npm install --save-dev @babel/core @babel/preset-env
babel-jest jest sqlite3 superagent supertest jest-mock-process

Buat file {root_project} > babel.config.js

```
module.exports = {
    presets: [
        ['@babel/preset-env', {
                targets: {node: 'current'},
            },
        ],
    ],
};
```

Ubah package.json

```
"scripts": {
  "start": "node index",
  "dev": "nodemon index || tr
  "test": "jest"
},
```

```
"jest": {
  "testEnvironment": "node",
  "coveragePathIgnorePatterns": [
    "/node_modules/"
  ]
}
```

Buat file src > server.js

```
import express from "express";
import {logErrorEvent} from "./events/logging.event'
import AppMiddleware from "./middlewares/app-middlew
import AppRouter from "./routes";
import http from "http";

const app = express();
app.use(AppMiddleware);
app.use(AppRouter);
export const server = http.createServer(app);
server.on('error', function (e) {
    logErrorEvent.emit('APP', {info: e});
    process.exit(1);
});
```

Ubah file src > app.js

```
import configure from './config';

import createDbConnection from "./database/connection"
import {logErrorEvent} from './events/logging.event';
import {server} from "./server";

createDbConnection(configure)
    .then((connection) => {
            if (connection.isConnected) {
                configure();
                server.listen(process.env.APP_PORT);
            }
        }
    ).catch((error) => {
    logErrorEvent.emit('DB', {info: error});
});
```

';

Buat file src > __tests__ > initializedTest.js

```
import ProductSchema from "../entities/product.schema";
import {createConnection, getConnection, getRepository} from "typeorm";
import UserSchema from "../entities/user.schema";
import CategorySchema from "../entities/category.schema";
import UserInfoSchema from "../entities/userInfo.schema";

export const init = async () => {
    const connection = await createConnection({
        type: 'sqlite',
        database: ':memory:',
        dropSchema: true,
        synchronize: true,
        entities: [UserSchema, UserInfoSchema, CategorySchema, ProductSchema]
    });

    await initData();
};
```

```
const initData = async () => {
    await  getRepository(UserSchema).save({
        userName: 'edo',
        userPassword: 'edo',
        userFullName: 'edo'
    });
    await  getRepository(CategorySchema).sav
        id: 1,
        categoryId: 'ABC',
        categoryName: 'A B C'
    });
    await  getRepository(ProductSchema).save
        id: 1,
        productId: 'XYZ',
        productName: 'X Y Z',
        price: 0,
        categoryId: 1
    });
};
```

```
export const clearance = () => {
    let conn = getConnection();
    return conn.close();
};

describe('Sample Test', () => {
    it('should test that true === true', () => {
        expect(true).toBe(true)
    })
});
```

Buat file src > __tests__ > category.router.test.js

```
import request from 'supertest';
import {server} from "../server";
import {clearance, init} from './initializeTest';

server.listen(process.env.APP_PORT);
const userCredentials = {
    userName: 'edo',
    userPassword: 'edo'
};
var authenticatedUser = request.agent(server);
beforeAll(async (done) => {
    await init();
    authenticatedUser
        .post('/auth')
        .send(userCredentials)
        .end((err, response) => {
            expect(response.statusCode).toEqual(200
            done();
        });
});
afterAll(async (done) => {
    await clearance();
    done();
});
```

```
describe('Category Router Test', () => {
    it('should call /', (done) => {
        authenticatedUser
            .get('/category')
            .end((err, response) => {
                expect(response.statusCode).toEq
                expect(response.body).toEqual([
                    {id: 1, categoryId: 'ABC', c
                ]);
                done();
            });
    })
});
```

```
ual(200);

ategoryName: 'A B C'}
```

Ubah file src > routes > category.router.js

```
import {Router} from 'express';
import {
    createACategory, getCategoryById, getListCategory,
    getListCategoryWithProduct
} from "../controller/category.controller";
import CategoryService from "../services/category.service";

const CategoryRouter = Router()
    .get('/', (req, res) => getListCategory(req, res, new CategoryService()))

...
```

Ubah file src > controller > category.controller.js

```
import {logErrorEvent, logInfoEvent} from "../events/logging.event";
import CategoryRepository from "../repository/category.repository";

const categoryRepository = new CategoryRepository();
export const getListCategory = async (req, res, categoryService) => {
    try {
        const rows = await categoryService.setRepository(categoryRepository).getAllCategory();
        return res.status(200).json(rows);
    } catch (err) {
        logErrorEvent.emit('CONTROLLER', {info: err, res: res});
    }
};
```

Buat file src > __tests__ > category.controller.test.js

```
import {
    categoryEvent, createACategory, getListCategory,
    getListCategoryWithProduct
} from "../controller/category.controller";
import CategoryService from "../services/category.se

let mockResponse;
let mockRequest;
beforeAll(async () => {
    mockResponse = () => {
        const res = {};
        res.status = jest.fn().mockReturnValue(res);
        res.json = jest.fn().mockReturnValue(res);
        return res;
    };
    mockRequest = () => {
        const res = {};
        return res;
    };
});
describe('Category Controller Test', () => {
    it('should call find all category', async () =>
        const req = mockRequest();
        const res = mockResponse();
        const categoryService = new CategoryService(
        categoryService.getAllCategory = jest.fn(()
        await getListCategory(req, res, categoryServ
        expect(res.json).toHaveBeenCalledWith([{id:
        expect(res.status).toHaveBeenCalledWith(200)
    });
```

```
it('should call find all category with product', asyn
        const req = mockRequest();
        const res = mockResponse();
        const categoryService = new CategoryService(
        categoryService.getAllCategoryProduct = jest.
        await getListCategoryWithProduct(req, res, ca
        expect(res.json).toHaveBeenCalledWith([{id: '
        expect(res.status).toHaveBeenCalledWith(200)
    });
    it('should call create product', async () => {
        const req = mockRequest();
        const res = mockResponse();
        const categoryService = new CategoryService(
        categoryService.createCategory = jest.fn(() =
        await createACategory(req, res,categoryServic
        expect(res.json).toHaveBeenCalledWith([{id: '
        expect(res.status).toHaveBeenCalledWith(200)
    });
});
```

```
nc () ⇒ {


);
.fn(() ⇒ [{id: '1'}]);
ategoryService);
'1'}]);




);
⇒ [{id: '1'}]);
ce);
'1'}]);
```

Ubah file src > services > category.service.js

```
export default class CategoryService {
    setRepository(repo) {
        this.repo = repo;
        return this;
    }

    getAllCategory() {
        return this.repo.findAllCategory();

    }

    getAllCategoryProduct() {
        return this.repo.findAllCategoryProduct();

    }

    getCategoryById(id) {
        return this.repo.findOne(id);
    }

    createCategory(category) {
        return this.repo.createCategory(category);
    }
}
```

Buat file src > __tests__ > category.service.test.js

```
import CategoryRepository from "../repository/category.repository";
import CategoryService from "../services/category.service";

let categoryRepository;
let categoryService;
beforeAll(() => {
    categoryRepository = new CategoryRepository();
    categoryService = new CategoryService();
});

describe('Category Service Test', () => {
    it('should call get all category', async () => {
        categoryRepository.findAllCategory = jest.fn(() => {
            return [{id: 1, categoryName: 'sample category'}]
        });
        categoryService.setRepository(categoryRepository);
        expect(categoryService.getAllCategory()).toEqual([{id: 1, categoryName: 'sample category'}]
    });
    it('should create category', async () => {
        const category = {id: '2', categoryId: 'Anything'};
        categoryRepository.createCategory = jest.fn((category) => {
            return category;
        });
        categoryService.setRepository(categoryRepository);
        expect(categoryService.createCategory(category)).toEqual(category)
    });
});
```

)

Ubah file src > events > logging.event.js

```
import events from 'events';
import {log} from "../logger";

export const logErrorEvent = new events.Event

export const logging = {
    recordLog(logInfo) {
        switch (logInfo.logType) {
            case 'ERROR':
                log.error(logInfo.logTitle, l
                break;
            case 'INFO':
                log.info(logInfo.logTitle, lo
                break;
            default:
                log.debug(logInfo.logTitle, l
        }
    }
};
```

```
logErrorEvent.on('APP', function (ev) {
    logging.recordLog({
        logType: 'ERROR', logTitle: 'APP-FAILED', logMessage: ev.
    });
});

logErrorEvent.on('DB', function (ev) {
    logging.recordLog({
        logType: 'ERROR', logTitle: 'DB-FAILED', logMessage: ev.i
    });
});

logErrorEvent.on('ROUTE', function (ev) {
    logging.recordLog({
        logType: 'ERROR', logTitle: 'ROUTE-FAILED', logMessage: e
    ev.res.status(404);
    ev.res.json({message: 'Not Found.'});
});

logErrorEvent.on('SESSION', function (ev) {
    ev.res.sendStatus(401);
});

logErrorEvent.on('CONTROLLER', function (ev) {
    logging.recordLog({
        logType: 'ERROR', logTitle: 'CONTROLLER-FAILED', logMessa
    ev.res.status(200);
    ev.res.json({message: 'We are sorry, your request can not be
});
```

```
.info


info


ev.info});


age: ev.info});
 processed'})
```

```
export const logInfoEvent = new events.EventEmitter();

logInfoEvent.on('ACCESS', function (ev) {
    logging.recordLog({logType: 'INFO', logTitle: 'USER-ROUTE', logMessage: ev.info});
});
logInfoEvent.on('NO_DATA_FOUND', function (ev) {
    ev.res.status(200)
    ev.res.json({status: '400', message: `No Data Found on ${ev.info}`})
});
```

Buat file src > __tests__ > logEvent.test.js

```
import {logErrorEvent, logging} from '../events/logging.event';

let spy;
beforeEach(() => {
    spy = jest.spyOn(logging, 'recordLog');
});
afterEach(() => {
    jest.clearAllMocks();
});

describe('Logging Event Test', () => {
    it('should log error in APP', () => {
        const info = {info: 'test'};
        const logInfoExpected = {logType: 'ERROR', logTitle: 'APP-FAILED', logMessage: info.info};
        logErrorEvent.emit('APP', info);
        expect(spy).toHaveBeenCalledTimes(1);
        expect(spy).toHaveBeenCalledWith(logInfoExpected);
    });

    it('should log error in CONTROLLER', () => {
        const info = {info: 'test', res: {json: jest.fn(), status: jest.fn()}};
        logErrorEvent.emit('CONTROLLER', info);
        expect(info.res.json).toHaveBeenCalledTimes(1);
        expect(info.res.status).toHaveBeenCalledTimes(1);
        expect(spy).toHaveBeenCalledTimes(1);
    })
```

```
it('should log on Session timeout and send 401', () => {
    const info = {res: {sendStatus: jest.fn()}};
    logErrorEvent.emit('SESSION', info);
    expect(info.res.sendStatus).toHaveBeenCalledTimes(1);
    expect(info.res.sendStatus).toHaveBeenCalledWith(401);
});

});
```