/${wageAgreementTableName}`;

   // check if template exist
   if (!isFoundFile(wageAgreementPath, wageAgreementTemplateFilename)) {
    return res.json({ status: 204 });
   }

   let filterModel = req.body.filterModel;
   let pageIndex = req.body.pageIndex;
   let rowsPerPage = req.body.rowsPerPage;
   let sortModel = req.body.sortModel;

   // get template column data
   const templateData = await getExcelTemplateData(
    wageAgreementPath + "/" + wageAgreementTemplateFilename,
    worksheetTemplateColumnTitleIndex,
    worksheetTemplateRowRuleIndex,
    worksheetTemplateRowTitleIndex
   );

   // add ID column
   templateData.columns.unshift("ID");
   templateData.fields.unshift("wage_agreement_id");
   templateData.rules.unshift("id");

   let filterModelString = convertFilterModelToSQL(filterModel);
   let sortModelString = convertSortModelToSQL(sortModel);

   // fetch data
   const connection = await mysql.connection();
   await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);

   let query = await connection.query(`
    SELECT *
    FROM ${wageAgreementTableName}
    ${filterModelString}
    ${sortModelString}
```

```
    LIMIT ${rowsPerPage}
    OFFSET ${pageIndex * rowsPerPage}
  `);

  // get filtered row count
  let rowCount = await connection.query(`
    SELECT COUNT(*) AS row_count
    FROM ${wageAgreementTableName}
    ${filterModelString}
    ${sortModelString}
  `);

  connection.release();

  return res.json({
    payload: { query: query, rowCount: rowCount[0].row_count },
    status: 200,
    templateData: templateData,
  });
 }
```

Sistem akan mengecek apakah *template* ada. Jika ada, sistem akan mengambil data kolom, menambahkan kolom ID, mengambil data tersebut untuk diberikan ke pengguna, lalu mengirim data tersebut ke *website*. *Source code* dapat dilihat pada Segmen Program 4.46.

Segmen Program 4.47. *Import Template Wage Agreement*

```
async (req, res) => {
  if (req.fileValidationError) {
    return res.json({
      message: req.fileValidationError,
      status: 500,
    });
  }

  // get uploaded template filename
  const filename = req.file.originalname;

  // check is template valid
  const isInvalidTemplate = await isInvalidExcelTemplateData(
    modules.WAGE_AGREEMENT.key,
    tempPath + "/" + filename,
    worksheetTemplateColumnTitleIndex,
    worksheetTemplateRowRuleIndex,
    worksheetTemplateRowTitleIndex
  );

  if (isInvalidTemplate) {
```

```
  // delete uploaded template at temp folder
  fs.unlinkSync(tempPath + "/" + filename);

  return res.json({
    message: isInvalidTemplate,
    status: 500,
  });
}

// if template valid
// get template column data
const templateData = await getExcelTemplateData(
  tempPath + "/" + filename,
  worksheetTemplateColumnTitleIndex,
  worksheetTemplateRowRuleIndex,
  worksheetTemplateRowTitleIndex
);

// update path
const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
wageAgreementPath = `public/uploads/${currentUser}/${wageAgreementTableName}`;

// create if directory not exist
const dir = wageAgreementPath;

if (!fs.existsSync(dir)) {
  fs.mkdirSync(dir, { recursive: true });
}

// copy template to correct path
fs.copyFileSync(
  tempPath + "/" + filename,
  wageAgreementPath + "/" + wageAgreementTemplateFilename
);

// delete uploaded template at temp folder
fs.unlinkSync(tempPath + "/" + filename);

// delete file data
if (fs.existsSync(wageAgreementPath + "/" + wageAgreementFilename)) {
  fs.unlinkSync(wageAgreementPath + "/" + wageAgreementFilename);
}

// generate SQL syntax
let sqlString = "";

for (let i = 0; i < templateData.columns.length; i++) {
  const field = templateData.fields[i];
  const dataType = convertRuleToSQL(templateData.rules[i]);
```

```
    sqlString += `${field} ${dataType},\n`;
  }

  const connection = await mysql.connection();
  await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);

  // drop table if exist
  await connection.query(`DROP TABLE IF EXISTS ${wageAgreementTableName}`);

  // create table
  await connection.query(
    `CREATE TABLE IF NOT EXISTS ${wageAgreementTableName} (
      ${wageAgreementTableName}_id int NOT NULL AUTO_INCREMENT,
      ${sqlString}
      PRIMARY KEY (${wageAgreementTableName}_id)
    )`
  );

  connection.release();

  return res.json({
    message: "Import template success",
    status: 200,
  });
 }
```

Sistem akan mengecek apakah format *template* valid. Jika format *template* tersebut valid, sistem akan mengambil data-data kolom tersebut untuk ditambahkan dalam bentuk SQL. *Source code* dapat dilihat pada Segmen Program 4.47.

Segmen Program 4.48. *Import Data Wage Agreement*

```
async (req, res) => {
  if (req.fileValidationError) {
    return res.json({
      message: req.fileValidationError,
      status: 500,
    });
  }

  // update path
  const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
  wageAgreementPath = `public/uploads/${currentUser}/${wageAgreementTableName}`;

  // get uploaded data filename
  const filename = req.file.originalname;
```

```
// check if template not exist
if (!isFoundFile(wageAgreementPath, wageAgreementTemplateFilename)) {
  // delete uploaded file at temp folder
  fs.unlinkSync(tempPath + "/" + filename);

  return res.json({ message: "Template not found", status: 204 });
}

// get template column data
const templateData = await getExcelTemplateData(
  wageAgreementPath + "/" + wageAgreementTemplateFilename,
  worksheetTemplateColumnTitleIndex,
  worksheetTemplateRowRuleIndex,
  worksheetTemplateRowTitleIndex
);

// check is data valid
const isInvalidData = await isInvalidExcelData(
  templateData,
  tempPath + "/" + filename,
  worksheetDataColumnTitleIndex,
  worksheetDataRowDataIndex,
  worksheetDataRowTitleIndex
);

if (isInvalidData) {
  // delete uploaded template at temp folder
  fs.unlinkSync(tempPath + "/" + filename);

  return res.json({
    message: isInvalidData,
    status: 500,
  });
}

// if file valid
// get data
const data = await getExcelData(
  tempPath + "/" + filename,
  worksheetDataRowDataIndex
);

// create if directory not exist
const dir = wageAgreementPath;

if (!fs.existsSync(dir)) {
  fs.mkdirSync(dir, { recursive: true });
}
```

```
// copy file to correct path
fs.copyFileSync(
  tempPath + "/" + filename,
  wageAgreementPath + "/" + wageAgreementFilename
);

// delete uploaded file at temp folder
fs.unlinkSync(tempPath + "/" + filename);

const connection = await mysql.connection();
await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);
await connection.query("START TRANSACTION");
// if replace data
if (req.query.isReplace === "true") {
  // empty table
  await connection.query(`TRUNCATE TABLE ${wageAgreementTableName}`);
}

let sqlFieldString = "";
// init field string
for (let i = 0; i < templateData.columns.length; i++) {
  const field = templateData.fields[i];
  sqlFieldString += `${field}, `;
}

// remove comma
sqlFieldString =
  `${wageAgreementTableName}_id, ` + sqlFieldString.slice(0, -2);

let sqlValueString = "";

// init value string
for (let i = 0; i < data.length; i++) {
  const row = data[i];
  sqlValueString = "";

  for (let j = 0; j < row.length; j++) {
    const cellValue = row[j];
    const dataType = convertRuleToSQL(templateData.rules[j]);

    // if cell value not null
    if (cellValue) {
      if (dataType === "DATE") {
        sqlValueString += `STR_TO_DATE('${dateFns.format(
          excelDateToJSDate(cellValue),
          "dd-MM-yyyy"
        )}', '%d-%m-%Y'), `;
      } else if (dataType === "DOUBLE") {
        sqlValueString += `${cellValue}, `;
```

```
        } else {
          sqlValueString += `'${cellValue}', `;
        }
      } else {
        if (dataType === "DATE") {
          sqlValueString += `NULL, `;
        } else if (dataType === "DOUBLE") {
          sqlValueString += `0, `;
        } else {
          sqlValueString += `'', `;
        }
      }
    }

    // remove comma
    sqlValueString = `null, ` + sqlValueString.slice(0, -2);
    // transaction
    try {
      await connection.query(
        `INSERT INTO ${wageAgreementTableName} (${sqlFieldString})
        VALUES (${sqlValueString})`
      );
    } catch (error) {
      await connection.query("ROLLBACK");
      return res.json({
        message: "Import data failed",
        status: 400,
      });
    }
  }

  connection.query("COMMIT");
  connection.release();

  return res.json({
    message: "Import data success",
    status: 200,
  });
}
```

Sistem akan mengecek apakah *template* ada. Jika ada, sistem akan mengambil data kolom, lalu sistem akan mengecek apakah data yang diberikan valid. Jika valid, kosongkan tabel sesuai data untuk diisi ulang, setelah itu sistem akan mengisi *database* sesuai data yang diberikan. *Source code* dapat dilihat pada Segmen Program 4.48.

Segmen Program 4.49. *Export Template Wage Agreement*

```
async (req, res) => {
```

```
  // update path
  const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
  wageAgreementPath = `public/uploads/${currentUser}/${wageAgreementTableName}`;

  if (!isFoundFile(wageAgreementPath, wageAgreementTemplateFilename)) {
    return res.json({ message: "Template not found", status: 204 });
  }

  return res.download(
    wageAgreementPath + "/" + wageAgreementTemplateFilename,
    wageAgreementTemplateFilename,
    (err) => {
      if (err) {
        res.json({ message: "Could not download the file", status: 500 });
      }
    }
  );
}
```

Sistem akan mengecek apakah data *template* kesepakatan upah ada. Jika ada, sistem akan mengirim data *template* kesepakatan upah. *Source code* dapat dilihat pada Segmen Program 4.49.

Segmen Program 4.50. *Export Data Wage Agreement*

```
async (req, res) => {
  // update path
  const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
  wageAgreementPath = `public/uploads/${currentUser}/${wageAgreementTableName}`;

  let filterModel = req.body.filterModel;
  let columnVisibilityModel = req.body.columnVisibilityModel;
  let sortModel = req.body.sortModel;

  // get template column data
  const templateData = await getExcelTemplateData(
    wageAgreementPath + "/" + wageAgreementTemplateFilename,
    worksheetTemplateColumnTitleIndex,
    worksheetTemplateRowRuleIndex,
    worksheetTemplateRowTitleIndex
  );

  let columnVisibilityModelString = convertColumnVisibilityModelToSQL(
    columnVisibilityModel,
    templateData
  );
  let filterModelString = convertFilterModelToSQL(filterModel);
  let sortModelString = convertSortModelToSQL(sortModel);
```

```
  let templateDataAccordingRule =
    getExcelTemplateDataAccordingColumnVisibilityModel(
      columnVisibilityModel,
      templateData
    );

  const connection = await mysql.connection();
  await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);

  let query = await connection.query(`
    SELECT ${columnVisibilityModelString}
    FROM ${wageAgreementTableName}
    ${filterModelString}
    ${sortModelString}
  `);

  connection.release();

  // if file exist
  if (isFoundFile(wageAgreementPath, wageAgreementFilename)) {
    // delete uploaded file
    fs.unlinkSync(wageAgreementPath + "/" + wageAgreementFilename);
  }

  // export file
  await exportExcelData(
    query,
    wageAgreementPath + "/" + wageAgreementFilename,
    templateDataAccordingRule,
    worksheetDataRowDataIndex
  );

  return res.download(
    wageAgreementPath + "/" + wageAgreementFilename,
    wageAgreementFilename,
    (err) => {
      if (err) {
        res.json({ message: "Could not download the file", status: 500 });
      }
    }
  );
}
```

Sistem akan mengambil data *template* kesepakatan upah, lalu merangkai data sesuai *filter* yang digunakan, dan mengurutkan sesuai *sort* yang dipilih, setelah itu sistem akan mengirim data kesepakatan upah ke pengguna. *Source code* dapat dilihat pada Segmen Program 4.50.

Segmen Program 4.51. *Create Wage Agreement*

```
async (req, res) => {
  // update path
  const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
  wageAgreementPath = `public/uploads/${currentUser}/${wageAgreementTableName}`;

  let newWageAgreementData = req.body.newWageAgreementData;

  // get template column data
  const templateData = await getExcelTemplateData(
    wageAgreementPath + "/" + wageAgreementTemplateFilename,
    worksheetTemplateColumnTitleIndex,
    worksheetTemplateRowRuleIndex,
    worksheetTemplateRowTitleIndex
  );

  const isInvalidData = await isInvalidInsertData(
    templateData,
    newWageAgreementData
  );

  if (isInvalidData) {
    return res.json({
      message: isInvalidData,
      status: 500,
    });
  }

  // generate SQL syntax
  let newWageAgreementDataValues = Object.values(newWageAgreementData);

  // remove id field
  newWageAgreementDataValues.shift();

  let sqlFieldString = "";

  // init field string
  for (let i = 0; i < templateData.columns.length; i++) {
    const field = templateData.fields[i];
    sqlFieldString += `${field}, `;
  }

  // remove comma
  sqlFieldString =
    `${wageAgreementTableName}_id, ` + sqlFieldString.slice(0, -2);

  let sqlValueString = "";

  // init value string
```

```
  for (let i = 0; i < newWageAgreementDataValues.length; i++) {
    const newWageAgreementDataValue = newWageAgreementDataValues[i];
    const dataType = convertRuleToSQL(templateData.rules[i]);

    if (newWageAgreementDataValue) {
      if (dataType === "DATE") {
        sqlValueString += `STR_TO_DATE('${dateFns.format(
          dateFns.parseISO(newWageAgreementDataValue),
          "dd-MM-yyyy"
        )}', '%d-%m-%Y'), `;
      } else if (dataType === "DOUBLE") {
        sqlValueString += `${newWageAgreementDataValue}, `;
      } else {
        sqlValueString += `'${newWageAgreementDataValue}', `;
      }
    } else {
      if (dataType === "DATE") {
        sqlValueString += `NULL, `;
      } else if (dataType === "DOUBLE") {
        sqlValueString += `0, `;
      } else {
        sqlValueString += `'', `;
      }
    }
  }

  // remove comma
  sqlValueString = `null, ` + sqlValueString.slice(0, -2);
  const connection = await mysql.connection();
  await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);
  await connection.query(`
    INSERT INTO ${wageAgreementTableName} (${sqlFieldString})
    VALUES (
    ${sqlValueString}
    )
  `);
  connection.release();

  return res.json({
    message: "Create wage agreement success",
    status: 200,
  });
}
```

Sistem akan mengambil data kolom pada *template*, lalu sistem akan mengecek apakah data yang baru adalah valid. Jika data tersebut valid, sistem akan memasukkan data kesepakatan upah pada *database*. *Source code* dapat dilihat pada Segmen Program 4.51.

Segmen Program 4.52. *Update Wage Agreement*

```
async (req, res) => {
  // update path
  const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
  wageAgreementPath = `public/uploads/${currentUser}/${wageAgreementTableName}`;

  let newWageAgreementData = req.body.newWageAgreementData;

  // get template column data
  const templateData = await getExcelTemplateData(
    wageAgreementPath + "/" + wageAgreementTemplateFilename,
    worksheetTemplateColumnTitleIndex,
    worksheetTemplateRowRuleIndex,
    worksheetTemplateRowTitleIndex
  );

  const isInvalidData = await isInvalidUpdateData(
    templateData,
    newWageAgreementData
  );

  if (isInvalidData) {
    return res.json({
      message: isInvalidData,
      status: 500,
    });
  }

  // generate SQL syntax
  let newWageAgreementDataValues = Object.values(newWageAgreementData);

  // remove id field
  newWageAgreementDataValues.shift();

  // generate SQL syntax
  let sqlString = "";

  for (let i = 0; i < newWageAgreementDataValues.length; i++) {
    const newWageAgreementDataValue = newWageAgreementDataValues[i];
    const dataType = convertRuleToSQL(templateData.rules[i]);

    sqlString += `${templateData.fields[i]} = `;

    if (dataType === "DATE") {
      if (newWageAgreementDataValue) {
        sqlString += `STR_TO_DATE('${dateFns.format(
          dateFns.parseISO(newWageAgreementDataValue),
          "dd-MM-yyyy"
        )}', '%d-%m-%Y'), `;
```

```
      } else {
        sqlString += `NULL, `;
      }
    } else if (dataType === "DOUBLE") {
      sqlString += `${newWageAgreementDataValue}, `;
    } else {
      sqlString += `'${newWageAgreementDataValue}', `;
    }
  }

  // remove comma
  sqlString = sqlString.slice(0, -2);

  const connection = await mysql.connection();
  await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);
  await connection.query(`
    UPDATE ${wageAgreementTableName}
    SET ${sqlString}
    WHERE ${wageAgreementTableName}_id =
${newWageAgreementData.wage_agreement_id}
  `);
  connection.release();

  return res.json({
    message: "Update wage agreement success",
    status: 200,
  });
 }
```

Sistem akan mengambil data kolom pada *template*, lalu sistem akan mengecek apakah data yang baru adalah valid. Jika data tersebut valid, sistem akan memperbarui data pada *database*. *Source code* dapat dilihat pada Segmen Program 4.52.

Segmen Program 4.53. *Delete Wage Agreement*

```
                                async (req, res) => {
  // update path
  const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
  wageAgreementPath = `public/uploads/${currentUser}/${wageAgreementTableName}`;

  let selectedWageAgreementID = req.body.selectedWageAgreementID;
  let sqlValueString = "";

  for (let i = 0; i < selectedWageAgreementID.length; i++) {
    sqlValueString += `${selectedWageAgreementID[i]}, `;
  }

  // remove comma
```

```
  sqlValueString = sqlValueString.slice(0, -2);

  const connection = await mysql.connection();
  await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);
  await connection.query(`
    DELETE FROM ${wageAgreementTableName}
    WHERE ${wageAgreementTableName}_id IN (${sqlValueString})
  `);
  connection.release();

  return res.json({
    message: "Delete wage agreement success",
    status: 200,
  });
 }
```

Sistem akan menghapus data yang terdapat pada *database* sesuai dengan ID yang dikirimkan pengguna dari *website*. *Source code* dapat dilihat pada Segmen Program 4.53.

4.2.5. Menu Rekap Absensi

Segmen Program 4.54. *Get Attendance Summary*

```
async (req, res) => {
  // update path
  const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
  attendanceSummaryPath =
`public/uploads/${currentUser}/${attendanceSummaryTableName}`;

  // check if template exist
  if (
    !isFoundFile(attendanceSummaryPath, attendanceSummaryTemplateFilename)
  ) {
    return res.json({ status: 204 });
  }

  let filterModel = req.body.filterModel;
  let pageIndex = req.body.pageIndex;
  let rowsPerPage = req.body.rowsPerPage;
  let sortModel = req.body.sortModel;

  // get template column data
  const templateData = await getExcelTemplateData(
    attendanceSummaryPath + "/" + attendanceSummaryTemplateFilename,
    worksheetTemplateColumnTitleIndex,
    worksheetTemplateRowRuleIndex,
    worksheetTemplateRowTitleIndex
  );
```

```
  // add ID column
  templateData.columns.unshift("ID");
  templateData.fields.unshift("attendance_summary_id");
  templateData.rules.unshift("id");

  let filterModelString = convertFilterModelToSQL(filterModel);
  let sortModelString = convertSortModelToSQL(sortModel);

  // fetch data
  const connection = await mysql.connection();
  await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);

  let query = await connection.query(`
    SELECT *
    FROM ${attendanceSummaryTableName}
    ${filterModelString}
    ${sortModelString}
    LIMIT ${rowsPerPage}
    OFFSET ${pageIndex * rowsPerPage}
  `);

  // get filtered row count
  let rowCount = await connection.query(`
    SELECT COUNT(*) AS row_count
    FROM ${attendanceSummaryTableName}
    ${filterModelString}
    ${sortModelString}
  `);

  connection.release();

  return res.json({
    payload: { query: query, rowCount: rowCount[0].row_count },
    status: 200,
    templateData: templateData,
  });
 }
```

Sistem akan mengecek apakah *template* ada. Jika ada, sistem akan mengambil data kolom, menambahkan kolom ID, mengambil data tersebut untuk diberikan ke pengguna, lalu mengirim data tersebut ke *website*. *Source code* dapat dilihat pada Segmen Program 4.54.

Segmen Program 4.55. *Import Template Attendance Summary*

```
async (req, res) => {
  if (req.fileValidationError) {
    return res.json({
      message: req.fileValidationError,
```

```
      status: 500,
    });
  }

  // get uploaded template filename
  const filename = req.file.originalname;

  // check is template valid
  const isInvalidTemplate = await isInvalidExcelTemplateData(
    modules.ATTENDANCE_SUMMARY.key,
    tempPath + "/" + filename,
    worksheetTemplateColumnTitleIndex,
    worksheetTemplateRowRuleIndex,
    worksheetTemplateRowTitleIndex
  );

  if (isInvalidTemplate) {
    // delete uploaded template at temp folder
    fs.unlinkSync(tempPath + "/" + filename);

    return res.json({
      message: isInvalidTemplate,
      status: 500,
    });
  }

  // if template valid
  // get template column data
  const templateData = await getExcelTemplateData(
    tempPath + "/" + filename,
    worksheetTemplateColumnTitleIndex,
    worksheetTemplateRowRuleIndex,
    worksheetTemplateRowTitleIndex
  );

  // update path
  const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
  attendanceSummaryPath =
`public/uploads/${currentUser}/${attendanceSummaryTableName}`;

  // create if directory not exist
  const dir = attendanceSummaryPath;

  if (!fs.existsSync(dir)) {
    fs.mkdirSync(dir, { recursive: true });
  }

  // copy template to correct path
  fs.copyFileSync(
```

```
    tempPath + "/" + filename,
    attendanceSummaryPath + "/" + attendanceSummaryTemplateFilename
  );

  // delete uploaded template at temp folder
  fs.unlinkSync(tempPath + "/" + filename);

  // delete file data
  if (
    fs.existsSync(attendanceSummaryPath + "/" + attendanceSummaryFilename)
  ) {
    fs.unlinkSync(attendanceSummaryPath + "/" + attendanceSummaryFilename);
  }

  // generate SQL syntax
  let sqlString = "";

  for (let i = 0; i < templateData.columns.length; i++) {
    const field = templateData.fields[i];
    const dataType = convertRuleToSQL(templateData.rules[i]);

    sqlString += `${field} ${dataType},\n`;
  }

  const connection = await mysql.connection();
  await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);

  // drop table if exist
  await connection.query(
    `DROP TABLE IF EXISTS ${attendanceSummaryTableName}`
  );

  // create table
  await connection.query(
    `CREATE TABLE IF NOT EXISTS ${attendanceSummaryTableName} (
      ${attendanceSummaryTableName}_id int NOT NULL AUTO_INCREMENT,
      ${sqlString}
      PRIMARY KEY (${attendanceSummaryTableName}_id)
    )`
  );

  connection.release();

  return res.json({
    message: "Import template success",
    status: 200,
  });
}
```

Sistem akan mengecek apakah format *template* valid. Jika format *template* tersebut valid, sistem akan mengambil data-data kolom tersebut untuk ditambahkan dalam bentuk SQL. *Source code* dapat dilihat pada Segmen Program 4.55.

Segmen Program 4.56. *Import Data Attendance Summary*

```
async (req, res) => {
  if (req.fileValidationError) {
    return res.json({
      message: req.fileValidationError,
      status: 500,
    });
  }

  // update path
  const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
  attendanceSummaryPath =
`public/uploads/${currentUser}/${attendanceSummaryTableName}`;

  // get uploaded data filename
  const filename = req.file.originalname;

  // check if template not exist
  if (
    !isFoundFile(attendanceSummaryPath, attendanceSummaryTemplateFilename)
  ) {
    // delete uploaded file at temp folder
    fs.unlinkSync(tempPath + "/" + filename);

    return res.json({ message: "Template not found", status: 204 });
  }

  // get template column data
  const templateData = await getExcelTemplateData(
    attendanceSummaryPath + "/" + attendanceSummaryTemplateFilename,
    worksheetTemplateColumnTitleIndex,
    worksheetTemplateRowRuleIndex,
    worksheetTemplateRowTitleIndex
  );

  // check is data valid
  const isInvalidData = await isInvalidExcelData(
    templateData,
    tempPath + "/" + filename,
    worksheetDataColumnTitleIndex,
    worksheetDataRowDataIndex,
    worksheetDataRowTitleIndex
  );
```

```
if (isInvalidData) {
  // delete uploaded template at temp folder
  fs.unlinkSync(tempPath + "/" + filename);

  return res.json({
    message: isInvalidData,
    status: 500,
  });
}

// if file valid
// get data
const data = await getExcelData(
  tempPath + "/" + filename,
  worksheetDataRowDataIndex
);

// create if directory not exist
const dir = attendanceSummaryPath;

if (!fs.existsSync(dir)) {
  fs.mkdirSync(dir, { recursive: true });
}

// copy file to correct path
fs.copyFileSync(
  tempPath + "/" + filename,
  attendanceSummaryPath + "/" + attendanceSummaryFilename
);

// delete uploaded file at temp folder
fs.unlinkSync(tempPath + "/" + filename);

const connection = await mysql.connection();
await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);
await connection.query("START TRANSACTION");
// if replace data
if (req.query.isReplace === "true") {
  // empty table
  await connection.query(`TRUNCATE TABLE ${attendanceSummaryTableName}`);
}

let sqlFieldString = "";
// init field string
for (let i = 0; i < templateData.columns.length; i++) {
  const field = templateData.fields[i];
  sqlFieldString += `${field}, `;
}
```

```
// remove comma
sqlFieldString =
  `${attendanceSummaryTableName}_id, ` + sqlFieldString.slice(0, -2);

let sqlValueString = "";

// init value string
for (let i = 0; i < data.length; i++) {
  const row = data[i];
  sqlValueString = "";

  for (let j = 0; j < row.length; j++) {
    const cellValue = row[j];
    const dataType = convertRuleToSQL(templateData.rules[j]);

    // if cell value not null
    if (cellValue) {
      if (dataType === "DATE") {
        sqlValueString += `STR_TO_DATE('${dateFns.format(
          excelDateToJSDate(cellValue),
          "dd-MM-yyyy"
        )}', '%d-%m-%Y'), `;
      } else if (dataType === "DOUBLE") {
        sqlValueString += `${cellValue}, `;
      } else {
        sqlValueString += `'${cellValue}', `;
      }
    } else {
      if (dataType === "DATE") {
        sqlValueString += `NULL, `;
      } else if (dataType === "DOUBLE") {
        sqlValueString += `0, `;
      } else {
        sqlValueString += `'', `;
      }
    }
  }

  // remove comma
  sqlValueString = `null, ` + sqlValueString.slice(0, -2);
  // transaction
  try {
    await connection.query(
      `INSERT INTO ${attendanceSummaryTableName} (${sqlFieldString})
      VALUES (${sqlValueString})`
    );
  } catch (error) {
    await connection.query("ROLLBACK");
```

```
      return res.json({
        message: "Import data failed",
        status: 400,
      });
    }
  }
  connection.query("COMMIT");
  connection.release();

  return res.json({
    message: "Import data success",
    status: 200,
  });
 }
```

Sistem akan mengecek apakah *template* ada. Jika ada, sistem akan mengambil data kolom, lalu sistem akan mengecek apakah data yang diberikan valid. Jika valid, kosongkan tabel sesuai data untuk diisi ulang, setelah itu sistem akan mengisi *database* sesuai data yang diberikan. *Source code* dapat dilihat pada Segmen Program 4.56.

Segmen Program 4.57. *Export Template Attendance Summary*

```
async (req, res) => {
  // update path
  const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
  attendanceSummaryPath =
`public/uploads/${currentUser}/${attendanceSummaryTableName}`;

  if (
    !isFoundFile(attendanceSummaryPath, attendanceSummaryTemplateFilename)
  ) {
    return res.json({ message: "Template not found", status: 204 });
  }

  return res.download(
    attendanceSummaryPath + "/" + attendanceSummaryTemplateFilename,
    attendanceSummaryTemplateFilename,
    (err) => {
      if (err) {
        res.json({ message: "Could not download the file", status: 500 });
      }
    }
  );
 }
```

Sistem akan mengecek apakah data *template* rekap absensi ada. Jika ada, sistem akan mengirim data *template* rekap absensi. *Source code* dapat dilihat pada Segmen Program 4.57.

Segmen Program 4.58. *Export Data Attendance Summary*

```
async (req, res) => {
  // update path
  const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
  attendanceSummaryPath =
`public/uploads/${currentUser}/${attendanceSummaryTableName}`;

  let filterModel = req.body.filterModel;
  let columnVisibilityModel = req.body.columnVisibilityModel;
  let sortModel = req.body.sortModel;

  // get template column data
  const templateData = await getExcelTemplateData(
   attendanceSummaryPath + "/" + attendanceSummaryTemplateFilename,
   worksheetTemplateColumnTitleIndex,
   worksheetTemplateRowRuleIndex,
   worksheetTemplateRowTitleIndex
  );

  let columnVisibilityModelString = convertColumnVisibilityModelToSQL(
   columnVisibilityModel,
   templateData
  );
  let filterModelString = convertFilterModelToSQL(filterModel);
  let sortModelString = convertSortModelToSQL(sortModel);
  let templateDataAccordingRule =
   getExcelTemplateDataAccordingColumnVisibilityModel(
    columnVisibilityModel,
    templateData
   );

  const connection = await mysql.connection();
  await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);

  let query = await connection.query(`
   SELECT ${columnVisibilityModelString}
   FROM ${attendanceSummaryTableName}
   ${filterModelString}
   ${sortModelString}
  `);

  connection.release();

  // if file exist
  if (isFoundFile(attendanceSummaryPath, attendanceSummaryFilename)) {
   // delete uploaded file
   fs.unlinkSync(attendanceSummaryPath + "/" + attendanceSummaryFilename);
  }
```

```
  // export file
  await exportExcelData(
    query,
    attendanceSummaryPath + "/" + attendanceSummaryFilename,
    templateDataAccordingRule,
    worksheetDataRowDataIndex
  );

  return res.download(
    attendanceSummaryPath + "/" + attendanceSummaryFilename,
    attendanceSummaryFilename,
    (err) => {
      if (err) {
        res.json({ message: "Could not download the file", status: 500 });
      }
    }
  );
}
```

Sistem akan mengambil data *template*, lalu merangkai data sesuai *filter* yang digunakan, dan mengurutkan sesuai *sort* yang dipilih, setelah itu sistem akan mengirim data ke pengguna. *Source code* dapat dilihat pada Segmen Program 4.58.

Segmen Program 4.59. *Create Attendance Summary*

```
async (req, res) => {
  // update path
  const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
  attendanceSummaryPath =
`public/uploads/${currentUser}/${attendanceSummaryTableName}`;

  let newAttendanceSummaryData = req.body.newAttendanceSummaryData;

  // get template column data
  const templateData = await getExcelTemplateData(
    attendanceSummaryPath + "/" + attendanceSummaryTemplateFilename,
    worksheetTemplateColumnTitleIndex,
    worksheetTemplateRowRuleIndex,
    worksheetTemplateRowTitleIndex
  );

  const isInvalidData = await isInvalidInsertData(
    templateData,
    newAttendanceSummaryData
  );

  if (isInvalidData) {
    return res.json({
```

```
    message: isInvalidData,
    status: 500,
   });
}

// generate SQL syntax
let newAttendanceSummaryDataValues = Object.values(
  newAttendanceSummaryData
);

// remove id field
newAttendanceSummaryDataValues.shift();

let sqlFieldString = "";

// init field string
for (let i = 0; i < templateData.columns.length; i++) {
  const field = templateData.fields[i];
  sqlFieldString += `${field}, `;
}

// remove comma
sqlFieldString =
  `${attendanceSummaryTableName}_id, ` + sqlFieldString.slice(0, -2);

let sqlValueString = "";

// init value string
for (let i = 0; i < newAttendanceSummaryDataValues.length; i++) {
  const newAttendanceSummaryDataValue = newAttendanceSummaryDataValues[i];
  const dataType = convertRuleToSQL(templateData.rules[i]);

  if (newAttendanceSummaryDataValue) {
    if (dataType === "DATE") {
      sqlValueString += `STR_TO_DATE('${dateFns.format(
        dateFns.parseISO(newAttendanceSummaryDataValue),
        "dd-MM-yyyy"
      )}', '%d-%m-%Y'), `;
    } else if (dataType === "DOUBLE") {
      sqlValueString += `${newAttendanceSummaryDataValue}, `;
    } else {
      sqlValueString += `'${newAttendanceSummaryDataValue}', `;
    }
  } else {
    if (dataType === "DATE") {
      sqlValueString += `NULL, `;
    } else if (dataType === "DOUBLE") {
      sqlValueString += `0, `;
    } else {
```

```
        sqlValueString += `'', `;
      }
     }
    }

    // remove comma
    sqlValueString = `null, ` + sqlValueString.slice(0, -2);

    const connection = await mysql.connection();
    await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);
    await connection.query(`
     INSERT INTO ${attendanceSummaryTableName} (${sqlFieldString})
     VALUES (
     ${sqlValueString}
     )
    `);
    connection.release();

    return res.json({
     message: "Create attendanceSummary success",
     status: 200,
    });
  }
```

Sistem akan mengambil data kolom pada *template*, lalu sistem akan mengecek apakah data yang baru adalah valid. Jika data tersebut valid, sistem akan memasukkan data rekap absensi pada *database*. *Source code* dapat dilihat pada Segmen Program 4.59.

Segmen Program 4.60. *Update Data Attendance Summary*

```
async (req, res) => {
    // update path
    const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
    attendanceSummaryPath =
`public/uploads/${currentUser}/${attendanceSummaryTableName}`;

    let newAttendanceSummaryData = req.body.newAttendanceSummaryData;

    // get template column data
    const templateData = await getExcelTemplateData(
     attendanceSummaryPath + "/" + attendanceSummaryTemplateFilename,
     worksheetTemplateColumnTitleIndex,
     worksheetTemplateRowRuleIndex,
     worksheetTemplateRowTitleIndex
    );

    const isInvalidData = await isInvalidUpdateData(
     templateData,
```

```
  newAttendanceSummaryData
);

if (isInvalidData) {
  return res.json({
    message: isInvalidData,
    status: 500,
  });
}

// generate SQL syntax
let newAttendanceSummaryDataValues = Object.values(
  newAttendanceSummaryData
);

// remove id field
newAttendanceSummaryDataValues.shift();

// generate SQL syntax
let sqlString = "";

for (let i = 0; i < newAttendanceSummaryDataValues.length; i++) {
  const newAttendanceSummaryDataValue = newAttendanceSummaryDataValues[i];
  const dataType = convertRuleToSQL(templateData.rules[i]);

  sqlString += `${templateData.fields[i]} = `;

  if (dataType === "DATE") {
    if (newAttendanceSummaryDataValue) {
      sqlString += `STR_TO_DATE('${dateFns.format(
        dateFns.parseISO(newAttendanceSummaryDataValue),
        "dd-MM-yyyy"
      )}', '%d-%m-%Y'), `;
    } else {
      sqlString += `NULL, `;
    }
  } else if (dataType === "DOUBLE") {
    sqlString += `${newAttendanceSummaryDataValue}, `;
  } else {
    sqlString += `'${newAttendanceSummaryDataValue}', `;
  }
}

// remove comma
sqlString = sqlString.slice(0, -2);

const connection = await mysql.connection();
await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);
await connection.query(`
```

```
    UPDATE ${attendanceSummaryTableName}
    SET ${sqlString}
    WHERE ${attendanceSummaryTableName}_id =
${newAttendanceSummaryData.attendance_summary_id}
  `);
  connection.release();

  return res.json({
    message: "Update attendanceSummary success",
    status: 200,
  });
 }
```

Sistem akan mengambil data kolom pada *template*, lalu sistem akan mengecek apakah data yang baru adalah valid. Jika data tersebut valid, sistem akan memperbarui data pada *database*. *Source code* dapat dilihat pada Segmen Program 4.60.

Segmen Program 4.61. *Delete Data Attendance Summary*

```
async (req, res) => {
  // update path
  const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
  attendanceSummaryPath =
`public/uploads/${currentUser}/${attendanceSummaryTableName}`;

  let selectedAttendanceSummaryID = req.body.selectedAttendanceSummaryID;
  let sqlValueString = "";

  for (let i = 0; i < selectedAttendanceSummaryID.length; i++) {
    sqlValueString += `${selectedAttendanceSummaryID[i]}, `;
  }

  // remove comma
  sqlValueString = sqlValueString.slice(0, -2);

  const connection = await mysql.connection();
  await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);
  await connection.query(`
    DELETE FROM ${attendanceSummaryTableName}
    WHERE ${attendanceSummaryTableName}_id IN (${sqlValueString})
  `);
  connection.release();

  return res.json({
    message: "Delete attendanceSummary success",
    status: 200,
  });
 }
```

Sistem akan menghapus data yang terdapat pada *database* sesuai dengan ID yang dikirimkan pengguna dari *website*. *Source code* dapat dilihat pada Segmen Program 4.61.

4.2.6. Menu Perhitungan Penggajian

Segmen Program 4.62. *Get Data Payroll Calculation*

```
async (req, res) => {
  const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
  const connection = await mysql.connection();
  await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);

  let attendance_summary = await connection.query(`
    SELECT *
    FROM attendance_summary
  `);

  let holiday = await connection.query(`
    SELECT *
    FROM holiday
  `);

  let period = await connection.query(`
    SELECT *
    FROM period
  `);

  let wage_agreement = await connection.query(`
    SELECT *
    FROM wage_agreement
  `);

  let workday = await connection.query(`
    SELECT *
    FROM workday
  `);

  let work_agreement = await connection.query(`
    SELECT *
    FROM work_agreement
  `);

  let result = calculateAttendance(
    attendance_summary,
    holiday,
    period,
    wage_agreement,
    workday,
```

```
  work_agreement
);

try {
  await connection.query(`TRUNCATE TABLE attendance_calculation`);
  await connection.query("START TRANSACTION");

  let insertData = "";
  for (let i = 0; i < result.length; i++) {
    insertData = "";
    for (const key in result[i]) {
      if (key === "tanggal_awal_bekerja") {
        if (result[i]["tanggal_awal_bekerja"]) {
          insertData += `STR_TO_DATE('${dateFns.format(
            result[i]["tanggal_awal_bekerja"],
            "dd-MM-yyyy"
          )}', '%d-%m-%Y'),`;
        } else {
          insertData += `NULL,`;
        }
      } else if (key === "tanggal_akhir_bekerja") {
        if (result[i]["tanggal_akhir_bekerja"]) {
          insertData += `STR_TO_DATE('${dateFns.format(
            result[i]["tanggal_akhir_bekerja"],
            "dd-MM-yyyy"
          )}', '%d-%m-%Y'),`;
        } else {
          insertData += `NULL,`;
        }
      } else if (typeof result[i][key] === "number") {
        insertData += `${result[i][key]},`;
      } else {
        insertData += `'${result[i][key]}',`;
      }
    }
    insertData = insertData.slice(0, -1);

    await connection.query(`
      INSERT INTO attendance_calculation
      VALUES (null, ${insertData})`);
  }
} catch (error) {
  await connection.query("ROLLBACK");
} finally {
  connection.query("COMMIT");
}

let filterModel = req.body.filterModel;
let pageIndex = req.body.pageIndex;
```

```
  let rowsPerPage = req.body.rowsPerPage;
  let sortModel = req.body.sortModel;

  let filterModelString = convertFilterModelToSQL(filterModel);
  let sortModelString = convertSortModelToSQL(sortModel);

  query = await connection.query(`
    SELECT *
    FROM ${attendanceCalculationTableName}
    ${filterModelString}
    ${sortModelString}
    LIMIT ${rowsPerPage}
    OFFSET ${pageIndex * rowsPerPage}
  `);

  // get filtered row count
  let rowCount = await connection.query(`
    SELECT COUNT(*) AS row_count
    FROM ${attendanceCalculationTableName}
    ${filterModelString}
    ${sortModelString}
  `);

  connection.release();

  return res.json({
    payload: { query: query, rowCount: rowCount[0].row_count },
    status: 200,
  });
 }
```

Sistem akan mengambil semua data terlebih dahulu, lalu memproses data pada *calculateattendance function,* setelah itu sistem akan mengosongkan tabel *attendance calculation* untuk memperbarui data, dan sistem akan mengirim tabel yang sudah diperbarui ke pengguna. *Source code* dapat dilihat pada Segmen Program 4.62.

Segmen Program 4.63. *Export Data Payroll Calculation*

```
async (req, res) => {
  // update path
  const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
  attendanceCalculationPath =
`public/uploads/${currentUser}/${attendanceCalculationTableName}`;

  let filterModel = req.body.filterModel;
  let columnVisibilityModel = req.body.columnVisibilityModel;
  let sortModel = req.body.sortModel;
```

```
const templateData = {
 columns: [
  "NIP",
  "Nama",
  "Departemen",
  "Tanggal Awal Bekerja",
  "Tanggal Akhir Bekerja",
  "Status DB",
  "Status HK",
  "Kode HK",
  "Jenis Perhitungan Absensi",
  "HK Kalender",
  "HK Kalender Minus Tanggal Merah",
  "Total Tanggal Merah",
  "Total Tidak Hadir",
  "Total Absen",
  "HK Real",
  "Gaji Pokok per Bulan",
  "Gaji Pokok per Hari",
  "Tunjangan Tetap per Bulan",
  "Tunjangan Tetap per Hari",
  "Tunjangan Tidak Tetap per Bulan",
  "Tunjangan Tidak Tetap per Hari",
  "Gaji Sebelum Potongan",
  "Potongan Tidak Hadir",
  "Potongan Total Absen",
  "Gaji Setelah Potongan",
 ],
 fields: [
  "nip",
  "nama",
  "departemen",
  "tanggal_awal_bekerja",
  "tanggal_akhir_bekerja",
  "status_db",
  "status_hk",
  "kode_hk",
  "jenis_perhitungan_absensi",
  "hk_kalender",
  "hk_kalender_minus_tanggal_merah",
  "tanggal_merah",
  "total_tidak_hadir",
  "total_absen",
  "hk_real",
  "gp_per_bulan",
  "gp_per_hari",
  "total_tunjangan_tetap_per_bulan",
  "total_tunjangan_tetap_per_hari",
  "total_tunjangan_tidak_tetap_per_bulan",
```

```
    "total_tunjangan_tidak_tetap_per_hari",
    "gaji_sebelum_potongan",
    "potongan_tidak_hadir",
    "potongan_absen",
    "gaji_setelah_potongan",
  ],
  rules: [],
};

let columnVisibilityModelString = convertColumnVisibilityModelToSQL(
  columnVisibilityModel,
  templateData
);
let filterModelString = convertFilterModelToSQL(filterModel);
let sortModelString = convertSortModelToSQL(sortModel);
let templateDataAccordingRule =
  getExcelTemplateDataAccordingColumnVisibilityModel(
    columnVisibilityModel,
    templateData
  );

const connection = await mysql.connection();
await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);

let query = await connection.query(`
  SELECT ${columnVisibilityModelString}
  FROM ${attendanceCalculationTableName}
  ${filterModelString}
  ${sortModelString}
`);

connection.release();

// if file exist
if (isFoundFile(attendanceCalculationPath, attendanceCalculationFilename)) {
  // delete uploaded file
  fs.unlinkSync(
    attendanceCalculationPath + "/" + attendanceCalculationFilename
  );
}

// export file
await exportExcelData(
  query,
  attendanceCalculationPath + "/" + attendanceCalculationFilename,
  templateDataAccordingRule,
  worksheetDataRowDataIndex
);
```

```
  return res.download(
    attendanceCalculationPath + "/" + attendanceCalculationFilename,
    attendanceCalculationFilename,
    (err) => {
      if (err) {
        res.json({ message: "Could not download the file", status: 500 });
      }
    }
  );
}
```

Sistem akan menyiapkan kolom - kolom yang akan diberikan ke pengguna, lalu sistem akan mengambil semua data penggajian untuk diberikan ke pengguna. *Source code* dapat dilihat pada Segmen Program 4.63.

4.2.7. Menu Pengaturan

Segmen Program 4.64. *Create User*

```
async (req, res) => {
  const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
  const connection = await mysql.connection();
  await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);

  // check username used
  let query = await connection.query(`
    SELECT * FROM user WHERE username = '${req.body.username}'
  `);

  if (query.length > 0) {
    connection.release();

    return res.json({
      message: "Username already taken",
      status: 400,
    });
  }

  let passwordHash = bcrypt.hashSync(
    req.body.password,
    parseInt(process.env.SALT_ROUNDS)
  );

  await connection.query(`
    INSERT INTO user VALUES (
      null,
      ${req.body.role_id},
```

```
      '${req.body.username}',
      '${passwordHash}',
      0
    )
  `);

  connection.release();

  return res.json({
    message: "Add user success",
    status: 200,
  });
}
```

Sistem akan mengecek apakah *username* sudah dipakai atau belum. Jika belum, sistem akan *hash* password, setelah itu sistem akan membuat *user*. *Source code* dapat dilihat pada Segmen Program 4.64.

Segmen Program 4.65. *Get User*

```
async (req, res) => {
  const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
  const connection = await mysql.connection();
  await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);

  let query = await connection.query(`
    SELECT u.*, urj.name AS role_name
    FROM user u
    LEFT JOIN user_role_json urj
    ON u.role_id = urj.id
    WHERE u.is_deleted = 0
  `);

  connection.release();

  return res.json({
    payload: query,
    status: 200,
  });
}
```

Sistem akan mengambil semua data *user*, lalu menambahkan kolom nama peran dengan *join. Source code* dapat dilihat pada Segmen Program 4.65.

Segmen Program 4.66. *Update User*

```
async (req, res) => {
```

```
  const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
  const connection = await mysql.connection();
  await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);

  // check username used
  let query = await connection.query(`
   SELECT * FROM user WHERE username = '${req.body.username}'
  `);

  if (query.length > 0) {
   connection.release();

   return res.json({
    message: "Username already taken",
    status: 400,
   });
  }

  await connection.query(`
   UPDATE user
   SET role_id = ${req.body.role_id}
   WHERE id = ${req.body.id}
  `);

  connection.release();

  return res.json({
   message: "Edit user success",
   status: 200,
  });
 }
```

Sistem akan mengecek apakah *username* sudah dipakai. Jika belum mengubah nama, sistem akan mengubah *role* pengguna sesuai dengan ID yang dikirim. *Source code* dapat dilihat pada Segmen Program 4.66.

Segmen Program 4.67. *Delete User*

```
async (req, res) => {
  const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
  const connection = await mysql.connection();
  await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);

  await connection.query(
   `UPDATE user SET is_deleted = 1 WHERE id = ${req.params.id}`
  );

  connection.release();
```

```
  return res.json({
    message: "Delete user success",
    status: 200,
  });
}
```

Sistem akan menghapus pengguna dengan cara memperbarui (*soft delete*) sesuai dengan ID pengguna yang dihapus. *Source code* dapat dilihat pada Segmen Program 4.67.

Segmen Program 4.68. *Add Role*

```
async (req, res) => {
  const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
  const connection = await mysql.connection();
  await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);

  // check role name
  let query = await connection.query(`
    SELECT * FROM user_role_json WHERE name = '${req.body.name}'
  `);

  if (query.length > 0) {
    connection.release();
    return res.json({
      message: "Role name already taken",
      status: 400,
    });
  }

  await connection.query(`
    INSERT INTO user_role_json VALUES (
      null,
      '${req.body.name}',
      '${req.body.json}',
      0
    )
  `);

  connection.release();

  return res.json({
    message: "Add role success",
    status: 200,
  });
}
```

Sistem akan mengecek nama peran. Jika nama peran belum dipakai, sistem akan menambahkan peran ke *database*. *Source code* dapat dilihat pada Segmen Program 4.68.

Segmen Program 4.69. *Get Role*

```
async (req, res) => {
  const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
  const connection = await mysql.connection();
  await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);

  let query = await connection.query(`
    SELECT * FROM user_role_json WHERE is_deleted = 0
  `);

  connection.release();

  return res.json({
    payload: query,
    status: 200,
  });
 }
```

Sistem akan mengirimkan data peran yang tersimpan di *database* ke pengguna. *Source code* dapat dilihat pada Segmen Program 4.69.

Segmen Program 4.70. *Update Role*

```
async (req, res) => {
  const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
  const connection = await mysql.connection();
  await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);

  // update role json
  await connection.query(`
    UPDATE user_role_json
    SET name = '${req.body.name}',
    json = '${req.body.json}'
    WHERE id = ${req.body.id}
  `);

  connection.release();

  return res.json({
    message: "Edit role success",
    status: 200,
  });
 }
```

Sistem akan memperbarui peran dengan ID tertentu sesuai dengan data yang dikirimkan. *Source code* dapat dilihat pada Segmen Program 4.70.

Segmen Program 4.71. *Delete Role*

```
async (req, res) => {
   const currentUser = getCurrentUserFromToken(req.cookies["access-token"]);
   const connection = await mysql.connection();
   await connection.query(`USE ${process.env.DB_NAME}_${currentUser}`);

   await connection.query(
     `UPDATE user_role_json SET is_deleted = 1 WHERE id = ${req.params.id}`
   );

   connection.release();

   return res.json({
     message: "Delete role success",
     status: 200,
   });
  }
```

Sistem akan menghapus peran dengan cara memperbarui (*soft delete*) sesuai dengan ID peran yang dihapus. *Source code* dapat dilihat pada Segmen Program 4.71.