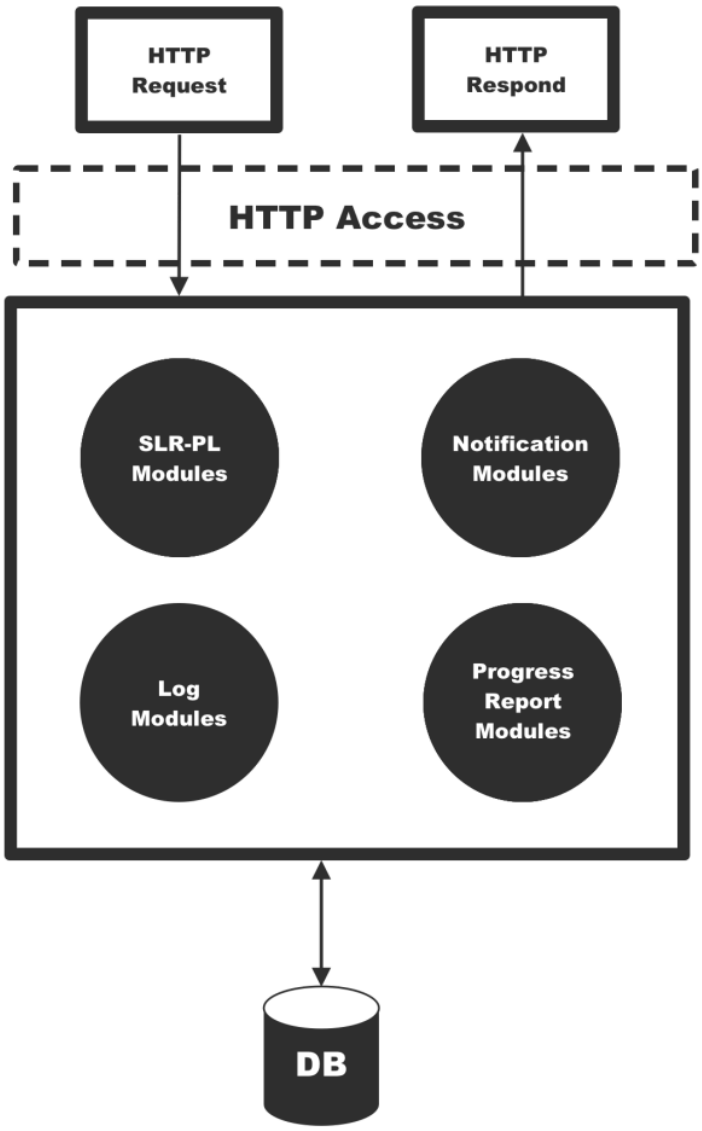
Design of Digital Platform for Self-Regulated and Personalized Learning

by Irwan Kautsar

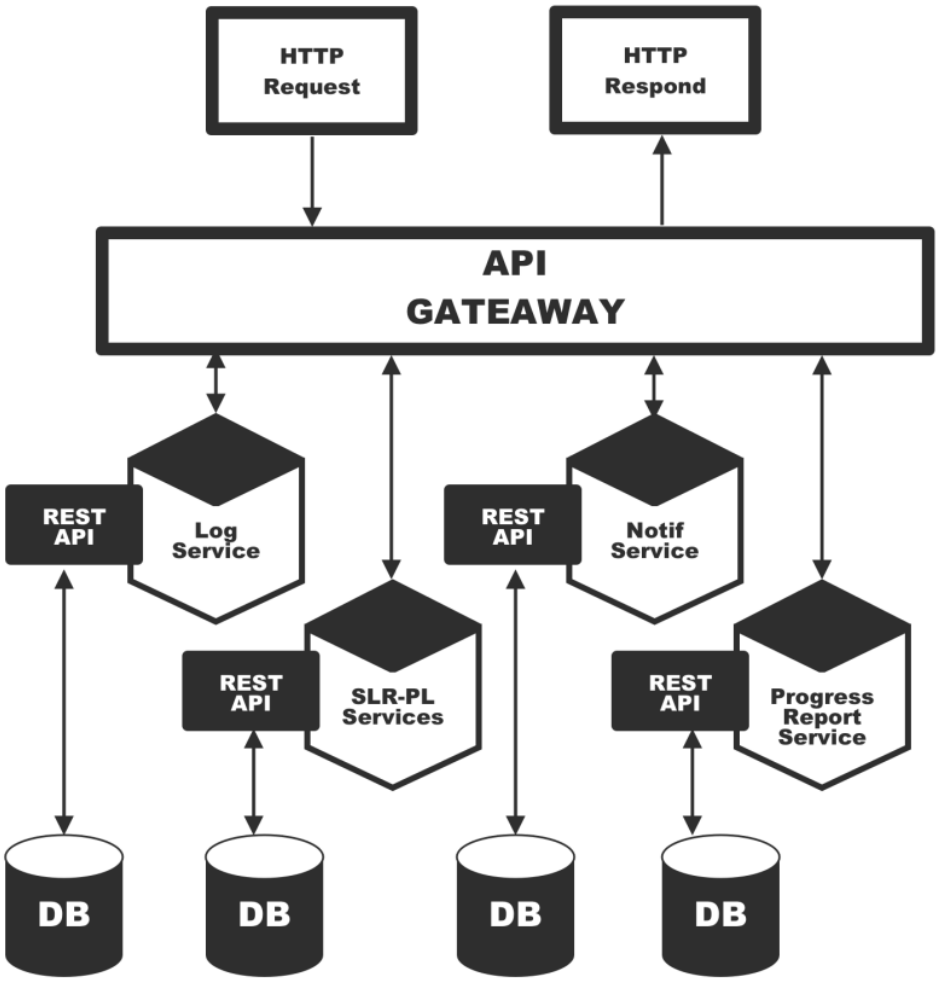
Submission date: 06-Aug-2023 12:56PM (UTC+0700)
Submission ID: 2141897101
File name: Deskripsi_Ciptaan_RKLN-CS.pdf (1.31M)
Word count: 221
Character count: 1613

Design of Digital Platform for Self-Regulated and Personalized Learning

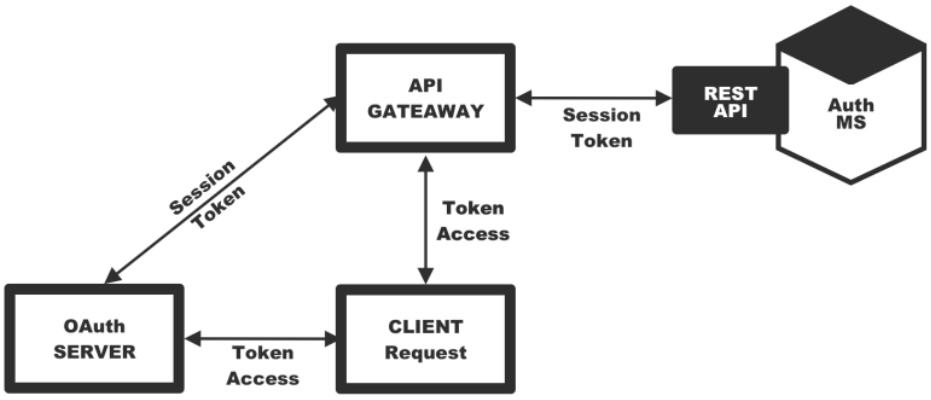
Penciptaan platform ini menyajikan migrasi sistem alat pendukung rapid prototyping dari monolit ke arsitektur microservice yang akan digunakan sebagai implementasi Pembelajaran Self-Regulated and Personalized Learning. Seperti pada pengembangan awal, alat pendukung yang dikembangkan adalah arsitektur monolitik dan platform berbasis web. Seiring dengan pertumbuhan mahasiswa sebagai pengguna dan penambahan modul rapid prototyping framework yang akan digunakan, arsitektur platform yang telah dikembangkan secara monolit akan menjadi lebih kompleks. Untuk itu arsitektur platform digital urban farming perlu untuk dikembangkan menjadi layanan web (web service) yang lebih modular dengan mengimplementasikan konsep pengembangan perangkat lunak berbasis microservice. Versi terbaru pada platform digital urban farming akan memberikan keunggulan dalam hal modularitas, skalabilitas, dan kemudahan dalam pemeliharaan fitur-fitur yang telah dikembangkan sebagai implementasi Pembelajaran Self-Regulated and Personalized Learning akan lebih mudah diintegrasikan sebagai manfaat dari arsitektur berbasis microservice.



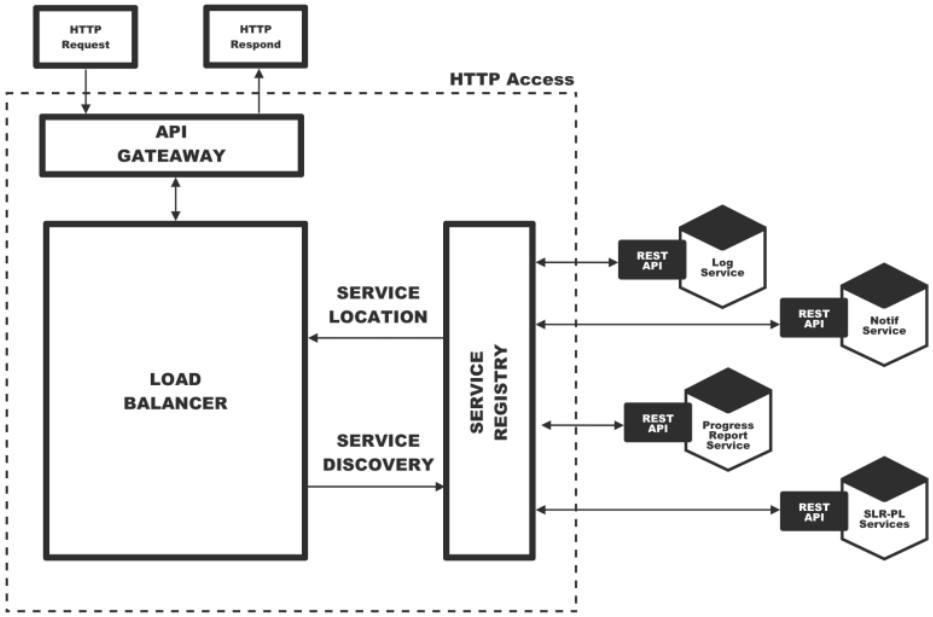
Gambar 1. Arsitektur Monolithic Platform Supportive Tools Self-Regulated dan Personalized Learning



Gambar 2. Arsitektur Microservice Platform Supportive Tools Self-Regulated dan Personalized Learning



Gambar 3. Arsitektur Microservice Otentikasi Platform Self-Regulated dan Personalized Learning



Gambar 4. Arsitektur Service Registry dan Load Balancer Platform Supportive Tool

```python
from flask import Flask, jsonify
from flask_restful import Api, Resource, reqparse
from peewee import *

db = "fnf.db"
database = SqliteDatabase(db)

class BaseModel(Model):
    class Meta:
        database=database

class fnf(BaseModel):
    idfnf = AutoField(primary_key=True)
    kodekelompok = TextField(null=True)
    type = IntegerField(null=True) # 1 for functional, 2 for non functional
    name = CharField()
    description = TextField(null=True)
    backlog = TextField(null=True)
    assign_to = TextField(null=True)
    target_finish = IntegerField(null=True)
    target_finish_print = CharField(null=True)
    status = CharField(null=True) # status progress fitur

def create_tables():
    with database:
        database.create_tables([fnf])

app = Flask(__name__)
api = Api(app)
```

Gambar 5. Implementasi: Flask dan Database

```python
class resourceFNF(Resource):
    def get(self):
        parser = reqparse.RequestParser()
        parser.add_argument('kodekelompok')
        parser.add_argument('idfnf')
        args = parser.parse_args()
        if args['idfnf'] and args['kodekelompok']:
            return jsonify({"data":None, "message":"Too Many Parameters"})
        elif args['idfnf']:
            dataFNF = list(fnf.select().where(fnf.idfnf==args['idfnf']).dicts())
        elif args['kodekelompok']:
            dataFNF = list(fnf.select().where(fnf.kodekelompok==args['kodekelompok']).dicts())
        else:
            dataFNF = list(fnf.select().dicts())
        return jsonify({"data":dataFNF, "message":"success"})

    def post(self):
        parser = reqparse.RequestParser()
        parser.add_argument('kodekelompok', location='json')
        parser.add_argument('type', location='json')
        parser.add_argument('name', location='json')
        parser.add_argument('description', location='json')
        parser.add_argument('backlog', location='json')
        args = parser.parse_args()

        fnf.create(
            kodekelompok=args['kodekelompok'],
            type=args['type'],
            name=args['name'],
            description=args['description'],
            backlog=args['backlog']
        )

        return jsonify({"data":None, "message":"Create FNF Success"})
```

Gambar 6. Implementasi Service: GET dan POST Task

```python
def put(self):
    parser = reqparse.RequestParser()
    parser.add_argument('idfnf', location='json')
    parser.add_argument('status', location='json')
    parser.add_argument('description', location='json')
    parser.add_argument('assign_to', location='json')
    args = parser.parse_args()

    # cek if fnf exists
    cek = fnf.select().where(fnf.idfnf == args['idfnf'])
    if cek.exists():
        update_fnf = fnf.update(
                status=args['status'],
                description=args['description'],
                assign_to=args['assign_to']
            ).where(fnf.idfnf == args['idfnf'])

        update_fnf.execute()

        return jsonify({"data":None, "message":"Update FNF Success"})
    else:
        return jsonify({"data":None, "message":"Update FNF Failed. FNF not Found"})
```

Gambar 7. Implementasi Service: PUT Task

```python
def delete(self):
    parser = reqparse.RequestParser()
    parser.add_argument('idfnf', location='args')
    args = parser.parse_args()

    # cek if fnf exists
    cek = fnf.select().where(fnf.idfnf == args['idfnf'])
    if cek.exists():
        delete_fnf = fnf.delete().where(fnf.idfnf == args['idfnf'])
        delete_fnf.execute()
        return jsonify({"data":None, "message":"Delete FNF Success"})
    else:
        return jsonify({"data":None, "message":"Delete FNF Failed. FNF not Found"})
```

Gambar 8. Implementasi Service : DELETE Task

```
api.add_resource(resourceFNF, '/')

if __name__ == '__main__':
    create_tables()
    app.run(debug=True)
```

Gambar 9. Implementasi Service Initiation

```python
class TaskFNF(TaskSet):
    @task
    def getFNFbykodekelompok(self):
        kodekelompok = random.randint(1,1000)
        self.client.get(f"/?kodekelompok={kodekelompok}")

    @task
    def getFNFbyidfnf(self):
        idfnf = random.randint(1,1000)
        self.client.get(f"/?idfnf={idfnf}")

    @task
    def postFNF(self):
        kodekelompok = random.randint(1,1000)
        tipe = random.randint(1,2)
        ranuuid = str(uuid.uuid4().hex)
        self.client.post("/",json={"kodekelompok":kodekelompok, "type":tipe, "name":ranuuid,"description":ranuuid, "backlog":ranuuid})

    @task
    def putFNF(self):
        idfnf = random.randint(1,1000)
        ranuuid = str(uuid.uuid4().hex)
        self.client.put("/",json={"idfnf":idfnf, "status":ranuuid,"description":ranuuid, "assign_to":ranuuid})

    @task
    def deleteFNF(self):
        idfnf = random.randint(1,1000)
        self.client.delete(f"/?idfnf={idfnf}")
```

Gambar 10. Implementasi Service Task: Load Initiation

Design of Digital Platform for Self-Regulated and Personalized Learning

ORIGINALITY REPORT

0%	0%	0%	0%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	On		