DAFTAR PUSTAKA

Cahyo Santoso, B., Natasya, Y., Willian, S., & Alfando, F. (t.t.-a). *Tinjauan Pustaka Sistematis terhadap Basis Data MongoDB*.

Cahyo Santoso, B., Natasya, Y., Willian, S., & Alfando, F. (t.t.-b). *Tinjauan Pustaka Sistematis terhadap Basis Data MongoDB*.

Charras, C., & Lecroq, T. (2004). *Handbook of Exact String Matching Algorithms Handbook of Exact String-Matching Algorithms*. https://www.researchgate.net/publication/220693416

Filcha, A., & Hayaty, M. (t.t.). *Implementasi Algoritma Rabin-Karp untuk Pendeteksi Plagiarisme pada Dokumen Tugas Mahasiswa (Rabin-Karp Algorithm Implementation to Detect Plagiarism on Student's Assignment Document): Vol. VII*.

Ginting, A. A. B., & Utomo, D. P. (2019). PERANCANGAN APLIKASI CATALOG WISATA DI SUMATERA UTARA MENGGUNAKAN ALGORITMA RABIN-KARP. *KOMIK (Konferensi Nasional Teknologi Informasi Dan Komputer)*, *3*(1). https://doi.org/10.30865/komik.v3i1.1568

Hidayat, W., Utami, E., & Sunyoto, A. (2022). Selection of the Best K-Gram Value on Modified Rabin-Karp Algorithm. *IJCCS (Indonesian Journal of Computing and Cybernetics Systems)*, *16*(1), 11. https://doi.org/10.22146/ijccs.63686

IMPLEMENTASI DETEKSI SIMILARITAS KODE PADA SISTEM PRAKTIKUM PEMROGRAMAN WEB BERBASIS UNIT TESTING JAVA SCRIPT. (t.t.).

Jonsson, M., Qvarnström, E., Lindell, R., & Gustafsson, J. (t.t.). *A PERFORMANCE COMPARISON ON REST-APIS IN EXPRESS.JS, FLASK AND ASP.NET CORE*.

Karisma Wibowo, R., & Hastuti, K. (2016). *PENERAPAN ALGORITMA WINNOWING UNTUK MENDETEKSI KEMIRIPAN TEKS PADA TUGAS AKHIR MAHASISWA* (Vol. 15, Issue 4).

Keputusan Dirjen Penguatan Riset dan Pengembangan Ristek Dikti, S., Pramusinto, W., Waluyo, S., Informatika, T., Teknologi Informasi, F., & Budi Luhur, U. (2017). Terakreditasi SINTA Peringkat 2 Pengamanan Restful API menggunakan JWT untuk Aplikasi Sales Order. *Masa Berlaku Mulai*, *1*(3), 106–112.

Maratkar, P. S., & Adkar, P. (2021). *React JS-An Emerging Frontend Javascript Library*. https://nodejs.org/en/download/

Panjaitan, J., & Pakpahan, A. F. (2021). Perancangan Sistem E-Reporting Menggunakan ReactJS dan Firebase. *Jurnal Teknik Informatika Dan Sistem Informasi*, *7*(1). https://doi.org/10.28932/jutisi.v7i1.3098

Penidas, S., & Magist, F. (t.t.). *RESTFul Web*.

Steveson, D., Agung, H., & Mulia, F. (2018). *Plagiarisme Detection Applications For Tasks and Problems in School Using Rabin Karp Algorithm: Vol. I* (Issue 1). Th. http://journal.ubm.ac.id/jalu

Yulianto, M. A., & Nurhasanah, N. (2021). The Hybrid of Jaro-Winkler and Rabin-Karp Algorithm in Detecting Indonesian Text Similarity. *Jurnal Online Informatika*, *6*(1), 88. https://doi.org/10.15575/join.v6i1.640

DAFTAR LAMPIRAN

Lampiran 1. Auth Controller

```
const User = require('../users/model');
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');
const {
  JWT_SIGNATURE_KEY =
'$2b$10$o/Um9jZOMZxohkDcSpmd7.2DPVB1GLYOo04rQdC3tc81WT2jzY
Tem',
} = process.env;

function isPasswordValid(password, encryptedPassword) {
  return bcrypt.compareSync(password, encryptedPassword);
}

function createToken(payload) {
  return jwt.sign(payload, JWT_SIGNATURE_KEY, {
    expiresIn: '24h',
  });
}

module.exports = {
  actionRegister: async (req, res) => {
    try {
      const { name, password, role, registrationNumber, email } = req.body;

      const encryptedPassword = bcrypt.hashSync(password, 10);

      const regNumberExist = await User.findOne({
        registrationNumber: registrationNumber,
      });

      if (regNumberExist) {
        return res.status(400).json({
          data: {
            status: 'FAIL',
            message: 'Registration number is already registered',
          },
        });
      }

      const user = await User.create({
        name,
        encryptedPassword,
```

```
      role,
      registrationNumber,
      email,
    });

    return res.status(201).json({
      status: 'OK',
      message: 'You are successfully registered',
      data: {
        id: user.id,
        name: user.name,
        registrationNumber: user.registrationNumber,
        email: user.email,
        role: user.role,
      },
    });
  } catch (err) {
    res.send({ message: err.message || 'Internal Server Errror' });
  }
},

actionLogin: async (req, res) => {
  try {
    const { username, password } = req.body;

    const user = await User.findOne({ registrationNumber: username });

    if (!user) {
      return res.status(401).json({
        status: 'FAIL',
        data: {
          name: 'UNAUTHORIZED',
          message: 'Username does not exist',
        },
      });
    }

    if (!isPasswordValid(password, user.encryptedPassword)) {
      return res.status(401).json({
        status: 'FAIL',
        data: {
          name: 'UNAUTHORIZED',
          message: 'Wrong password',
        },
      });
    }
```

```
    return res.status(201).json({
      status: 'OK',
      data: {
        token: createToken({
          id: user.id,
          name: user.name,
          username: user.registrationNumber,
          role: user.role,
          email: user.email,
        }),
      },
    });
  } catch (error) {
    res.send({ message: err.message || 'Internal Server Errror' });
  }
 },
};
```

Lampiran 2. Auth Router

Lampiran 2. Auth Router

```
const express = require('express');
const router = express.Router();

const { actionRegister, actionLogin } = require('./controller');
const isRole = require('../../middlewares/isRole');
const isAuth = require('../../middlewares/isAuth');

router.post('/register', actionRegister);
router.post('/login', actionLogin);
module.exports = router;
```

Lampiran 3. Class Controller

```
const Class = require('./model');
const User = require('../users/model');
const Work = require('../works/model');
const Code = require('../code/model');
const Course = require('../courses/model');
const mongoose = require('mongoose');
const fs = require('fs');
```

```
module.exports = {
  createClass: async (req, res) => {
    try {
      const { name, author } = req.body;

      const course = await Course.findById({ _id: req.params.courseId });

      const classCourse = await Class.create({
        name,
        author: author,
        courseId: req.params.courseId,
      });

      await classCourse.save();

      classCourse.author.forEach(async (v) => {
        const user = await User.findOne({ _id: v._id });
        user.classes.push(classCourse);
        await user.save();
      });

      course.classes.push(classCourse);

      const filteredAuthor = author.filter((v) => v !== req.user.id);

      // check if Author is not user logged in.
      if (filteredAuthor.length > 0) {
        course.author.push(...filteredAuthor);
      }
      await course.save();

      return res.status(200).json({
        status: 'OK',
        message: 'Successfully created class',
        data: classCourse,
      });
    } catch (error) {
      console.log(error);
    }
  },

  editClass: async (req, res) => {
    const { name, author } = req.body;

    const classCourse = await Class.findOneAndUpdate(
```

```
    { _id: req.params.id },
    { name, author }
  );

  const user = await User.findOne({ _id: req.user.id }).select({
    encryptedPassword: 0,
    courses: 0,
  });

  if (!classCourse)
    return res.status(404).json({ status: 'Fail', message: 'Not found' });

  return res.status(200).json({
    status: 'OK',
    message: 'Your updated sucessfully',
    data: { name: name, author: author },
  });
},

deleteClass: async (req, res) => {
  const getClass = await Class.findOne({ _id: req.params.id }).populate(
    'author'
  );

  const classCourse = await Class.findOneAndDelete({ _id: req.params.id });

  const isAuthor = getClass.author.some(
    (e) => e._id.toString() === req.user.id
  );

  if (!isAuthor)
    return res.status(403).json({
      status: 'FORBIDDEN',
      message: 'You are not the author of this class',
    });

  if (!classCourse)
    return res.status(404).json({ status: 'Fail', message: 'Not found' });

  return res.status(200).json({
    status: 'OK',
    message: 'Delete sucessfully',
  });
},

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

```
    const classCourse = await Class.find({
      courseId: req.params.courseId,
      author: req.user.id,
    }).populate('author works');

    return res.status(200).json({
      status: 'OK',
      data: classCourse,
    });
  },

  getListWorkOfClass: async (req, res) => {
    try {
      const classCourse = await Class.findOne({
        _id: req.params.id,
      }).populate('author works');

      const code = await Code.find({
        author: req.user.id,
        classId: req.params.id,
      }).populate('workId');

      const codeTeacher = await Code.find({
        classId: req.params.id,
      }).populate({ path: 'workId', populate: { path: 'code' } });

      classCourse.works.forEach(async (v) => {
        const todayTimestamp = parseInt(
          (new Date().getTime() / 1000).toFixed(0)
        );

        if (v.deadline < todayTimestamp && v.status !== 'Finished') {
          await Work.findOneAndUpdate(
            {
              _id: v._id,
            },
            { status: 'Ready to review' }
          );

          fs.rm(
            `./Programs/Work/${req.params.id}`,
            {
              recursive: true,
              force: true,
            },
            (err) => {
```

```
            if (err) {
              console.log(err.message);
            }
          }
        );
      }
    });

    if (req.user.role === 'teacher') {
      return res.status(200).json({
        status: 'OK',
        data: classCourse,
      });
    }

    return res.status(200).json({
      status: 'OK',
      data: {
        name: classCourse.name,
        author: classCourse.author[0].name,
        works: code,
      },
    });
  } catch (error) {
    console.log(error);
  }
},

getListClass: async (req, res) => {
  const classCourse = await Class.find().populate(
    'students author works',
    '-__v -encryptedPassword'
  );

  return res.status(200).json({
    status: 'OK',
    data: classCourse,
  });
},

joinClass: async (req, res) => {
  const classCourse = await Class.findOne({ _id: req.params.id }).populate(
    'students works'
  );

  const user = await User.findOne({ _id: req.user.id });
```

```
const isExist = classCourse?.students.some(
  (element) => element.registrationNumber === user.registrationNumber
);

const work = await Work.find({
  classId: req.params.id,
  courseId: classCourse.courseId,
}).populate('code');

if (isExist)
  return res.status(400).json({
    status: '400',
    message: 'You are already join this class',
  });

if (!classCourse)
  return res
    .status(404)
    .json({ status: 'Fail', message: 'Your class is not available' });

classCourse.works.forEach(async (v, index) => {
  const codes = await Code.create({
    htmlCode: '',
    cssCode: '',
    jsCode: '',
    author: req.user.id,
    status: 'Not Completed',
    workId: v._id,
    classId: req.params.id,
    courseId: classCourse.courseId,
  });

  const folderName = `./Programs/Work/${work[index]._id}`;
  const folderStudent = `${folderName}/${user.registrationNumber}`;
  try {
    if (!fs.existsSync(folderStudent)) {
      fs.mkdirSync(folderStudent);
    }
  } catch (error) {
    console.log(error);
  }

  v.code.push(codes);
  work[index].code.push(codes);
  await work[index].save();
```

```
    });

    classCourse.students.push(user);
    user.classes.push(classCourse);

    await user.save();
    await classCourse.save();

    return res.status(200).json({
      status: 'OK',
      message: `Successfully join ${classCourse.name} class`,
      data: classCourse,
    });
  },

  getMyClass: async (req, res) => {
    try {
      const user = await User.findOne({ _id: req.user.id })
        .populate({
          path: 'classes',
          populate: [
            {
              path: 'author',
              model: 'User',
              select: {
                name: 1,
              },
            },
            {
              path: 'works',
              model: 'Work',
              select: {
                name: 1,
              },
            },
          ],
        })
        .select({
          encryptedPassword: 0,
        });

      return res.status(200).json({
        status: 'OK',
        data: user,
      });
    } catch (error) {
```

```
      console.log(error);
    }
  },
};
```

Lampiran 4. Class Model

```
const mongoose = require('mongoose');
const classSchema = mongoose.Schema({
  name: { type: String },
  author: [{ type: mongoose.Schema.Types.ObjectId, ref: 'User' }],
  courseId: { type: mongoose.Schema.Types.ObjectId, ref: 'Course' },
  students: [{ type: mongoose.Schema.Types.ObjectId, ref: 'User' }],
  works: [{ type: mongoose.Schema.Types.ObjectId, ref: 'Work' }],
});

module.exports = mongoose.model('Class', classSchema);
```

Lampiran 5. Class Router

```
const express = require('express');
const isAuth = require('../../middlewares/isAuth');
const isRole = require('../../middlewares/isRole');
const {
  createClass,
  getClassByCourseId,
  getListWorkOfClass,
  getListClass,
  joinClass,
  getMyClass,
  editClass,
  deleteClass,
} = require('./controller');
const router = express.Router();

router.post('/class/:courseId', isAuth, isRole(['teacher']), createClass);
router.get(
  '/class/:courseId',
  isAuth,
```

```
  isRole(['teacher', 'student']),
  getClassByCourseId
);
router.get(
  '/class/:id/works',
  isAuth,
  isRole(['teacher', 'student']),
  getListWorkOfClass
);
router.put('/class/:id', isAuth, isRole(['teacher']), editClass);
router.delete('/class/:id', isAuth, isRole(['teacher']), deleteClass);

router.get('/class', getListClass);
router.get('/my-class', isAuth, isRole(['teacher', 'student']), getMyClass);
router.post('/join-class/:id', isAuth, isRole(['student']), joinClass);

module.exports = router;
```

Lampiran 6. Code Controller

```
const Course = require('../courses/model');
const fs = require('fs');

const Work = require('../works/model');
const User = require('../users/model');
const Code = require('./model');
const RabinKarpJs = require('../../helper/RabinKarp');
const JaroWrinker = require('../../helper/JaroWrinker');
const analyzeCode = require('../../helper/esprima');
const { default: mongoose } = require('mongoose');

module.exports = {
  submitWork: async (req, res) => {
    const { htmlCode, cssCode, jsCode } = req.body;
    try {
      const user = await User.findOne({ _id: req.user.id });

      const work = await Work.findOne({ _id: req.params.id });

      const todayTimestamp = parseInt((new Date().getTime() /
1000).toFixed(0));

      if (todayTimestamp > work.deadline) {
        // await Work.findOneAndUpdate(
```

```
    //   { _id: req.params.id },
    //   { status: 'Ready to review' }
    // );

    return res.status(401).json({
      status: 'Fail',
      message:
        "Deadline is already end. You can't submit this work anymore.",
    });
  }

  if (todayTimestamp < work.deadline) {
    await Code.findOneAndUpdate(
      { author: req.user.id, workId: req.params.id },
      {
        cssCode: cssCode,
        htmlCode: htmlCode,
        jsCode: jsCode,
        status: 'Completed',
      }
    );
  }

  const codeTeacher = await Code.find({
    workId: req.params.id,
  }).populate({ path: 'workId', populate: { path: 'code' } });

  const getStatus = codeTeacher.map((v) => v.status);

  if (!getStatus.includes('Not Completed')) {
    await Work.findOneAndUpdate(
      { _id: req.params.id },
      { status: 'Ready to review' }
    );
    fs.rm(
      `./Programs/Work/${req.params.id}`,
      { recursive: true, force: true },
      (err) => {
        if (err) {
          console.log(err.message);
        }
      }
    );
  }
```

```
    const folderName =
`./Programs/Work/${req.params.id}/${user.registrationNumber}/program_test.
js`;
    const folderName1 =
`./Programs/Work/${req.params.id}/${user.registrationNumber}/program.js`;

    fs.rm(folderName, { recursive: true, force: true }, (err) => {
      if (err) {
        console.log(err.message);
      }
    });
    fs.rm(folderName1, { recursive: true, force: true }, (err) => {
      if (err) {
        console.log(err.message);
      }
    });

    return res.status(200).json({
      status: 'OK',
      message: 'Successfully submit work',
    });
  } catch (error) {
    return res.status(500).json({
      error: error,
    });
  }
},

testWork: async (req, res) => {
  try {
    let jsTest = `
  const chai = require("chai");
  const chaiDom = require("chai-dom");
  const resnap = require("resnap")();
  const JSDOM = require('jsdom').JSDOM;
  const axios = require("axios");
  chai.use(chaiDom);
  `;
    const clearCache = require('resnap')();
    // -------- requirements for js testing ---------
    const MochaTester = require('../../helper/MochaTester');
    const user = await User.findOne({ _id: req.user.id });

    await Work.findOne({ _id: req.params.id }, (err, doc) => {
      const js = req.body.jsCode;
      const html = req.body.htmlCode;
```

```
      jsTest += `
         const document = new JSDOM(\`${html}\`,{runScripts: "dangerously"
}).window.document;
         ${js}
         ${doc.codeTest}
        `;
      fs.writeFileSync(

`./Programs/Work/${doc._id}/${user.registrationNumber}/program_test.js`,
        jsTest
      );
      fs.writeFile(
        `./Programs/Work/${doc._id}/${user.registrationNumber}/program.js`,
        js,
        () => {
          MochaTester(

`./Programs/Work/${doc._id}/${user.registrationNumber}/program_test.js`
          )
            .then((pass) => {
              let testedJsCode = pass.results.every((test) => test);
              clearCache();
              return res.status(200).json({
                status: 'OK',
                data: pass,
                solution: testedJsCode,
              });
            })
            .catch((err) => {
              clearCache();
              return res.status(200).json({
                solution: false,
                data: {
                  error_msg: [err.message],
                },
              });
            });
        }
      );
    })
      .clone()
      .catch((err) => console.log(err));
  } catch (error) {
    console.log(error);
  }
```

```
},

getCode: async (req, res) => {
  const code = await Code.find();

  return res.status(200).json({ code });
},

checkSimilarity: async (req, res) => {
  try {
    const { algorithm, categoryClass } = req.body;
    const code = await Code.find({
      workId: req.params.id,
      status: 'Completed',
    }).populate('author workId');
    const ObjectId = mongoose.Types.ObjectId;

    if (categoryClass === 'all-class') {
      const work = await Work.findOne({
        _id: req.params.id,
      });

      // console.log('ini jalan yaw', work);

      const workSameExId = await Work.find({
        exerciseId: work.exerciseId,
      }).populate({
        path: 'code',
        populate: { path: 'workId', path: 'author' },
      });

      const worksCode = workSameExId.map((v) => v.code);
      const mergeWorksCode = worksCode.reduce((prev, next) =>
        prev.concat(next)
      );

      // console.log(mergeWorksCode, 'ini work same');

      mergeWorksCode.forEach(async (v, index) => {
        const similarityPercentage = [];
        const similarityResult = [];
        let esprimaCodeStudentA = '';
        mergeWorksCode.forEach(async (y) => {
          if (v.author.name !== y.author.name) {
            if (algorithm === 'RabinKarp') {
              const { resultSimilarity, esprimaCodeB, esprimaCodeA } =
```

```
            RabinKarpJs(v.jsCode, y.jsCode);
          esprimaCodeStudentA = esprimaCodeA;
          similarityPercentage.push(resultSimilarity);
          similarityResult.push({
            name: y.author.name,
            percentage: resultSimilarity,
            esprimaCode: esprimaCodeB,
            jsCode: y.jsCode,
          });
        }
        if (algorithm === 'JaroWinkler') {
          const studentA = analyzeCode(v.jsCode);
          const studentB = analyzeCode(y.jsCode);
          const resultSimilarity = (
            JaroWrinker(studentA, studentB) * 100
          ).toFixed(2);
          esprimaCodeStudentA = studentA;
          similarityPercentage.push(resultSimilarity);
          similarityResult.push({
            name: y.author.name,
            percentage: resultSimilarity,
            esprimaCode: studentB,
            jsCode: y.jsCode,
          });
        }
      }
    });
    await Code.findOneAndUpdate(
      { author: v.author._id, workId: v.workId },
      {
        esprimaCode: esprimaCodeStudentA,
        highestPercentage: `${Math.max(...similarityPercentage)}`,
        similarityResult: similarityResult,
      }
    );
  });

  workSameExId.forEach(async (v) => {
    const workStatus = await Work.findOneAndUpdate(
      { _id: v._id },
      { status: 'Finished', algorithmSimilarity: algorithm }
    );
    await workStatus.save();
  });

  return res.status(200).json({
```

```
      status: 'OK',
      message: `Sucessfully check al the similarity of this assignment`,
    });
  }

  if (categoryClass === 'one-class') {
    code.forEach(async (v, index) => {
      const similarityPercentage = [];
      const similarityResult = [];
      let esprimaCodeStudentA = '';
      code.forEach(async (y) => {
        if (v.author.name !== y.author.name) {
          if (algorithm === 'RabinKarp') {
            const { resultSimilarity, esprimaCodeB, esprimaCodeA } =
              RabinKarpJs(v.jsCode, y.jsCode);
            esprimaCodeStudentA = esprimaCodeA;
            similarityPercentage.push(resultSimilarity);
            similarityResult.push({
              name: y.author.name,
              percentage: resultSimilarity,
              esprimaCode: esprimaCodeB,
              jsCode: y.jsCode,
            });
          }
          if (algorithm === 'JaroWinkler') {
            const studentA = analyzeCode(v.jsCode);
            const studentB = analyzeCode(y.jsCode);
            const resultSimilarity = (
              JaroWrinker(studentA, studentB) * 100
            ).toFixed(2);
            esprimaCodeStudentA = studentA;
            similarityPercentage.push(resultSimilarity);
            similarityResult.push({
              name: y.author.name,
              percentage: resultSimilarity,
              esprimaCode: studentB,
              jsCode: y.jsCode,
            });
          }
        }
      });
      await Work.findOneAndUpdate(
        { _id: req.params.id },
        { status: 'Finished', algorithmSimilarity: algorithm }
      );
      await Code.findOneAndUpdate(
```

```
          { author: v.author._id, workId: req.params.id },
          {
           esprimaCode: esprimaCodeStudentA,
           highestPercentage: `${Math.max(...similarityPercentage)}`,
           similarityResult: similarityResult,
          }
         );
        });
        return await res.status(200).json({
         status: 'OK',
         message: `Sucessfully check the similarity of this assignment`,
        });
       }
      } catch (error) {
       console.log(error);
      }
    },

    detailStudentCode: async (req, res) => {
      const code = await Code.find({
       author: req.params.studentId,
       workId: req.params.workId,
      }).populate('author', 'name');

      return res.status(200).json({
       status: 'OK',
       data: code,
      });
    },

    updateScore: async (req, res) => {
      const { score } = req.body;
      const code = await Code.findOneAndUpdate(
       { _id: req.params.id },
       { score: score }
      );

      if (!code)
       return res.status(404).json({
        status: 'FAIL',
        message: 'Not found',
       });

      return res
       .status(200)
       .json({ status: 'OK', message: 'Update score sucessfully' });
```

```
  },
};
```

Lampiran 7. Code Model

```
const mongoose = require('mongoose');
const codeSchema = mongoose.Schema(
  {
   jsCode: {
     type: String,
     default: '',
   },
   htmlCode: {
     type: String,
     default: '',
   },
   cssCode: {
     type: String,
     default: '',
   },
   author: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },
   workId: { type: mongoose.Schema.Types.ObjectId, ref: 'Work' },
   status: {
     type: String,
     required: [true, 'Status tidak boleh kosong'],
     default: 'Not Completed',
   },
   highestPercentage: {
     type: String,
     default: '',
   },
   esprimaCode: { type: String, default: '' },
   similarityResult: [],
   classId: { type: mongoose.Schema.Types.ObjectId, ref: 'Class' },
   courseId: { type: mongoose.Schema.Types.ObjectId, ref: 'Course' },
   score: { type: String, default: '0' },
  },
  { timestamps: true }
);

module.exports = mongoose.model('Code', codeSchema);
```

Lampiran 8. Code Router

```
const express = require('express');
const isAuth = require('../../middlewares/isAuth');
const isRole = require('../../middlewares/isRole');
const {
  submitWork,
  testWork,
  getCode,
  checkSimilarity,
  detailStudentCode,
  updateScore,
} = require('./controller');
const router = express.Router();

router.post('/works/:id/submit-work', isAuth, isRole(['student']), submitWork);
router.post('/works/:id/test-work', isAuth, isRole(['student']), testWork);
router.get('/code-aja', getCode);
router.post(
  '/check-similarity/:id',
  isAuth,
  isRole(['teacher']),
  checkSimilarity
);

router.get(
  '/detail-work/:workId/student/:studentId',
  isAuth,
  isRole(['teacher', 'student']),
  detailStudentCode
);
router.put('/works/code/:id/score', isAuth, isRole(['teacher']), updateScore);

module.exports = router;
```

Lampiran 9. Course Controller

```
const Course = require('./model');
const User = require('../users/model');
const Code = require('../code/model');
const Work = require('../works/model');
const Class = require('../class/model');

module.exports = {
  createCourse: async (req, res) => {
```

```
  const { name } = req.body;

  const user = await User.findOne({ _id: req.user.id }).select({
    encryptedPassword: 0,
    courses: 0,
  });

  const course = await Course({ name });
  course.author.push(user);
  await course.save();

  return res.status(200).json({
    status: 'OK',
    message: 'Successfully created course',
    data: { name: course.name, author: user },
  });
},

deleteCourse: async (req, res) => {
  const getCourse = await Course.findOne({ _id: req.params.id }).populate(
    'author'
  );
  const course = await Course.findOneAndDelete({ _id: req.params.id });

  const classCourse = await Class.find({ courseId: req.params.id });

  classCourse.forEach(async (v) => {
    await Class.findOneAndDelete({ _id: v._id });
  });

  const isAuthor = getCourse.author.some(
    (e) => e._id.toString() === req.user.id
  );

  if (!isAuthor)
    return res.status(403).json({
      status: 'FORBIDDEN',
      message: 'You are not the author of this coursedd',
    });

  if (!course)
    return res.status(404).json({ status: 'Fail', message: 'Not found' });

  return res.status(200).json({
    status: 'OK',
    message: 'Delete sucessfully',
```

```
  });
},

editCourse: async (req, res) => {
  const { name } = req.body;

  const course = await Course.findOneAndUpdate(
    { _id: req.params.id },
    { name }
  );

  const user = await User.findOne({ _id: req.user.id }).select({
    encryptedPassword: 0,
    courses: 0,
  });

  const isAuthor = await Course.findOne({ author: { _id: req.user.id } });

  if (!isAuthor)
    return res.status(403).json({
      status: 'FORBIDDEN',
      message: 'You are not the author of this course',
    });

  if (!course)
    return res.status(404).json({ status: 'Fail', message: 'Not found' });

  return res.status(200).json({
    status: 'OK',
    message: 'Your updated sucessfully',
    data: { name: name, author: user },
  });
},

joinCourse: async (req, res) => {
  const course = await Course.findOne({ _id: req.params.id }).populate(
    'students works'
  );

  const user = await User.findOne({ _id: req.user.id });

  const isExist = course?.students.some(
    (element) => element.registrationNumber === user.registrationNumber
  );

  if (isExist)
```

```
      return res.status(400).json({
        status: '400',
        message: 'You are already join this class',
      });

    course.works.map(async (v) => {
      const codes = await Code({
        htmlCode: '',
        cssCode: '',
        jsCode: '',
        author: req.user.id,
        status: 'Not Completed',
        workId: v._id,
        courseId: course._id,
      });
      v.code.push(codes);
      await codes.save();
    });

    course.students.push(user);
    user.courses.push(course);
    user.save();
    course.save();

    if (!course)
      return res
        .status(404)
        .json({ status: 'Fail', message: 'Your course is not available' });

    return res.status(200).json({
      status: 'OK',
      message: `Successfully join ${course.name} course`,
      data: course,
    });
  },

  getMyCourse: async (req, res) => {
    const course = await Course.find({ author: { _id: req.user.id } }).populate(
      'students author classes',
      '-courses -encryptedPassword'
    );

    const user = await User.findOne({ _id: req.user.id })
      .populate({
        path: 'courses',
        select: {
```

```
        name: 1,
      },
      populate: [
        {
          path: 'author',
          model: 'User',
          select: {
            name: 1,
          },
        },
        {
          path: 'works',
          model: 'Work',
          select: {
            name: 1,
          },
        },
      ],
    })
    .select({
      encryptedPassword: 0,
      __v: 0,
    });

  if (user.role === 'teacher')
    return res.status(200).json({
      status: 'OK',
      data: course,
    });

  return res.status(200).json({
    status: 'OK',
    data: user,
  });
},

getCourseById: async (req, res) => {
  try {
    const { name } = req.body;

    const course = await Course.findOneAndUpdate(
      { _id: req.params.id },
      { name: name }
    );

    return res.status(200).json({
```

```
      status: 'OK',
      data: course,
    });
  } catch (error) {
    console.log(error);
  }
},

getCourseswithStudents: async (req, res) => {
  const course = await Course.find().populate(
    'students author works classes',
    '-__v -encryptedPassword'
  );

  return res.status(200).json({
    status: 'OK',
    data: course,
  });
},

getListWorkOfCourse: async (req, res) => {
  try {
    const course = await Course.findOne({ _id: req.params.id }).populate(
      'author works'
    );

    const code = await Code.find({
      author: req.user.id,
      courseId: req.params.id,
    }).populate('workId');
    //   const statusCode = code.map((v) => v.status);

    const codeTeacher = await Code.find({
      author: req.user.id,
      courseId: req.params.id,
    }).populate({ path: 'workId', populate: { path: 'code' } });

    const getStatus = codeTeacher.map((v) => v.status);
    if (!getStatus.includes('Not Completed')) {
      await Work.findOneAndUpdate(
        { _id: req.params.id },
        { status: 'Ready to review' }
      );
    }

    if (req.user.role === 'teacher') {
```

```
        return res.status(200).json({
          status: 'OK',
          data: course,
        });
      }

      return res.status(200).json({
        status: 'OK',
        data: {
          name: course.name,
          author: course.author[0].name,
          works: code,
        },
      });
    } catch (error) {
      console.log(error.message);
    }
  },
};
```

Lampiran 10. Course Model

```
const mongoose = require('mongoose');

const coursesSchema = mongoose.Schema({
  name: {
    type: String,
    required: [true, 'Nama tidak boleh kosong'],
  },
  author: [{ type: mongoose.Schema.Types.ObjectId, ref: 'User' }],
  students: [{ type: mongoose.Schema.Types.ObjectId, ref: 'User' }],
  classes: [{ type: mongoose.Schema.Types.ObjectId, ref: 'Class' }],
  works: [{ type: mongoose.Schema.Types.ObjectId, ref: 'Work' }],
});

module.exports = mongoose.model('Course', coursesSchema);
```

Lampiran 11. Course Router

```
var express = require('express');
const isAuth = require('../../middlewares/isAuth');
const isRole = require('../../middlewares/isRole');
```

```
var router = express.Router();
const {
  createCourse,
  joinCourse,
  getCourseswithStudents,
  deleteCourse,
  editCourse,
  getMyCourse,
  getListWorkOfCourse,
  getCourseById,
} = require('./controller');

router.post('/courses', isAuth, isRole(['teacher']), createCourse);
router.post('/join-course/:id', isAuth, isRole(['student']), joinCourse);
router.get('/courses', getCourseswithStudents);
router.get('/courses/:id', getCourseById);
router.delete('/courses/:id', isAuth, isRole(['teacher']), deleteCourse);
router.put('/courses/:id', isAuth, isRole(['teacher']), editCourse);
router.get('/my-course', isAuth, isRole(['teacher', 'student']), getMyCourse);
router.get(
  '/courses/:id/works',
  isAuth,
  isRole(['teacher', 'student']),
  getListWorkOfCourse
);
module.exports = router;
```

Lampiran 12. User Controller

```
const User = require('./model');

module.exports = {
  getUsers: async (req, res) => {
    const user = await User.find();

    return res.status(200).json({
      status: 'OK',
      data: user,
    });
  },

  getTeacher: async (req, res) => {
    const teacher = await User.find({ role: 'teacher' }).select({
      encryptedPassword: 0,
        __v: 0,
```

```
    });

    return res.status(200).json({
      status: 'OK',
      data: teacher,
    });
  },

  myProfile: async (req, res) => {
    const user = await User.findOne({ _id: req.user.id }).select({
      encryptedPassword: 0,
      __v: 0,
    });

    if (!user)
      return res.status(404).json({
        status: 'Fail',
        message: 'Not Found',
      });

    return res.status(200).json({
      status: 'OK',
      data: user,
    });
  },
};
```

Lampiran 13. User Model

```
const mongoose = require('mongoose');

const userSchema = mongoose.Schema({
  name: {
    type: String,
    required: [true, 'Nama tidak boleh kosong'],
  },
  registrationNumber: {
    type: String,
    required: [true, 'NIM/NIP tidak boleh kosong'],
    unique: true,
  },
  email: {
    type: String,
    required: [true, 'Email tidak boleh kosong'],
    unique: true,
```

```
  },
  encryptedPassword: {
    type: String,
    required: [true, 'Password tidak boleh kosong'],
  },
  role: {
    type: String,
    default: 'student',
    enum: ['student', 'teacher', 'admin'],
  },
  courses: [{ type: mongoose.Schema.Types.ObjectId, ref: 'Course' }],
  classes: [{ type: mongoose.Schema.Types.ObjectId, ref: 'Class' }],
});

module.exports = mongoose.model('User', userSchema);
```

Lampiran 14. User Router

```
const express = require('express');
const isAuth = require('../../middlewares/isAuth');
const isRole = require('../../middlewares/isRole');
var router = express.Router();
const { getUsers, myProfile, getTeacher } = require('./controller');

router.get('/users', isAuth, isRole(['teacher']), getUsers);
router.get('/profile', isAuth, isRole(['teacher', 'student']), myProfile);
router.get('/user/teacher', isAuth, isRole(['teacher', 'student']), getTeacher);
module.exports = router;
```

Lampiran 15. Work Controller

```
const Course = require('../courses/model');
const Work = require('./model');
const Code = require('../code/model.js');
const Class = require('../class/model.js');
const { default: mongoose } = require('mongoose');
const User = require('../users/model');
const fs = require('fs');

module.exports = {
  createWork: async (req, res) => {
    try {
      const {
        name,
        description,
```

```
    codeTest,
    deadline,
    htmlStarter,
    cssStarter,
    jsStarter,
  } = req.body;

  const classCourse = await Class.findOne({
    _id: req.params.id,
  }).populate('author works students');

  const courseClass = await Class.find({
    courseId: classCourse.courseId,
    author: req.user.id,
  }).populate('students');

  const exerciseId = new mongoose.Types.ObjectId();
  courseClass.map(async (singleClass) => {
    const work = await Work({
      name,
      description,
      classId: singleClass._id,
      codeTest,
      exerciseId,
      deadline,
      htmlStarter,
      cssStarter,
      jsStarter,
      courseId: classCourse.courseId,
    });

    const folderName = `./Programs/Work/${work._id}`;
    try {
      if (!fs.existsSync(folderName)) {
        fs.mkdirSync(folderName);
      }
    } catch (error) {
      console.log(error);
    }

    if (singleClass.students.length > 0) {
      singleClass.students.map(async (student) => {
        const codes = await Code({
          htmlCode: '',
          cssCode: '',
          jsCode: '',
```

```
            author: student._id,
            status: 'Not Completed',
            workId: work._id,
            classId: singleClass._id,
            courseId: classCourse.courseId,
          });

          const folderStudent = `${folderName}/${student.registrationNumber}`;
          try {
            if (!fs.existsSync(folderStudent)) {
              fs.mkdirSync(folderStudent);
            }
          } catch (error) {
            console.log(error);
          }

          work.code.push(codes._id);
          await codes.save();
        });
      }

      singleClass.works.push(work._id);
      await singleClass.save();
      await work.save();
    });

    return res.status(200).json({
      status: 'OK',
      message: 'Successfully created work',
    });
  } catch (error) {
    console.log(error.message);
  }
},

editWork: async (req, res) => {
  try {
    const {
      name,
      description,
      codeTest,
      deadline,
      htmlStarter,
      cssStarter,
      jsStarter,
    } = req.body;
```

```
    const work = await Work.findOneAndUpdate(
      { _id: req.params.id },
      {
        name,
        description,
        codeTest,
        deadline,
        htmlStarter,
        cssStarter,
        jsStarter,
      }
    );

    return res.status(200).json({
      status: 'OK',
      message: 'Succesfully update work',
      data: work,
    });
  } catch (error) {
    console.log(error);
  }
},

deleteWork: async (req, res) => {
  try {
    const work = await Work.findOneAndDelete({ _id: req.params.id });

    const isAuthor = await Work.findOne({ author: { _id: req.user.id } });

    if (!isAuthor)
      return res.status(403).json({
        status: 'FORBIDDEN',
        message: 'You are not the author of this course',
      });

    if (!work)
      return res.status(404).json({ status: 'Fail', message: 'Not found' });

    return res.status(200).json({
      status: 'OK',
      message: 'Delete sucessfully',
    });
  } catch (error) {
    console.log(error);
  }
},
```

```
getWorkById: async (req, res) => {
  try {
    const work = await Work.findById({ _id: req.params.id }).populate({
      path: 'code',
      populate: {
        path: 'author',
        select: 'name',
      },
    });

    const codeTeacher = await Code.find({
      workId: req.params.id,
    }).populate({ path: 'workId', populate: { path: 'code' } });

    const getStatus = codeTeacher.map((v) => v.status);

    if (!getStatus.includes('Not Completed')) {
      await Work.findOneAndUpdate(
        { _id: req.params.id },
        { status: 'Ready to review' }
      );

      fs.rm(
        `./Programs/Work/${req.params.id}`,
        {
          recursive: true,
          force: true,
        },
        (err) => {
          if (err) {
            console.log(err.message);
          }
        }
      );
    }

    return res.status(200).json({
      status: 'OK',
      data: work,
    });
  } catch (error) {
    console.log(error.message);
  }
},
```

```
  changeVisibleWork: async (req, res) => {
    try {
      const work = await Work.findOne({ _id: req.params.id });

      await Work.findOneAndUpdate(
        { _id: req.params.id },
        { isVisible: !work.isVisible }
      );

      return res.status(200).json({
        status: 'OK',
        message: 'Succesfully change the visible of work',
      });
    } catch (error) {
      console.log(error);
    }
  },
};
```

Lampiran 16. Work Model

```
const mongoose = require('mongoose');

const workSchema = mongoose.Schema(
  {
    name: {
      type: String,
      required: [true, 'Nama tidak boleh kosong'],
    },
    description: {
      type: String,
      required: [true, 'Deskripsi tidak boleh kosong'],
    },
    codeTest: {
      type: String,
      required: [true, 'Code Test tidak boleh kosong'],
    },
    htmlStarter: { type: String },
    cssStarter: { type: String },
    jsStarter: { type: String },
    code: [{ type: mongoose.Schema.Types.ObjectId, ref: 'Code' }],
    status: {
      type: String,
      default: 'Not ready to review',
    },
    algorithmSimilarity: { type: String },
```

```
    deadline: { type: Number, required: [true, 'Deadline tidak boleh kosong'] },
    classId: { type: mongoose.Schema.Types.ObjectId, ref: 'Class' },
    exerciseId: { type: mongoose.Schema.Types.ObjectId },
    courseId: { type: mongoose.Schema.Types.ObjectId, ref: 'Class' },
    isVisible: { type: Boolean, default: true },
  },
  { timestamps: true }
);

module.exports = mongoose.model('Work', workSchema);
```

Lampiran 17. Work Router

```
var express = require('express');
const isAuth = require('../../middlewares/isAuth');
const isRole = require('../../middlewares/isRole');
const {
  createWork,
  getListWorkOfCourse,
  getWorkById,
  changeVisibleWork,
  editWork,
  deleteWork,
} = require('./controller');
var router = express.Router();

router.post('/courses/:id/works', isAuth, isRole(['teacher']), createWork);
router.get('/works/:id', isAuth, isRole(['teacher', 'student']), getWorkById);
router.post(
  '/works/:id/visible',
  isAuth,
  isRole(['teacher']),
  changeVisibleWork
);
router.put('/works/:id', isAuth, isRole(['teacher']), editWork);
router.delete('/works/:id', isAuth, isRole(['teacher']), deleteWork);

module.exports = router;
```

Lampiran 18. Config

```
const dotenv = require('dotenv');

dotenv.config();
```

```
module.exports = {
  serviceName: process.env.SERVICE_NAME,
  urlDb: process.env.MONGO_URL,
};
```

Lampiran 19. Database Config

```
const dotenv = require('dotenv');
const mongoose = require('mongoose');
const { urlDb } = require('../config');

mongoose.connect(urlDb);

const db = mongoose.connection;

module.exports = db;
```

Lampiran 20. Algoritma Rabin Karp

```
const analyzeCode = require('./esprima');

function RabinKarpJs(codeA, codeB) {
  const esprimaCodeA = analyzeCode(codeA);
  const esprimaCodeB = analyzeCode(codeB);

  const kGramA = kGram(esprimaCodeA);
  const kGramB = kGram(esprimaCodeB);

  const hashA = hashing(esprimaCodeA, 3, 5);
  const hashB = hashing(esprimaCodeB, 3, 5);

  const resultA = fingerPrint(hashA);
  const resultB = fingerPrint(hashB);

  const resultSimilarity = similarityCheck(resultA, resultB);
  return { resultSimilarity, esprimaCodeA, esprimaCodeB };
}

function similarityCheck(hashA, hashB) {
  let count = 0;
  const similarItems = [];
  for (let i = 0; i < hashA.length; i++) {
    for (let j = 0; j < hashB.length; j++) {
```

```
      if (hashA[i] === hashB[j]) {
        count++;
        similarItems.push(hashA[i]);
      }
    }
  }
  return Number(
    (((2 * count) / (hashA.length + hashB.length)) * 100).toFixed(2)
  );
}

function rollingHash(string, basis) {
  let hash = 0;
  for (let i = 0; i < string.length; i++) {
    let ascii = string.charCodeAt(i);
    hash += ascii * Math.pow(basis, string.length - (i + 1));
  }

  return hash;
}

function hashing(text, gram, basis) {
  const hashSplit = [];
  if (text.length < gram) {
    return text;
  }

  for (let i = 0; i <= text.length - gram; i++) {
    const textSplit = text.substr(i, gram);
    hashSplit.push(rollingHash(textSplit, basis));
  }
  return hashSplit;
}

function kGram(teks, gram) {
  const textSplit = [];

  if (teks.length < gram) {
    return teks;
  }
  for (let i = 0; i < teks.length; i++) {
    textSplit.push(teks.substr(i, gram));
  }
  return textSplit;
}
```

```
function fingerPrint(hash) {
  const finger = hash.filter((element, index) => {
    return hash.indexOf(element) === index;
  });

  return finger;
}

module.exports = RabinKarpJs;
```

Lampiran 21. Esprima

```
var esprima = require('esprima');

function traverse(node, func) {
  func(node); //1
  for (var key in node) {
    //2
    // console.log(key);
    if (node.hasOwnProperty(key)) {
      //3
      var child = node[key];
      if (typeof child === 'object' && child !== null) {
        //4

        if (Array.isArray(child)) {
          child.forEach(function (node) {
            //5
            traverse(node, func);
          });
        } else {
          traverse(child, func); //6
        }
      }
    }
  }
}
function analyzeCode(code) {
  var ast = esprima.parse(code);
  // console.log(esprima.parse(code));

  let temp = '';
  traverse(ast, function (node) {
    if (node.type === 'Program') node.type = 'A';
    if (node.type === 'AssignmentExpression') node.type = 'B';
```

```
    if (node.type === 'BinaryExpression') node.type = 'C';
    if (node.type === 'BlockStatement') node.type = 'D';
    if (node.type === 'CallExpression') node.type = 'E';
    if (node.type === 'ExpressionStatement') node.type = 'F';
    if (node.type === 'FunctionDeclaration') node.type = 'G';
    if (node.type === 'Identifier') node.type = 'H';
    if (node.type === 'Literal') node.type = 'I';
    if (node.type === 'MemberExpression') node.type = 'J';
    if (node.type === 'VariableDeclaration') node.type = 'K';
    if (node.type === 'VariableDeclarator') node.type = 'L';
    if (node.type === 'FunctionExpression') node.type = 'M';
    if (node.type === 'WhileStatement') node.type = 'N';
    if (node.type === 'IfStatement') node.type = 'O';
    if (node.type === 'LogicalExpression') node.type = 'P';
    if (node.type === 'ReturnStatement') node.type = 'Q';
    if (node.type === 'ArrayExpression') node.type = 'R';
    if (node.type === 'EmptyStatement') node.type = 'S';
    if (node.type === 'DoWhileStatement') node.type = 'T';
    if (node.type === 'ForStatement') node.type = 'U';
    if (node.type === 'UpdateExpression') node.type = 'V';
    if (node.type === 'UnaryExpression') node.type = 'W';
    if (node.type === 'BreakStatement') node.type = 'X';
    if (node.type === 'ArrowFunctionExpression') node.type = 'Y';
    if (node.type === 'TemplateLiteral') node.type = 'Z';
    if (node.type === 'TemplateElement') node.type = 'AB';
    if (node.type === undefined) node.type = 'UN';
    if (node.type === 'NewExpression') node.type = 'NE';
    if (node.type === 'ConditionalExpression') node.type = 'CE';
    if (node.type === 'ObjectExpression') node.type = 'OE';
    if (node.type === 'Property') node.type = 'PR';
    if (node.type === 'AwaitExpression') node.type = 'AE';

    temp += node.type;
  });
  return temp;
}

module.exports = analyzeCode;
```

DAFTAR PERBAIKAN

Muhammad Nur Faisi S – D121181503

Aplikasi Deteksi Similaritas Javascript Code Berbasis MERN Menggunakan Algoritma Rabin Karp

Tambahkan pseudocode algoritma Rabin Karp	BAB III Halaman 25
Perjelas urutan algoritma Rabin Karp	BAB III halaman 24 - 27

LEMBAR PERBAIKAN SKRIPSI

"APLIKASI DETEKSI SIMILARITAS JAVASCRIPT CODE BERBASIS MERN MENGGUNAKAN ALGORITMA RABIN KARP"

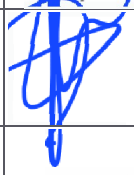
OLEH:

MUHAMMAD NUR FAISI S
D121181503

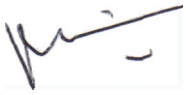
Skripsi ini telah dipertahankan pada Ujian Akhir Sarjana tanggal 17 Februari 2023.

Telah dilakukan perbaikan penulisan dan isi skripsi berdasarkan usulan dari penguji dan pembimbing skripsi.

Persetujuan perbaikan oleh tim penguji:

	Nama	Tanda Tangan
Ketua	Dr. Eng. Ir. Muhammad Niswar., S.T M.IT.	
Sekretaris	Iqra Aswad S.T M.T.	
Anggota	Dr. Amil Ahmad Ilham, S.T., M.IT.	
	Mukarramah Yusuf, B.Se., M.Sc.	

Persetujuan Perbaikan oleh pembimbing:

Pembimbing	Nama	Tanda Tangan
I	Dr. Eng. Ir. Muhammad Niswar., S.T M.IT.	
II	Iqra Aswad S.T M.T.