**Phase A: Install and Deploy Software in Virtualized Environments**

1. **Set Up the Virtual Machine (VM):**
   * Choose a cloud provider (e.g., Google Cloud, AWS, or Azure).
   * Create a VM instance with specifications similar to Ubuntu 20.04, 2 CPUs, and 4GB RAM.
   * Install necessary tools like Node.js, npm, Git, and MongoDB on the VM.
     + Use commands like:

bash

Kodu kopyala

sudo apt update

sudo apt install nodejs npm git mongodb

1. **Set Up Project Folder Structure:**
   * Create a directory structure:

bash

Kodu kopyala

/src

/routes -> Define API endpoints

/models -> Define MongoDB schema

/controllers -> Write business logic

/config -> Store configuration files (e.g., database connection)

1. **Deploy Code:**
   * Clone your project from your GitHub repository onto the VM:

bash

Kodu kopyala

git clone https://github.com/your-repo/piazza.git

cd piazza

npm install

* + Run the app:

bash

Kodu kopyala

node server.js

1. **Test API Availability:**
   * Test that the API endpoints are accessible via the VM's public IP address using Postman.
2. **Documentation:**
   * Include:
     + Screenshots of the VM setup and folder structure.
     + A short description of the deployment process.

**Phase B: Authentication and Verification**

1. **Set Up User Registration and Login:**
   * Create MongoDB collections to store user credentials (e.g., email, hashed password, role).
   * Implement API endpoints for:
     + **Registration**: Hash passwords using libraries like bcrypt or argon2.
     + **Login**: Verify credentials and issue JWTs.
2. **OAuth v2 and JWT Implementation:**
   * Use a library like jsonwebtoken to generate and validate tokens.
   * Example:

javascript

Kodu kopyala

const jwt = require('jsonwebtoken');

// Generate JWT

const token = jwt.sign({ userId: user.\_id }, 'secret\_key', { expiresIn: '1h' });

// Verify JWT

jwt.verify(token, 'secret\_key', (err, decoded) => {

if (err) {

return res.status(401).send("Unauthorized");

}

});

1. **Validation:**
   * Use libraries like express-validator to ensure valid inputs.
   * Example:

javascript

Kodu kopyala

const { body } = require('express-validator');

app.post('/register', [

body('email').isEmail(),

body('password').isLength({ min: 6 }),

], (req, res) => {

// Handle validation errors

});

1. **Documentation:**
   * Describe user authentication flow and input validations.
   * Include screenshots of the registration and login API responses.

**Phase C: Development of Piazza RESTful APIs**

1. **Design Database Models:**
   * Use mongoose to create schemas:

javascript

Kodu kopyala

const mongoose = require('mongoose');

const PostSchema = new mongoose.Schema({

title: String,

topic: [String],

body: String,

owner: String,

likes: Number,

dislikes: Number,

comments: [{ commenter: String, content: String }],

expirationTime: Date,

status: { type: String, enum: ['Live', 'Expired'] },

});

module.exports = mongoose.model('Post', PostSchema);

1. **Develop REST API Endpoints:**
   * **Create Post**:

javascript

Kodu kopyala

app.post('/posts', (req, res) => {

// Save new post to MongoDB

});

* + **Browse Posts by Topic**:

javascript

Kodu kopyala

app.get('/posts/:topic', (req, res) => {

// Retrieve posts by topic

});

* + **Like/Dislike/Comment**:

javascript

Kodu kopyala

app.post('/posts/:id/interact', (req, res) => {

// Update likes, dislikes, or comments

});

1. **Secure APIs:**
   * Apply JWT middleware for all protected endpoints:

javascript

Kodu kopyala

const authMiddleware = (req, res, next) => {

const token = req.headers.authorization.split(' ')[1];

jwt.verify(token, 'secret\_key', (err, decoded) => {

if (err) return res.status(401).send("Unauthorized");

req.user = decoded;

next();

});

};

app.use(authMiddleware);

1. **Documentation:**
   * Include endpoint examples and database schema in the report.

**Phase D: Testing Application**

1. **Manual Testing with Postman:**
   * Create and save a collection for all API endpoints.
   * Test edge cases like unauthorized access or invalid inputs.
2. **Automated Testing with Mocha/Chai:**
   * Write scripts to test API functionality:

javascript

Kodu kopyala

const chai = require('chai');

const chaiHttp = require('chai-http');

chai.use(chaiHttp);

describe('Piazza API', () => {

it('should create a post', (done) => {

chai.request(server)

.post('/posts')

.send({ title: 'Hello World' })

.end((err, res) => {

res.should.have.status(200);

done();

});

});

});

1. **Document Results:**
   * Add screenshots or logs from Postman and testing scripts.

**Phase E: Deploy on Docker**

1. **Write a Dockerfile:**

dockerfile

Kodu kopyala

FROM node:14

WORKDIR /app

COPY . .

RUN npm install

CMD ["node", "server.js"]

1. **Deploy on Google Cloud VM:**
   * Build and run the Docker container:

bash

Kodu kopyala

docker build -t piazza .

docker run -p 8080:8080 piazza

1. **Document Commands and Screenshots.**

**Phase F: Deploy on Kubernetes**

1. **Set Up Kubernetes Cluster:**
   * Use Google Cloud to create a Kubernetes cluster.
2. **Deploy Application:**
   * Write deployment and service YAML files:

yaml

Kodu kopyala

apiVersion: apps/v1

kind: Deployment

metadata:

name: piazza

spec:

replicas: 5

template:

spec:

containers:

- name: piazza

image: piazza:latest

1. **Document Kubernetes Commands and Results.**

**Phase G: Technical Report**

1. **Content:**
   * Write about each phase.
   * Include challenges, solutions, and screenshots.
2. **References:**
   * Use Harvard referencing to cite resources.

**Phase H: Submit Quality Scripts**

1. **Code Review:**
   * Ensure proper formatting and meaningful comments.
   * Remove unnecessary logs and unused code.
2. **Final Testing:**
   * Run all test cases and check for bugs.