

LAPORAN KOMPONEN SIMULASI DARI LIFECYCLE SEBUAH API

Nama Kelompok:

1. Yogi Suryo Saputro(211111139)
2. Faiz Wildan (211111137)
3. Ainul Irsyad (211111144)

Langkah-langkah Pembuatan API

- a. Pastikan sudah menginstal node js
- b. Instalasi NPM dengan kode berikut:

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Hp\OneDrive\Dokumen\backend> npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See 'npm help init' for definitive documentation on these fields
and exactly what they do.

Use 'npm install <pkg>' afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (backend)
version: (1.0.0)
description:
entry point: (index.js)
test command:
git repository:
keywords:
author: |
```

- c. Lakukan instalasi package

```
PROBLEMS 10 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Hp\OneDrive\Dokumen\backend> npm i express sequelize mysql2 nodemon

added 1 package, changed 1 package, and audited 224 packages in 10s

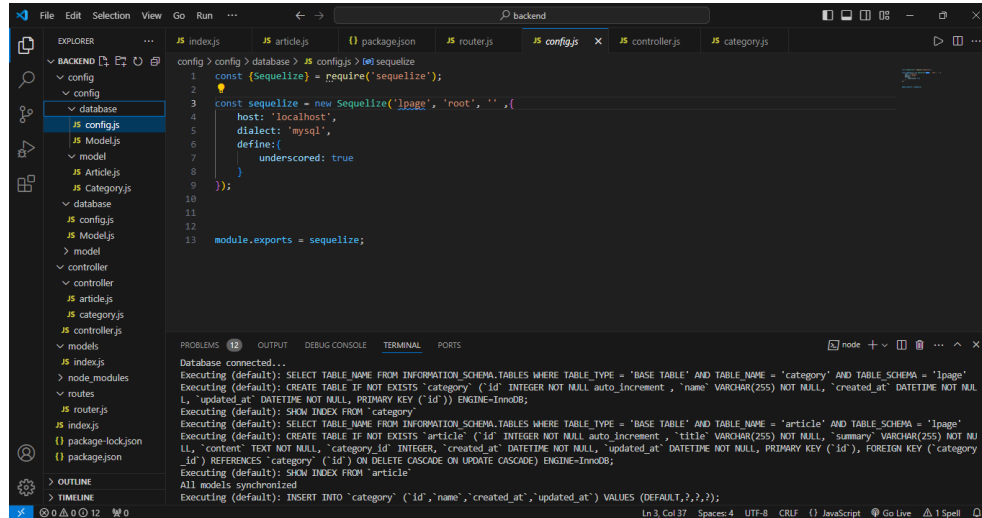
34 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
PS C:\Users\Hp\OneDrive\Dokumen\backend> |
```

d. Buat Kode Program

1. Mengatur dan menyiapkan koneksi ke database MySQL

Kode ini digunakan untuk mengatur koneksi ke database MySQL menggunakan Sequelize, yang merupakan ORM (Object-Relational Mapping) untuk Node.js.



```
config > config > database > JS config.js > @ sequelize
1 const {Sequelize} = require('sequelize');
2
3 const sequelize = new Sequelize('ipage', 'root', '', {
4   host: 'localhost',
5   dialect: 'mysql',
6   define: {
7     underscored: true
8   }
9 });
10
11
12
13 module.exports = sequelize;
```

Database connected...

Executing (default): SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_TYPE = 'BASE TABLE' AND TABLE_NAME = 'category' AND TABLE_SCHEMA = 'ipage'

Executing (default): CREATE TABLE IF NOT EXISTS 'category' ('id' INTEGER NOT NULL auto_increment, 'name' VARCHAR(255) NOT NULL, 'created_at' DATETIME NOT NULL, 'updated_at' DATETIME NOT NULL, PRIMARY KEY ('id')) ENGINE=InnoDB;

Executing (default): SHOW INDEX FROM 'category'

Executing (default): SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_TYPE = 'BASE TABLE' AND TABLE_NAME = 'article' AND TABLE_SCHEMA = 'ipage'

Executing (default): CREATE TABLE IF NOT EXISTS 'article' ('id' INTEGER NOT NULL auto_increment, 'title' VARCHAR(255) NOT NULL, 'summary' VARCHAR(255) NOT NULL, 'content' TEXT NOT NULL, 'category_id' INTEGER, 'created_at' DATETIME NOT NULL, 'updated_at' DATETIME NOT NULL, PRIMARY KEY ('id'), FOREIGN KEY ('category_id') REFERENCES 'category' ('id') ON DELETE CASCADE ON UPDATE CASCADE) ENGINE=InnoDB;

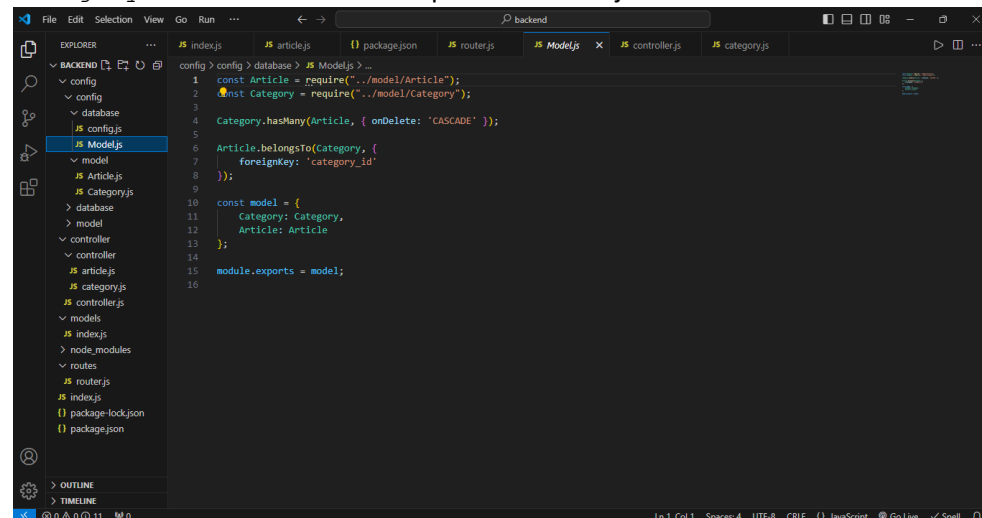
Executing (default): SHOW INDEX FROM 'article'

All models synchronized

Executing (default): INSERT INTO 'category' ('id','name','created_at','updated_at') VALUES (DEFAULT,?,?,?);

2. Menetapkan dan mengelola hubungan antara tabel dan Category di database

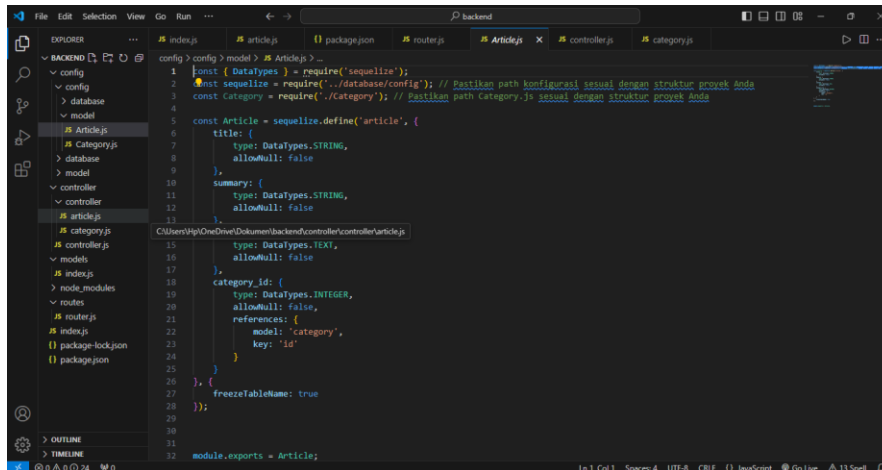
Kode ini digunakan untuk mendefinisikan hubungan antara model Article dan Category dalam konteks ORM Sequelize di Node.js



```
config > config > database > JS Model.js > ...
1 const Article = require("../model/Article");
2 const Category = require("../model/Category");
3
4 Category.hasMany(Article, { onDelete: 'CASCADE' });
5
6 Article.belongsTo(Category, {
7   foreignKey: 'category_id'
8 });
9
10 const model = {
11   Category: Category,
12   Article: Article
13 };
14
15 module.exports = model;
```

3. Mendefinisikan skema tabel Article di database

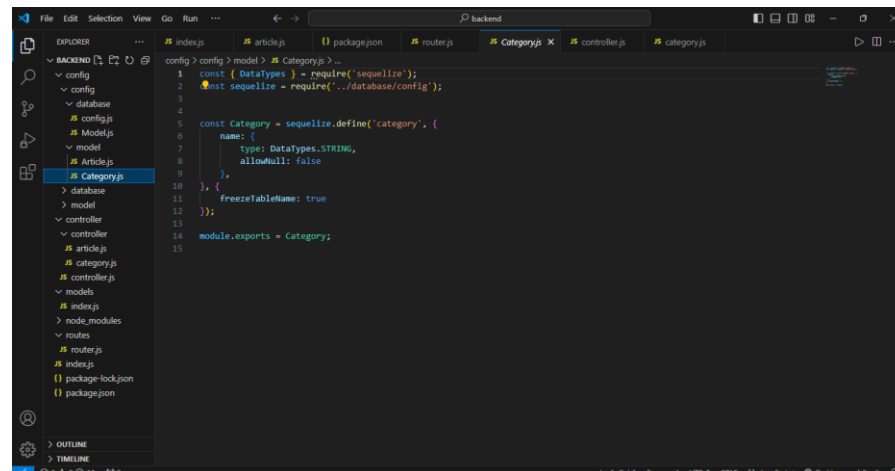
Kode ini digunakan untuk mendefinisikan model Article dalam ORM Sequelize di Node.js, serta mengatur skema dan hubungan tabel di database



```
1 const { DataTypes } = require('sequelize');
2 const sequelize = require('../database/config'); // Pastikan path konfigurasi sesuai dengan struktur proyek Anda
3 const Category = require('../Category'); // Pastikan path Category.js sesuai dengan struktur proyek Anda
4
5 const Article = sequelize.define('article', {
6   title: {
7     type: DataTypes.STRING,
8     allowNull: false
9   },
10   summary: {
11     type: DataTypes.STRING,
12     allowNull: false
13   },
14   category_id: {
15     type: DataTypes.INTEGER,
16     allowNull: false,
17     references: {
18       model: 'category',
19       key: 'id'
20     }
21   },
22   freezeTableName: true
23 });
24
25 module.exports = Article;
```

4. Mendefinisikan skema tabel Category di database

Kode ini digunakan untuk mendefinisikan model Category dalam ORM Sequelize di Node.js.



```
1 const { DataTypes } = require('sequelize');
2 const sequelize = require('../database/config');
3
4
5 const Category = sequelize.define('category', {
6   name: {
7     type: DataTypes.STRING,
8     allowNull: false
9   },
10   freezeTableName: true
11 });
12
13 module.exports = Category;
```

5. Mengelola operasi CRUD untuk artikel dalam aplikasi dengan menangani permintaan HTTP.

Kode ini adalah definisi dari controller untuk mengelola operasi CRUD (Create, Read, Update, Delete) pada artikel menggunakan Sequelize ORM di Node.js.

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

const controller = {};

controller.getArticle = async (req, res) => {
  try {
    const articles = await model.Article.findAll({
      include: model.Category
    });
    res.json(articles);
  }
}
```

```

    } catch (error) {
      console.error('Error fetching articles:', error);
      res.status(500).json({ error: 'Internal Server Error' });
    }
  });

controller.postArticle = async (req, res) => {
  try {
    const { title, summary, content, category_id } = req.body;
    if (!title || !summary || !content || !category_id) {
      return res.status(400).json({ error: 'Missing Input Data' });
    }
    const newArticle = await model.Article.create({
      title: title,
      summary: summary,
      content: content,
      category_id: category_id,
    },
    {
      include: model.Category
    });
    res.status(201).json({ message: 'Article created successfully', article:
newArticle });
  } catch (error) {
    console.error('Error creating article:', error);
    res.status(500).json({ error: 'Internal Server Error' });
  }
};

controller.getArticleById = async (req, res) => {
  try {
    const articleId = req.params.id;
    const article = await model.Article.findByPk(articleId, {
      include: model.Category
    });
    if (!article) {
      return res.status(404).json({ error: 'Article not found' });
    }
    res.json(article);
  } catch (error) {
    console.error('Error fetching article by id:', error);
    res.status(500).json({ error: 'Internal Server Error' });
  }
};

```

```

controller.deleteArticleById = async (req, res) => {
  const articleId = req.params.id;
  try {
    if (!articleId) res.status(404).json("Not Found")
    const category = await model.Article.destroy({
      where: {
        id: articleId
      },
      include: model.Category
    });
    if (!category) res.json("Delete Failure");
    res.status(200).json({ message: "Succes Delete Data" })
  } catch (error) {
    console.error('Error fetching category by id:', error);
    res.status(500).json({ error: 'Internal Server Error' });
  }
};

controller.editArticleById = async (req, res) => {
  const articleId = req.params.id;
  const { title, summary, content, category_id } = req.body;
  if (!title || !summary || !content || !category_id) {
    return res.status(400).json({ error: 'Missing Input Data' });
  }
  try {
    const [updated] = await model.Article.update(
      {
        title,
        summary,
        content,
        category_id
      },
      {
        where: { id: articleId },
        include: [model.Category]
      }
    );
    if (!updated) {
      return res.status(404).json({ error: "Update Data Failure" });
    }
    const updatedArticle = await model.Article.findByPk(articleId, {
      include: [model.Category]
    });
    return res.status(200).json({ message: "Update Success", article:
updatedArticle });
  }
};

```

```

    } catch (error) {
      console.error('Error updating article:', error);
      return res.status(500).json({ error: 'Internal Server Error' });
    }
  }
}

module.exports = controller;

```

6. Mengelola operasi CRUD untuk kategori dalam aplikasi dengan menangani permintaan HTTP.

Kode ini adalah definisi dari controller untuk mengelola operasi CRUD (Create, Read, Update, Delete) pada kategori menggunakan Sequelize ORM di Node.js.

```

const model = require('../../config/database/Model');

const controller = {};

controller.getCategory = async (req, res) => {
  try {
    const categories = await model.Category.findAll();
    res.json(categories);
  } catch (error) {
    console.error('Error fetching categories:', error);
    res.status(500).json({ error: 'Internal Server Error' });
  }
};

controller.postCategory = async (req, res) => {
  try {
    const { name } = req.body;
    if (!name) {
      return res.status(400).json({ error: 'Name is required' });
    }
    const newCategory = await model.Category.create({
      name: name
    });
    res.status(201).json({ message: 'Category created successfully',
category: newCategory });
  } catch (error) {
    console.error('Error creating category:', error);
    res.status(500).json({ error: 'Internal Server Error' });
  }
};

```

```

    }
  };

  controller.getCategoryById = async (req, res) => {
    try {
      const categoryId = req.params.id;
      const category = await model.Category.findPk(categoryId);
      if (!category) {
        return res.status(404).json({ error: 'Category not found' });
      }
      res.json(category);
    } catch (error) {
      console.error('Error fetching category by id:', error);
      res.status(500).json({ error: 'Internal Server Error' });
    }
  };

  controller.deleteCategoryById = async (req, res) => {
    const categoryId = req.params.id;
    try {
      if (!categoryId) return res.status(404).json("Not Found");

      const rowsDeleted = await model.Category.destroy({
        where: {
          id: categoryId
        },
      });
      if (rowsDeleted === 0) {
        return res.status(404).json({ message: "Delete Failure" });
      }
      res.status(200).json({ message: "Success Delete Data" });
    } catch (error) {
      console.error('Error deleting category by id:', error);
      res.status(500).json({ error: 'Internal Server Error' });
    }
  };

  controller.editCategoryById = async (req, res) => {
    const categoryId = req.params.id;
    const { name } = req.body
    try {
      const [numRowsUpdated, updatedCategories] = await model.Category.update(
        { name: name },
        { where: { id: categoryId } }
      );
    }
  };

```

```

        if (numRowsUpdated === 0) {
            return res.status(404).json({ message: 'Category not found or no
changes applied' });
        }

        res.status(200).json({ message: 'Update Success', category:
updatedCategories });
    } catch (error) {
        console.error('Error updating category by id:', error);
        res.status(500).json({ error: 'Internal Server Error' });
    }
};

module.exports = controller;

```

7. Menyederhanakan akses ke semua metode controller

Kode ini digunakan untuk menggabungkan semua metode controller untuk kategori dan artikel menjadi satu objek `controller` yang diekspor, sehingga dapat digunakan di seluruh aplikasi.

```

const articleController = require('./controller/article')
const categoryController = require('./controller/category')

const controller = {
    getCategory : categoryController.getCategory,
    getCategoryById : categoryController.getCategoryById,
    postCategory : categoryController.postCategory,
    editCategoryById : categoryController.editCategoryById,
    deleteCategoryById : categoryController.deleteCategoryById,

    getArticle : articleController.getArticle,
    getArticleById : articleController.getArticleById,
    postArticle : articleController.postArticle,
    editArticleById : articleController.editArticleById,
    deleteArticleById : articleController.deleteArticleById,
}

module.exports = controller

```


8. Mengatur dan mengarahkan permintaan HTTP untuk operasi CRUD

Kode ini digunakan untuk mengatur routing dalam aplikasi Express.js, mengarahkan permintaan HTTP ke metode controller yang sesuai untuk artikel dan kategori.

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

const controller = require('../controller/controller');

const router = express.Router();

// Article Router
router.get('/api/article', controller.getArticle);
router.get('/api/article/:id', controller.getArticleById);
router.post('/api/article', controller.postArticle);
router.patch('/api/article/:id', controller.editArticleById);
router.delete('/api/article/:id', controller.deleteArticleById);

// Category Router
router.get('/api/category', controller.getCategory);
router.get('/api/category/:id', controller.getCategoryById);
router.post('/api/category', controller.postCategory);
router.patch('/api/category/:id', controller.editCategoryById);
router.delete('/api/category/:id', controller.deleteCategoryById);

module.exports = router;
```

9.

- Mengatur dan menjalankan aplikasi Express.js yang terhubung dengan database.
- Memproses data JSON dari request.
- Mengatur rute aplikasi.
- Mensinkronkan model-model Sequelize dengan database.
- Memulai server pada port tertentu untuk menerima permintaan klien.

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

const sequelize = require('./config/database/config');
const router = require('./routes/router');

const app = express();

// Middleware untuk meng-handle JSON body dari request
app.use(express.json());
```

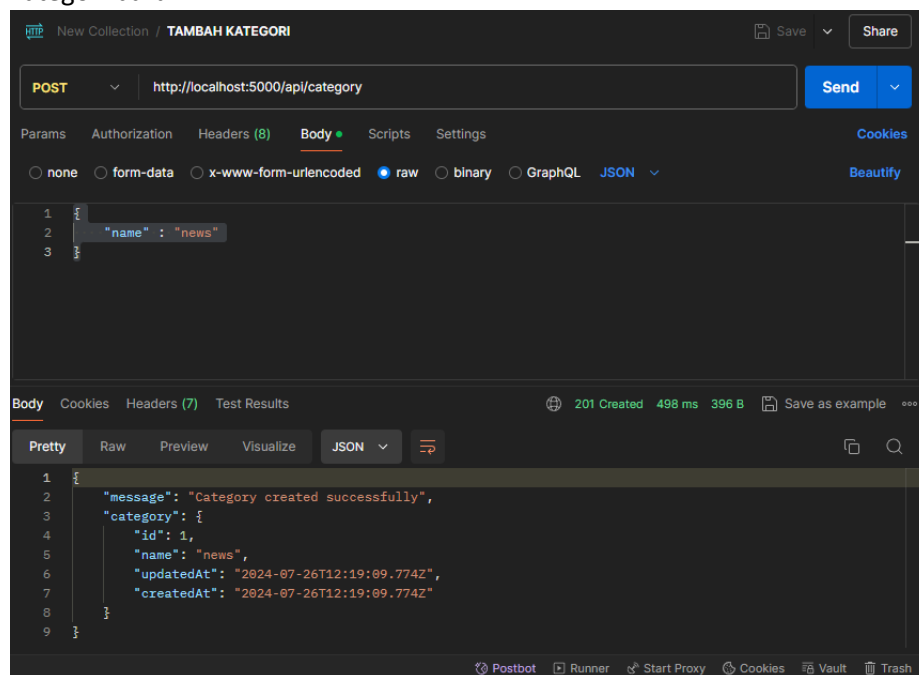
```
// Gunakan router sebagai middleware utama
app.use(router);

// Sambungkan ke database dan sinkronkan model-model
sequelize.authenticate()
  .then(() => {
    console.log('Database connected...');
    return sequelize.sync();
  })
  .then(() => {
    console.log('All models synchronized');
  })
  .catch((err) => {
    console.error('Unable to connect to the database:', err);
  });

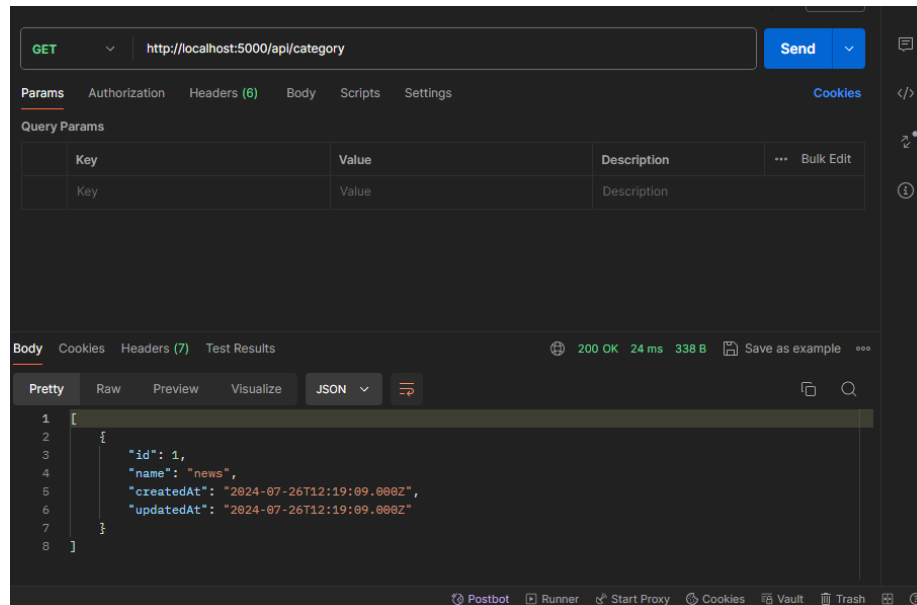
// Mulai aplikasi di port 5000
const PORT = 5000;
app.listen(PORT, () => {
  console.log(`App listening on port ${PORT}...`);
});
```

e. Langkah-langkah dalam postman

1. Menyediakan konfirmasi bahwa kategori baru telah berhasil dibuat di server.
Output dari Postman ini menunjukkan respons dari server setelah berhasil membuat kategori baru.



2. Output dari Postman ini menunjukkan data yang dikembalikan oleh server sebagai respons terhadap permintaan untuk mendapatkan daftar kategori



3. Output dari Postman ini menunjukkan respons server setelah berhasil memperbarui data.

