



UNIVERSITAS PAMULANG
KARTU UJIAN AKHIR SEMESTER GANJIL 2024/2025
NOMOR UJIAN : 01212413857172

FAKULTAS / PRODI : ILMU KOMPUTER / TEKNIK INFORMATIKA S1

NAMA MAHASISWA : KURNIA SANDY

NIM : 211011450327

SHIFT : REGULER C

No	Hari/ Tanggal	Waktu	Ruang	Kelas	Mata Kuliah	Paraf
1	Sabtu, 4 Jan 2025	07.40 - 09.20	V.623	06TPLE005	PEMROGRAMAN WEB 2	1
2	Sabtu, 4 Jan 2025	07.40 - 09.20	V.623	06TPLE005	SISTEM INFORMASI MANAJEMEN	2
3	Sabtu, 4 Jan 2025	09.20 - 11.00	V.623	06TPLE005	KERJA PRAKTEK	3
4	Sabtu, 4 Jan 2025	09.20 - 11.00	V.623	06TPLE005	MOBILE PROGRAMMING	4
5	Sabtu, 4 Jan 2025	11.00 - 13.50	V.623	06TPLE005	TEKNIK KOMPILASI	5
6	Sabtu, 4 Jan 2025	13.50 - 15.30	V.623	06TPLE005	KOMPUTER GRAFIK I	6
7	Sabtu, 4 Jan 2025	13.50 - 15.30	V.623	06TPLE005	REKAYASA PERANGKAT LUNAK	7
8	Sabtu, 4 Jan 2025	16.00 - 17.40	V.623	06TPLE005	KECERDASAN BUATAN	8

Peraturan dan Tata Tertib Peserta Ujian

1. Peserta ujian harus berpakaian rapi, sopan dan memakai jaket Almamater
2. Peserta ujian sudah berada di ruangan sepuluh menit sebelum ujian dimulai
3. Peserta ujian yang terlambat diperkenankan mengikuti ujian setelah mendapat ijin, tanpa perpanjangan waktu
4. Peserta ujian hanya diperkenankan membawa alat-alat yang ditentukan oleh panitia ujian
5. Peserta ujian dilarang membantu teman, mencontoh dari teman dan tindakan-tindakan lainnya yang mengganggu peserta ujian lain
6. Peserta ujian yang melanggar tata tertib ujian dikenakan sanksi akademik



Tangerang Selatan, 3 Januari 2025
Ketua Panitia Ujian

Dr. Ubaid Al Faruq, S.Pd., M.Pd.
NIDN. 0418028702

UJIAN AKHIR SEMESTER
SEMESTER GANJIL
TAHUN AKADEMIK 2024/2025

Mata Kuliah/SKS	: Pemrograman Web 2	Semester / Kelas	: 6 / 6TPL
Program Studi	: Teknik Informatika	Ruang	: V.
Fakultas	: Ilmu Komputer	Waktu	: 60 Menit
Nama Dosen	: Fajar Agung N	Jenis Ujian	: Praktek Observasi

Kerjakan soal dibawah ini dengan Uraian yang jelas!

1. Anda diminta untuk membangun sebuah sistem autentikasi pengguna pada sebuah aplikasi web. Sistem ini harus memiliki fitur lupa password dan perlindungan terhadap serangan brute force.


Pertanyaan:

- a) Algoritma hashing apa yang paling cocok digunakan untuk menyimpan password pengguna? Jelaskan alasannya dan berikan contoh programnya. (10 poin)
 - b) Bagaimana cara Anda mengimplementasikan fitur lupa password dengan aman? Buatlah contoh programnya! (20 poin)
 - c) Teknik apa saja yang dapat Anda gunakan untuk mencegah serangan brute force? Berikan contoh implementasi programnya! (10 poin)
2. Buatlah sebuah aplikasi web sederhana yang berfungsi sebagai to-do list. Aplikasi ini harus memiliki fitur untuk menambahkan, menghapus, dan mengedit tugas. Data tugas harus disimpan dalam database MySQL.

Tugas:

- a) Buatlah Query DDL untuk membuat database dan tabel yang akan digunakan untuk menyimpan data tugas! (20 poin)
- b) Fungsi-fungsi PHP apa saja yang akan Anda gunakan untuk berinteraksi dengan database! (20 poin)
- c) Buatlah REST API untuk GET, POST, PUT dan DELETE ke tabel todo_list!(20 poin)

****** Selamat Mengerjakan ******

Nama : Kurnia Sandy	Nama Dosen : Fajar Agung N
NIM : 211011450327	Mata Kuliah : Pemrograman Web 2
Semester / Kelas : 6 / 06TPLE005	
Program Studi : Teknik Informatika	

1. a) Algoritma hashing yang paling cocok untuk menyimpan password pengguna adalah bcrypt. Alasannya:

1. Bcrypt dirancang khusus untuk hashing password dan memiliki faktor work yang dapat disesuaikan.
2. Bcrypt menggunakan salt secara otomatis untuk mencegah serangan rainbow table.
3. Bcrypt relatif lambat, yang merupakan keuntungan untuk mencegah serangan brute force.

contoh implementasi bcrypt menggunakan Node.js:

```
import bcrypt from 'bcrypt';

async function hashPassword(password) {
  const saltRounds = 10;

  try {
    const hashedPassword = await bcrypt.hash(password, saltRounds);
    console.log('Hashed password:', hashedPassword);
    return hashedPassword;
  } catch (error) {
    console.error('Error hashing password:', error);
  }
}

// Contoh penggunaan
hashPassword('mySecurePassword123');
```

b) Implementasi fitur lupa password dengan aman

Cara implementasi aman: bisa menggunakan metode token reset password yang dikirim melalui email.

contoh implementasinya:

```
import { NextResponse } from 'next/server';
import crypto from 'crypto';
import { sendEmail } from '@lib/email';

// Asumsikan kita memiliki fungsi untuk menyimpan token ke database
import { saveResetToken } from '@lib/db';

export async function POST(req: Request) {
  const { email } = await req.json();

  // Generate token
  const token = crypto.randomBytes(20).toString('hex');

  // Simpan token ke database dengan waktu kadaluarsa (misalnya 1 jam)
  const expiryDate = new Date(Date.now() + 3600000);
  await saveResetToken(email, token, expiryDate);

  // Kirim email
  const resetUrl = `https://yourdomain.com/reset-password?token=${token}`;
  await sendEmail({
    to: email,
    subject: 'Reset Password',
    text: `Klik link berikut untuk reset password Anda: ${resetUrl}`
  });

  return NextResponse.json({ message: 'Email reset password telah dikirim' });
}
```

c) Teknik mencegah serangan brute force

Teknik pencegahan:

1. Rate limiting: Membatasi jumlah percobaan login dalam waktu tertentu.
2. Captcha: Menambahkan verifikasi captcha setelah beberapa kali percobaan gagal.
3. Account locking: Mengunci akun sementara setelah beberapa kali percobaan gagal.

contoh implementasi rate limiting menggunakan Express.js dan Redis:

```
import express from 'express';
import rateLimit from 'express-rate-limit';
import RedisStore from 'rate-limit-redis';
import Redis from 'ioredis';

const app = express();

const redis = new Redis({
  host: 'localhost',
  port: 6379,
});

const limiter = rateLimit({
  store: new RedisStore({
    sendCommand: (...args) => redis.call(...args),
  }),
  windowMs: 15 * 60 * 1000, // 15 menit
  max: 100, // Batas 100 request per IP dalam 15 menit
  message: 'Terlalu banyak percobaan login, silakan coba lagi nanti.',
});

// Terapkan rate limiting ke rute login
app.post('/login', limiter, (req, res) => {
  // Logika login di sini
});
```

```
app.listen(3000, () => {  
  console.log('Server berjalan di port 3000');  
});
```

2. a) Query DDL untuk Membuat Database dan Tabel

Berikut adalah query SQL untuk membuat database dan tabel `todo_list`:

```
CREATE DATABASE IF NOT EXISTS todo_app;  
  
USE todo_app;  
  
CREATE TABLE IF NOT EXISTS todo_list (  
  id INT AUTO_INCREMENT PRIMARY KEY,  
  task VARCHAR(255) NOT NULL,  
  status ENUM('pending', 'completed') DEFAULT 'pending',  
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON  
  UPDATE CURRENT_TIMESTAMP  
);
```

b) Fungsi PHP untuk Berinteraksi dengan Database

Koneksi ke Database:

- `mysqli_connect()`: Membuka koneksi ke database (prosedural).
- `new mysqli()`: Membuka koneksi (OOP).
- `mysqli_close()`: Menutup koneksi.

Menjalankan Query:

- `mysqli_query()`: Menjalankan query SQL.
- `mysqli_prepare()`: Mempersiapkan query untuk digunakan dengan parameter bind.
- `mysqli_stmt_bind_param()`: Mengikat parameter untuk query.
- `mysqli_stmt_execute()`: Menjalankan query yang sudah dipersiapkan.
- `mysqli_fetch_assoc()`: Mengambil hasil query dalam bentuk array asosiatif.
- `mysqli_fetch_row()`: Mengambil hasil query dalam bentuk array numerik.

Error Handling:

- `mysqli_error()`: Mengambil pesan error dari database.
- `mysqli_connect_error()`: Mengambil pesan error dari koneksi.

c) REST API untuk GET, POST, PUT, dan DELETE

REST API sederhana untuk mengelola data dalam tabel `todo_list`.

Generated lib/db.ts

```
import mysql from 'mysql2/promise';

export async function query({ query, values = [] }: { query: string; values?: any[] }) {
  const connection = await mysql.createConnection({
    host: process.env.DB_HOST,
    user: process.env.DB_USER,
    password: process.env.DB_PASSWORD,
    database: process.env.DB_NAME,
  });

  try {
    const [results] = await connection.execute(query, values);
    return results;
  } finally {
    await connection.end();
  }
}
```

Generated app/api/todos/route.ts

```
import { NextResponse } from 'next/server';
import { query } from '@lib/db';

// GET /api/todos
export async function GET() {
```

```

try {
  const todos = await query({
    query: 'SELECT * FROM todo_list',
  });
  return NextResponse.json(todos);
} catch (error) {
  return NextResponse.json({ error: 'Internal Server Error' }, { status: 500 });
}
}

// POST /api/todos
export async function POST(request: Request) {
  try {
    const { task } = await request.json();
    const result = await query({
      query: 'INSERT INTO todo_list (task) VALUES (?)',
      values: [task],
    });
    return NextResponse.json({ id: result.insertId, task }, { status: 201 });
  } catch (error) {
    return NextResponse.json({ error: 'Internal Server Error' }, { status: 500 });
  }
}

```

Generated app/api/todos/[id]/route.ts

```

import { NextResponse } from 'next/server';
import { query } from '@lib/db';

// PUT /api/todos/[id]
export async function PUT(request: Request, { params }: { params: { id: string } }) {

```



```

try {
  const { task, status } = await request.json();
  await query({
    query: 'UPDATE todo_list SET task = ?, status = ? WHERE id = ?',
    values: [task, status, params.id],
  });
  return NextResponse.json({ message: 'Todo updated successfully' });
} catch (error) {
  return NextResponse.json({ error: 'Internal Server Error' }, { status: 500 });
}
}

// DELETE /api/todos/[id]
export async function DELETE(request: Request, { params }: { params: { id: string } }) {
  try {
    await query({
      query: 'DELETE FROM todo_list WHERE id = ?',
      values: [params.id],
    });
    return NextResponse.json({ message: 'Todo deleted successfully' });
  } catch (error) {
    return NextResponse.json({ error: 'Internal Server Error' }, { status: 500 });
  }
}

```

Generated .env.local

```

DB_HOST=localhost
DB_USER=your_username
DB_PASSWORD=your_password
DB_NAME=todo_app

```