PROGRESS 04

MVC: Controller-Model-View

Pemrograman Web
Program Studi Sistem Informasi

Dosen pengampu:
Ir. Aryo Pinandito, Ph.D.

Anggota kelompok
1. Calista Aulia Rianka Rachma– 235150407111032
2. Ailsa Zahra Sahdana - 235150401111017
3. Rahman Zaky – 235150401111030

Daftar Isi

Table of Contents

Rancangan Implementasi MVC

Use Case: Mencari dan Mengelola Acara GrowTogether

Penyusun Use Case: Rahman Zaky - 2351504011111030

Use case Mencari dan Mengelola Acara GrowTogether menjelaskan proses interaksi pengguna dengan fitur acara pada GrowTogether di platform GrowLink. Pada fitur ini pengguna dapat melakukan berbagai aktivitas seperti melihat daftar acara kegiatan, mencari acara, memberikan ulasan, mendaftarkan diri sebagai pemateri, dan membuat acara baru yang dilakukan pada halaman-halaman tertentu. Untuk lebih detail, dapat dilihat pada use case scenario berikut:

Actor:	1. Pengguna 2. Pemateri (Turunan dari Pengguna)
Description:	Platform GrowTogether memungkinkan Pengguna untuk mencari, melihat detail, dan mendaftar pada event (seperti webinar atau diskusi) yang dibuat oleh Pemateri. Pengguna yang telah mendaftar dan mengikuti event dapat memberikan ulasan berupa rating dan teks. Pengguna juga dapat mendaftar untuk menjadi Pemateri agar bisa membuat dan mengelola event mereka sendiri, termasuk mengunggah materi pendukung.
Use Case Goal:	Pengguna dapat mencari, mendaftar, mengikuti, dan memberikan ulasan pada event. Pemateri dapat membuat, mengelola, dan membagikan materi untuk event mereka.
Preconditions:	1. Pengguna (atau Pemateri) diasumsikan telah "masuk" ke dalam aplikasi Growlink dan memasuki fitur GrowTogether (saat ini disimulasikan dengan ID pengguna yang di-hardcode di controller). 2. Pengguna yang ingin membuat event harus memiliki peran sebagai 'Pemateri'.
Main Flow:	1. Mencari Daftar Event (Beranda) a. Pengguna membuka halaman utama GrowTogether b. Sistem menampilkan daftar semua event yang tersedia, beserta nama pembuat dan topik event. c. Pengguna dapat menekan salah satu event untuk melihat detailnya. 2. Mencari Event a. Pengguna menggunakan fitur pencarian di halaman utama. b. Pengguna mengetik kata kunci yang relevan dengan event yang dicari (misalnya judul, deskripsi, nama pemateri, atau topik). c. Pengguna menekan tombol "Search".

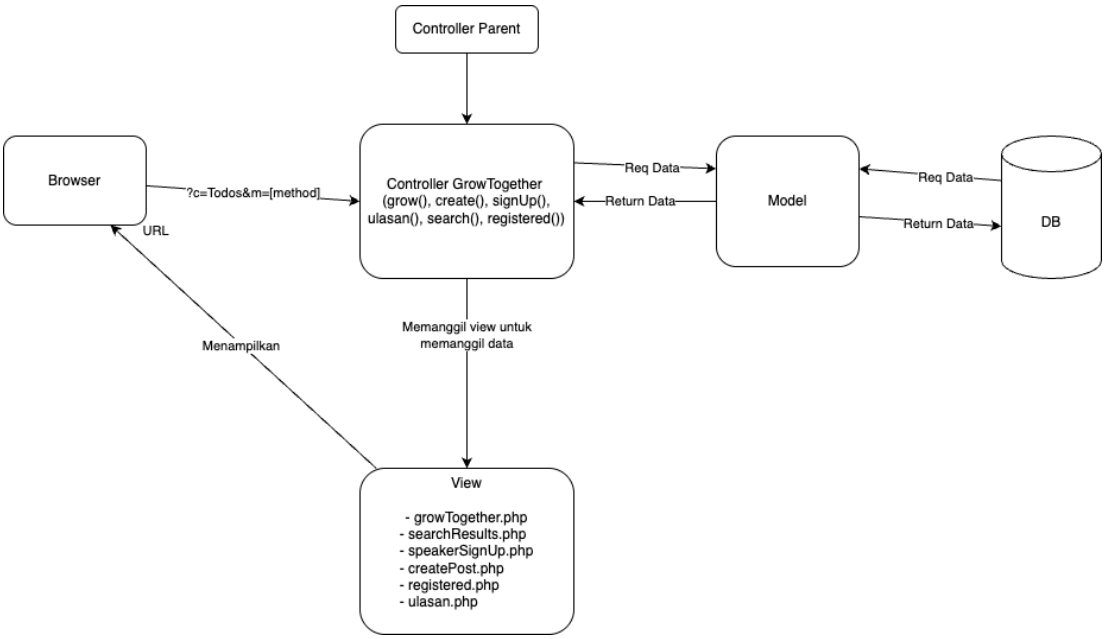
	d. Sistem menampilkan daftar event yang sesuai dengan kata kunci pencarian pada halaman Hasil Pencarian (searchResults.php). 3. Melihat Detail Event a. Dari halaman Beranda atau Hasil Pencarian, Pengguna menekan pada judul atau gambar event. b. Sistem menampilkan halaman Detail Event (eventDetail.php) yang berisi informasi lengkap tentang event tersebut (gambar, judul, deskripsi, pembuat, topik, dan link ke ringkasan materi jika ada). 4. Mendaftar (Join) Event (Ekstensi dari Melihat Daftar Event atau Melihat Detail Event) a. Pengguna menekan tombol "Join Event" pada kartu event di halaman Beranda atau di halaman Detail Event. b. Sistem mencatat bahwa Pengguna tersebut telah terdaftar pada event tersebut (menyimpan data ke tabel event_registrations). c. Sistem menampilkan notifikasi pop-up "Successfully joined the event!" dan tombol pada event tersebut berubah menjadi "✓ Joined". (Catatan: Notifikasi email belum diimplementasikan). 5. Melihat Aktivitas Saya a. Pengguna menekan tombol "My Activities" di navigasi footer. b. Sistem menampilkan halaman Aktivitas Saya (registered.php). c. Halaman ini menampilkan: i. Daftar event yang telah diikuti/didaftari oleh Pengguna. ii. Daftar event yang perlu diberikan ulasan oleh Pengguna (event yang telah diikuti namun belum diberi ulasan). 6. Memberikan Ulasan (Testimoni) (Ekstensi dari Melihat Aktivitas Saya atau Detail Event) a. Dari halaman Aktivitas Saya (pada bagian "Events Needing Your Review") atau dari halaman Detail Event (dengan menekan tombol bintang/ulasan), Pengguna diarahkan ke halaman Formulir Ulasan (reviewForm.php) untuk event terkait. b. Sistem menampilkan informasi event yang akan diulas. c. Jika Pengguna belum memberikan ulasan, sistem menampilkan formulir untuk mengisi rating (1-5 bintang) dan teks ulasan (opsional). d. Pengguna memilih rating dan mengisi teks ulasan (jika diinginkan). e. Pengguna menekan tombol "Submit Review". f. Sistem menyimpan ulasan tersebut ke database (tabel event_reviews). g. Sistem mengarahkan Pengguna kembali ke halaman Detail Event dengan notifikasi pop-up "Your review has been posted successfully!". h. Ulasan yang telah dikirim akan muncul di bagian "Event Reviews" pada halaman Formulir Ulasan (dan bisa juga

	ditampilkan di halaman Detail Event jika fitur tersebut ditambahkan). 7. Mendaftarkan Diri sebagai Pemateri a. Pengguna (dengan peran 'user') menekan tombol "Create" di navigasi footer, yang mengarahkannya ke halaman Daftar sebagai Pemateri (speakerSignUp.php). b. Pengguna mengisi formulir pendaftaran pemateri, termasuk nama lengkap, URL LinkedIn, mengunggah file CV, dan memilih kategori keahlian. c. Pengguna menekan tombol "Sign Up". d. Sistem memproses data, menyimpan informasi tambahan dan file CV (jalur filenya) ke database, dan mengubah peran Pengguna menjadi 'speaker'. e. Sistem mengarahkan Pengguna ke halaman Beranda dengan notifikasi pop-up "Congratulations! You are now signed up as a speaker." (Catatan: Proses validasi manual oleh admin belum diimplementasikan; promosi menjadi pemateri bersifat langsung). 8. Membuat Event (Hanya bisa dilakukan oleh Pemateri) a. Pemateri (yang perannya sudah 'speaker') menekan tombol "Create" di navigasi footer, yang kini mengarahkannya ke halaman Buat Event (createPost.php). b. Pemateri mengisi detail event pada formulir, termasuk: i. Judul Event ii. Topik Event iii. Deskripsi Event iv. Mengunggah Gambar Event (opsional) v. Mengunggah Ringkasan Materi (PDF, opsional) c. Pemateri menekan tombol "Create". d. Sistem menyimpan detail event baru tersebut ke database (tabel events), termasuk menyimpan file yang diunggah dan jalur filenya. e. Sistem mengarahkan Pemateri ke halaman Beranda dengan notifikasi pop-up "Event created successfully!". f. Event yang baru dibuat akan muncul di halaman Beranda dan dapat diakses oleh Pengguna lain. (Catatan: Fitur tanggal & waktu, link webinar/lokasi belum ada di formulir saat ini).
Alternative Flow - 1:	5.c.i. (Melihat Aktivitas Saya - Events Joined): Jika Pengguna belum mendaftar event apapun, sistem menampilkan pesan "You haven't joined any events yet." 5.c.ii. (Melihat Aktivitas Saya - Events Needing Review): Jika Pengguna sudah memberi ulasan untuk semua event yang diikutinya atau belum ada event yang memenuhi syarat untuk diulas, sistem menampilkan pesan "No events awaiting your review. Great job!".

Alternative Flow - 2:	8.a. (Membuat Event): Jika Pengguna (bukan Pemateri) mencoba mengakses halaman Buat Event secara langsung (misalnya, dengan mengubah URL), sistem akan mengarahkannya ke halaman Beranda dengan pesan error error=access_denied. 8.d. (Membuat Event): • Jika Pemateri mengirimkan formulir Buat Event dengan data tidak valid (misalnya, judul atau topik kosong), sistem mengarahkan kembali ke halaman Buat Event dengan pesan error (misalnya, error=create_failed). (Validasi input di sisi server perlu diperkuat). • Jika terjadi kesalahan saat menyimpan event ke database atau mengunggah file (misalnya, karena ukuran file melebihi batas atau masalah izin), sistem mengarahkan kembali ke halaman Buat Event dengan pesan error.
Postconditions:	• Event yang dibuat oleh Pemateri akan tersedia di daftar event GrowTogether (Beranda dan Hasil Pencarian) dan dapat dilihat detailnya. • Pengguna yang mendaftar pada suatu event akan tercatat sebagai peserta event tersebut dan dapat melihatnya di halaman "Aktivitas Saya" pada bagian "Events Joined". • Ulasan (rating dan teks) yang diberikan oleh Pengguna akan tersimpan dalam database dan dapat ditampilkan pada halaman Formulir Ulasan untuk event tersebut. • File materi (gambar event, ringkasan PDF) yang diunggah oleh Pemateri saat membuat event akan tersimpan di server, dan jalurnya akan tersimpan di database, memungkinkan file tersebut diakses/diunduh. • Pengguna yang berhasil mendaftar sebagai Pemateri akan memiliki peran 'speaker' di database, dan tombol "Create" di navigasi footer akan mengarahkan mereka ke halaman pembuatan event. • Sistem menampilkan notifikasi pop-up yang sesuai setelah aksi berhasil (misalnya, mendaftar event, mengirim ulasan, mendaftar pemateri, membuat event).

Implementasi Controller

Berikut merupakan diagram yang menggambarkan alur dari proses dalam elemen MVC yang digunakan dalam pembuatan projek:



Arsitektur Model-View-Controller (MVC) pada aplikasi web ini mengatur alur kerja sebagai berikut: browser mengirimkan permintaan ke server melalui URL yang spesifik. URL ini mengindikasikan controller dan metode mana yang harus dieksekusi. Controller utama (Controller) bertugas menerima permintaan tersebut, mengelola logika aplikasi, dan menjembatani interaksi antara Model (data), Session (informasi sesi pengguna, jika ada), dan View (tampilan).

Controller induk (Controller) menyediakan fungsi dasar seperti loadModel() untuk memanggil model dan loadView()untuk menampilkan halaman. Metode loadView() menerima data dari controller dalam bentuk array dan mengekstraknya menjadi variabel individual agar mudah digunakan dalam file view. Selain itu, metode ini menyertakan file view yang sesuai berdasarkan nama yang diberikan, yang dicari di dalam direktori views/.

Halaman growTogether.php URL: http://localhost/project-pemweb/index.php?c=Todos&m=grow URL ini mengarahkan pengguna ke file index.php dengan parameter c=Todos dan m=grow. Parameter c=Todos memberitahukan sistem untuk menjalankan controller Todos, dan m=grow menginstruksikan untuk mengeksekusi metode grow() di dalam controller tersebut.

Uraian Komponen MVC:

1. **Browser:** Antarmuka pengguna tempat pengguna berinteraksi dengan aplikasi melalui URL. Setiap aksi pengguna menghasilkan permintaan HTTP yang dikirim ke server.
2. **URL:** Membawa parameter (`c` untuk Controller dan `m` untuk method) yang menentukan logika apa yang akan dieksekusi di sisi server.
3. **`index.php` (Front Controller):** Bertindak sebagai titik masuk tunggal untuk semua permintaan. File ini bertugas menganalisis parameter URL untuk menentukan Controller dan method mana yang harus dipanggil.

4. **Controller Parent (`Controller.class.php`):** Kelas dasar yang menyediakan fungsionalitas umum untuk semua controller turunan.
   Kode `Controller.class.php`:

```php
<?php
class Controller {
  protected function loadModel($model) {
    require_once 'models/' . $model . '.class.php';
    return new $model();
  }

  protected function loadView($view, $data = []) {
    // Ekstrak data agar bisa diakses sebagai variabel di view
    foreach ($data as $key => $val) {
      $$key = $val;
    }
    require_once 'views/' . $view;
  }
}
?>
```

   a. **`loadModel($model)`:** Metode ini digunakan oleh controller turunan untuk memuat instance dari kelas Model yang spesifik (misalnya, `EventModel`, `UserModel`). Ini memastikan bahwa Controller dapat berinteraksi dengan lapisan data.
   b. **`loadView($view, $data = [])`:** Metode ini bertanggung jawab untuk menampilkan halaman ke pengguna.Ia menerima nama file view dan sebuah array data. Data ini kemudian diekstrak menjadi variabel-variabel individual yang dapat diakses langsung di dalam file view, sebelum file view tersebut di-`include`.

5. **Controller Spesifik Fitur (misalnya, `Todos.class.php`, `User.class.php`):** Kelas-kelas ini mewarisi dari `Controller` parent dan berisi method-method yang menangani logika spesifik untuk setiap fitur dalam use case "Mencari dan Mengelola Acara GrowTogether". Mereka menerima input dari pengguna (melalui parameter URL atau data form), berinteraksi dengan Model untuk mengambil atau memodifikasi data, dan kemudian memilih View yang sesuai untuk menampilkan hasilnya kepada pengguna
   Model Parent (`Model.class.php`): Kelas dasar untuk semua model, bertanggung jawab untuk menginisialisasi koneksi database.
   Kode `Model.class.php`:

```php
<?php
class Model {
  protected $db;

  public function __construct() {
    $hostname = getenv('DB_HOST') ?: 'localhost';
    $username = getenv('DB_USER') ?: 'root';
    $password = getenv('DB_PASSWORD') ?: '';
```

```
        $dbname   = getenv('DB_NAME') ?: 'growlink_db';

        $this->db = new mysqli($hostname, $username, $password, $dbname);

        if ($this->db->connect_error) {
           die("Koneksi database gagal: " . $this->db->connect_error);
        }
     }
}
?>
```

a. Konstruktor kelas ini secara otomatis membuat koneksi ke database MySQL menggunakan `mysqli` saat sebuah instance Model dibuat. Detail koneksi (hostname, username, password, dbname) diambil dari environment variables (jika ada, untuk Docker) atau menggunakan nilai default.

7. **Model Spesifik Fitur**
(misalnya, `EventModel.class.php`, `UserModel.class.php`): Kelas-kelas ini mewarisi dari `Model` parent dan berinteraksi langsung dengan database untuk melakukan operasi CRUD (Create, Read, Update, Delete) terkait data spesifik

(event, pengguna, ulasan, komentar, dll.). Mereka berisi query SQL dan logika untuk memanipulasi data. **Peran Model sangat krusial karena menjadi jembatan antara logika aplikasi (Controller) dan penyimpanan data (Database), memastikan integritas dan konsistensi data.**

8. **View (File-file `.php` di folder `views/`):** Bertanggung jawab untuk presentasi data kepada pengguna. File-file ini berisi HTML dan kode PHP minimal untuk menampilkan data yang telah disiapkan oleh
Controller. Contohnya `growTogether.php`, `eventDetail.php`, `reviewForm.php`, dll.

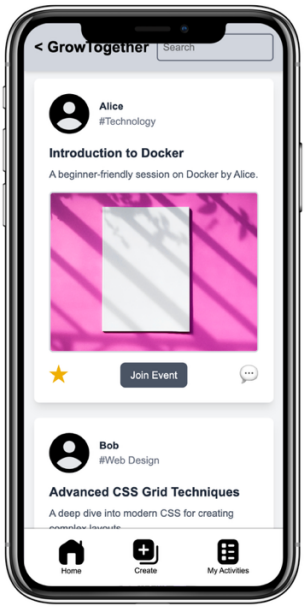
9. **Database (DB):** Tempat penyimpanan data persisten aplikasi, dikelola oleh MySQL. Model berinteraksi dengan DB untuk mengambil dan menyimpan informasi.

DDL (Data Definition Language) untuk Tabel yang Terlibat:

```
USE `growlink_db`;

CREATE TABLE `users` (
   `id` INT AUTO_INCREMENT PRIMARY KEY,
   `user_name` VARCHAR(255) NOT NULL UNIQUE,
   `password` VARCHAR(255) NOT NULL,
   `role` ENUM('user', 'speaker') NOT NULL DEFAULT 'user',
   `full_name` VARCHAR(255) NULL,
   `linkedin_url` VARCHAR(255) NULL,
   `cv_path` VARCHAR(255) NULL,
   `speaker_category` VARCHAR(100) NULL,
   `created_at` TIMESTAMP DEFAULT CURRENT_TIMESTAMP
```

```
);

CREATE TABLE `events` (
  `id` INT AUTO_INCREMENT PRIMARY KEY,
  `user_id` INT NOT NULL,
  `title` VARCHAR(255) NOT NULL,
  `topic` VARCHAR(100) DEFAULT 'General',
  `description` TEXT,
  `image_url` VARCHAR(255) NULL,
  `key_summary_path` VARCHAR(255) NULL,
  `created_at` TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (`user_id`) REFERENCES `users`(`id`) ON DELETE CASCADE
);

CREATE TABLE `event_registrations` (
  `id` INT AUTO_INCREMENT PRIMARY KEY,
  `user_id` INT NOT NULL,
  `event_id` INT NOT NULL,
  `registered_at` TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (`user_id`) REFERENCES `users`(`id`) ON DELETE CASCADE,
  FOREIGN KEY (`event_id`) REFERENCES `events`(`id`) ON DELETE CASCADE,
  UNIQUE KEY `user_event_unique` (`user_id`, `event_id`)
);

CREATE TABLE `comments` (
  `id` INT AUTO_INCREMENT PRIMARY KEY,
  `event_id` INT NOT NULL,
  `user_id` INT NOT NULL,
  `comment_text` TEXT NOT NULL,
  `created_at` TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (`event_id`) REFERENCES `events`(`id`) ON DELETE CASCADE,
  FOREIGN KEY (`user_id`) REFERENCES `users`(`id`) ON DELETE CASCADE
);

CREATE TABLE `event_reviews` (
  `id` INT AUTO_INCREMENT PRIMARY KEY,
  `event_id` INT NOT NULL,
  `user_id` INT NOT NULL,
  `rating` TINYINT NULL,
  `review_text` TEXT NULL,
  `created_at` TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (`event_id`) REFERENCES `events`(`id`) ON DELETE CASCADE,
  FOREIGN KEY (`user_id`) REFERENCES `users`(`id`) ON DELETE CASCADE,
  UNIQUE KEY `user_event_review_unique` (`user_id`, `event_id`)
);
```

Implementasi Halaman growTogether.php



- **URL:** `index.php?c=Todos&m=grow`
- **View File:** `views/growTogether.php`
- **Controller (`Todos.class.php`), Method: `grow()`**

```
function grow() {
   $currentUserId = 2; // Simulasi user login sebagai Erza
   $userModel = $this->loadModel('UserModel');
   $currentUser = $userModel->getUserById($currentUserId);
   $userRole = $currentUser ? $currentUser['role'] : 'user';

   $eventModel = $this->loadModel('EventModel');
   $events = $eventModel->getAllEvents();
   $registeredEventIds = $eventModel->getRegisteredEventIdsForUser($currentUserId);

   $this->loadView('growTogether.php', [
      'events' => $events,
      'userRole' => $userRole,
      'currentUserId' => $currentUserId,
      'registeredEventIds' => $registeredEventIds
   ]);
}
```

Method `grow()` pada `Todos` controller berfungsi sebagai pengatur utama untuk halaman beranda. Pertama, method ini mengambil informasi pengguna yang sedang aktif (disimulasikan dengan `$currentUserId`) dengan memuat `UserModel`dan memanggil `getUserById()` untuk mendapatkan peran pengguna. Selanjutnya, `EventModel` dimuat untuk mengambil semua event yang tersedia melalui method `getAllEvents()`. Selain itu, untuk menyesuaikan tampilan tombol

aksi pada setiap event (misalnya, "Join Event" atau "✓ Joined"), method ini juga memanggil `getRegisteredEventIdsForUser()` dari `EventModel` guna mendapatkan daftar ID event yang sudah diikuti oleh pengguna saat ini. Semua data yang terkumpul—daftar event, peran pengguna, ID pengguna, dan daftar ID event yang telah diikuti—kemudian diteruskan ke view `growTogether.php` untuk ditampilkan.

Model (`EventModel.class.php`)
Method: `getAllEvents()`

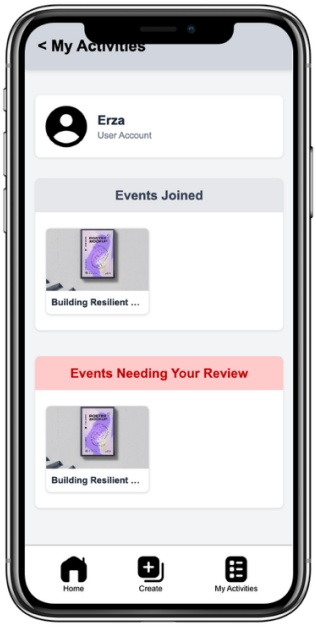
```
public function getAllEvents() {
   $sql = "SELECT events.*, users.user_name, events.topic
         FROM events
         JOIN users ON events.user_id = users.id
         ORDER BY events.created_at DESC";
   $result = $this->db->query($sql);
   return $result->fetch_all(MYSQLI_ASSOC);
}
```

Method `getAllEvents()` dalam `EventModel` bertanggung jawab untuk mengambil seluruh data event dari database. Method ini menjalankan query SQL yang melakukan `JOIN` antara tabel `events` dengan tabel `users` untuk menyertakan nama pembuat (`user_name`) dan topik dari event (`events.topic`) dalam setiap record event yang dikembalikan. Pengurutan dilakukan berdasarkan tanggal event dibuat (`created_at DESC`) sehingga event terbaru muncul di atas. Hasil query kemudian dikembalikan sebagai array asosiatif ke controller.

Method: getRegisteredEventIdsForUser($userId)

```
public function getRegisteredEventIdsForUser($userId) {
   $sql = "SELECT event_id FROM event_registrations WHERE user_id = ?";
   $stmt = $this->db->prepare($sql);
   $stmt->bind_param('i', $userId);
   $stmt->execute();
   $result = $stmt->get_result();
   $rows = $result->fetch_all(MYSQLI_ASSOC);
   return array_column($rows, 'event_id');
}
```

Implementasi Halaman registered.php



- **URL:** `index.php?c=Todos&m=registered`
- **View File:** `views/registered.php`
- **Controller (`Todos.class.php`), Method: `registered()`**

```
function registered() {
    $currentUserId = 2; //disini hardcode sebagai Erza

    $userModel = $this->loadModel('UserModel');
    $currentUser = $userModel->getUserById($currentUserId);

    $currentUsername = $currentUser ? $currentUser['user_name'] : 'Guest';
    $userRole = $currentUser ? $currentUser['role'] : 'user';

    $eventModel = $this->loadModel('EventModel');
    $registeredEvents = $eventModel->getRegisteredEventsForUser($currentUserId);
    $eventsNeedingReview = $eventModel->getEventsNeedingReview($currentUserId);

    $this->loadView('registered.php', [
        'username' => $currentUsername,
        'events' => $registeredEvents,
        'eventsNeedingReview' => $eventsNeedingReview,
        'userRole' => $userRole,
        'currentUserId' => $currentUserId
    ]);
}
```

Method registered() dalam Todos controller mengelola halaman "Aktivitas Saya". Ia mengambil ID pengguna yang sedang aktif dan memuat UserModel untuk mendapatkan detail

nama pengguna dan peran. Selanjutnya, EventModeldimuat untuk menjalankan dua fungsi utama: getRegisteredEventsForUser() mengambil semua event yang telah diikuti (didaftari) oleh pengguna, sementara getEventsNeedingReview() mengambil daftar event yang telah diikuti namun belum diberi ulasan oleh pengguna tersebut. Kedua set data event ini, bersama dengan informasi pengguna, kemudian dikirim ke view registered.php

Model (EventModel.class.php**)**
Method: getRegisteredEventsForUser($userId)

```
public function getRegisteredEventsForUser($userId) {
   $sql = "SELECT
            events.*,
            users.user_name AS event_creator_name
          FROM event_registrations
          JOIN events ON event_registrations.event_id = events.id
          JOIN users ON events.user_id = users.id
          WHERE event_registrations.user_id = ?
          ORDER BY events.created_at DESC";

   $stmt = $this->db->prepare($sql);
   $stmt->bind_param('i', $userId);
   $stmt->execute();

   $result = $stmt->get_result();
   return $result->fetch_all(MYSQLI_ASSOC);
}
```

Method getRegisteredEventsForUser() pada EventModel bertanggung jawab untuk mengambil daftar event yang telah diikuti oleh pengguna. Model melakukan query SQL yang menggabungkan (JOIN) tabel event_registrations, events, dan users (untuk mendapatkan nama pembuat event asli) berdasarkan user_id pengguna yang sedang aktif. Data ini kemudian dikembalikan sebagai array untuk ditampilkan.

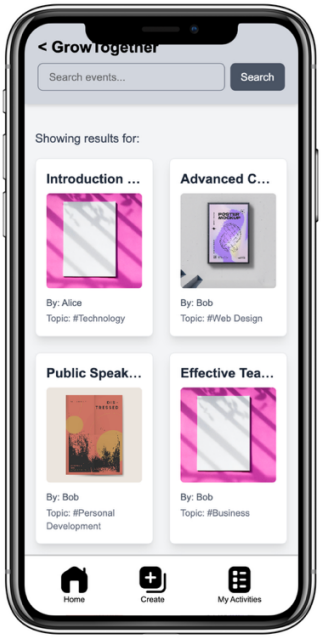
Method: getEventsNeedingReview($userId)

```
public function getEventsNeedingReview($userId) {
   $sql = "SELECT e.*, u.user_name
         FROM event_registrations er
         JOIN events e ON er.event_id = e.id
         JOIN users u ON e.user_id = u.id
         LEFT JOIN event_reviews rev ON er.event_id = rev.event_id AND er.user_id = rev.user_id
         WHERE er.user_id = ? AND rev.id IS NULL
         ORDER BY e.created_at DESC";

   $stmt = $this->db->prepare($sql);
   $stmt->bind_param('i', $userId);
   $stmt->execute();
   $result = $stmt->get_result();
   return $result->fetch_all(MYSQLI_ASSOC);
}
```

Method getEventsNeedingReview() memiliki peran penting dalam mengidentifikasi event mana yang bisa diulas oleh pengguna. Model menjalankan query SQL yang kompleks: pertama, ia mencari semua event yang diikuti pengguna dari event_registrations (di-JOIN dengan events dan users). Kemudian, ia menggunakan LEFT JOINke tabel event_reviews dan memfilter hasilnya untuk hanya menyertakan event di mana tidak ada ulasan yang cocok (rev.id IS NULL) dari pengguna tersebut. Ini memastikan hanya event yang belum diulas yang ditampilkan.

View (registered.php) View registered.php menerima data pengguna ($username, $userRole, $currentUserId), daftar event yang telah diikuti ($registeredEvents), dan daftar event yang perlu diulas ($eventsNeedingReview). View ini menampilkan informasi profil pengguna, kemudian menyajikan dua bagian terpisah: "Events Joined" yang menampilkan semua event yang diikuti pengguna, dan "Events Needing Your Review" yang menampilkan event yang dapat diulas. Setiap event dalam bagian "Need Review" memiliki link yang mengarahkan pengguna ke formulir ulasan untuk event tersebut.

Implementasi Halaman searchResults.php



- **URL:** `index.php?c=Todos&m=search` (dengan data pencarian dikirim melalui metode POST).
- **View File:** `views/searchResults.php`
- **Controller (`Todos.class.php`), Method: `search()`**

```
function search() {
   $currentUserId = 2; // Simulasi user login
   $userModel = $this->loadModel('UserModel');
   $currentUser = $userModel->getUserById($currentUserId);
   $userRole = $currentUser ? $currentUser['role'] : 'user';

   $query = $_POST['query'] ?? '';
   $eventModel = $this->loadModel('EventModel');
   $events = $eventModel->searchEvents($query);

   $this->loadView('searchResults.php', [
      'events' => $events,
      'userRole' => $userRole,
      'currentUserId' => $currentUserId,
   ]);
}
```

Method `search()` di `Todos` controller menangani permintaan pencarian event. Ia mengambil kata kunci pencarian dari `$_POST['query']`. Sama seperti method `grow()`, ia mengambil informasi pengguna saat ini untuk menentukan `$userRole`dan `$currentUserId` yang

mungkin diperlukan oleh view (misalnya untuk footer). Controller kemudian memuat `EventModel` dan memanggil method `searchEvents()` dengan query pengguna. Array hasil event yang dikembalikan oleh Model kemudian dikirim ke view `searchResults.php`.

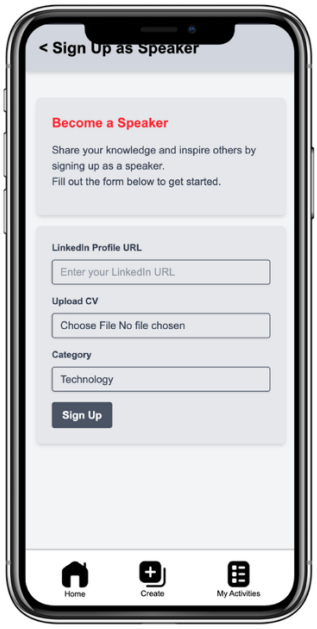
Model (EventModel.class.php), Method: searchEvents($query):

```
public function searchEvents($query) {
   $sql = "SELECT
            events.*,
            users.user_name
         FROM events
         JOIN users ON events.user_id = users.id
         WHERE events.title LIKE ? OR events.description LIKE ? OR users.user_name LIKE ? OR events.topic
LIKE ?
         ORDER BY events.created_at DESC";

   $stmt = $this->db->prepare($sql);
   $searchTerm = '%' . $query . '%';
   $stmt->bind_param('ssss', $searchTerm, $searchTerm, $searchTerm, $searchTerm);
   $stmt->execute();

   $result = $stmt->get_result();
   return $result->fetch_all(MYSQLI_ASSOC);
}
```

Method `searchEvents()` di `EventModel` memiliki peran krusial untuk berinteraksi dengan database guna menemukan event yang relevan. Ia menerima sebuah kata kunci `$query` dan membuat query SQL yang mencari kecocokan pada kolom judul event, deskripsi event, nama pembuat event (melalui `JOIN` dengan tabel `users`), dan topik event. Penggunaan prepared statement (`bind_param`) memastikan keamanan dari SQL injection. Hasilnya, berupa array event yang cocok, dikembalikan ke controller.

View (`searchResults.php`) View `searchResults.php` bertugas menampilkan event-event yang cocok dengan kriteria pencarian pengguna. Ia menerima data `$events` (hasil pencarian) dan `$userRole` (untuk footer dinamis). View ini menampilkan kata kunci pencarian dan kemudian melakukan iterasi pada array `$events` untuk menampilkan setiap event dalam format kartu ringkas, yang masing-masing memiliki link ke halaman detail event. Jika tidak ada event yang ditemukan, sebuah pesan akan ditampilkan.

Implementasi Halaman speakerSignUp.php



- **URL Akses Form:** `index.php?c=Todos&m=signUp` (diakses melalui tombol "Create" di footer jika pengguna bukan 'speaker')
- **URL Aksi Form:** `index.php?c=User&m=applyAsSpeaker` (method POST)
- **View File (Form):** `views/speakerSignUp.php`

Controller (Todos.class.php**), Method:** signUp()

```
function signUp() {
   $this->loadView('speakerSignUp.php');
}
```

Method `signUp()` pada `Todos` controller bertugas untuk menampilkan halaman `speakerSignUp.php` yang berisi formulir bagi pengguna untuk mendaftar sebagai pemateri.

Controller (User.class.php**), Method:** applyAsSpeaker()

```
function applyAsSpeaker() {
   $userId = 2;

   $fullName = $_POST['user-fullname'] ?? '';
   $linkedinUrl = $_POST['linkedin-url'] ?? '';
   $category = $_POST['category'] ?? '';
   $cvFile = $_FILES['cv'] ?? null;
   $cvPath = '';

   $userModelForFile = $this->loadModel('UserModel');
```

```
    $applyingUser = $userModelForFile->getUserById($userId);
    $userNameForFile = 'unknown_user';
    if ($applyingUser) {
        $userNameForFile = strtolower($applyingUser['user_name']);
        $userNameForFile = preg_replace('/[^a-z0-9_-]+/', '-', $userNameForFile);
        $userNameForFile = trim($userNameForFile, '-');
    }
    $dateUploaded = date('Ymd');

    if ($cvFile && $cvFile['error'] === UPLOAD_ERR_OK) {
        $uploadDir = 'uploads/cvs/';
        if (!is_dir($uploadDir)) {
            mkdir($uploadDir, 0755, true);
        }
        $fileExtension = pathinfo($cvFile['name'], PATHINFO_EXTENSION);
        $newFilename = "cv_" . $userId . "_" . $userNameForFile . "_" . $dateUploaded . "." . $fileExtension;
        $cvPath = $uploadDir . $newFilename;
        if (!move_uploaded_file($cvFile['tmp_name'], $cvPath)) {
            $cvPath = '';
        }
    }

    $userModel = $this->loadModel('UserModel');
    $success = $userModel->updateSpeakerApplication(
        $userId,
        $fullName,
        $linkedinUrl,
        $cvPath,
        $category
    );

    if ($success) {
        header('Location: ?c=Todos&m=grow&status=speaker_signup_success');
    } else {
        header('Location: ?c=Todos&m=signUp&error=speaker_apply_failed');
    }
    exit();
}
```

Method `applyAsSpeaker()` pada `User` controller menangani data yang dikirim dari formulir `speakerSignUp.php`. Ia mengambil data pengguna dari `$_POST` dan file CV dari `$_FILES`. Controller ini juga memuat `UserModel` untuk mengambil nama pengguna yang akan digunakan dalam pembuatan nama file CV yang unik. Proses unggah file CV, termasuk pembuatan nama file dan pemindahan file ke direktori `uploads/cvs/`, ditangani di sini. Path file CV yang berhasil diunggah beserta data formulir lainnya kemudian diteruskan ke method `updateSpeakerApplication()` pada `UserModel`. Tergantung status keberhasilan dari Model, pengguna akan diarahkan ke halaman Beranda dengan notifikasi sukses atau kembali ke halaman pendaftaran dengan notifikasi error.

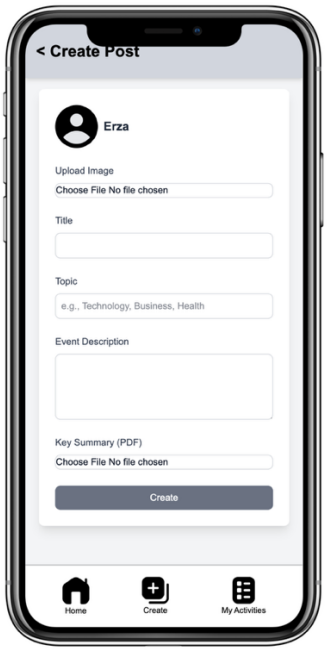
Model (UserModel.class.php**), Method:** updateSpeakerApplication()

```
public function updateSpeakerApplication($userId, $fullName, $linkedinUrl, $cvPath, $category) {
   $sql = "UPDATE users
         SET
            role = 'speaker',
            full_name = ?,
            linkedin_url = ?,
            cv_path = ?,
            speaker_category = ?
         WHERE id = ?";
   $stmt = $this->db->prepare($sql);
   $stmt->bind_param('ssssi', $fullName, $linkedinUrl, $cvPath, $category, $userId);
   return $stmt->execute();
}
```

Method updateSpeakerApplication() dalam UserModel adalah inti dari proses pendaftaran pemateri. Model ini bertanggung jawab untuk memperbarui data pengguna di tabel users pada database. Ia mengubah kolom rolepengguna menjadi 'speaker' dan menyimpan informasi tambahan seperti full_name, linkedin_url, cv_path (jalur file CV), dan speaker_category. Penggunaan prepared statement memastikan keamanan data.

View (speakerSignUp.php) View speakerSignUp.php menyediakan antarmuka formulir HTML bagi pengguna untuk memasukkan detail pendaftaran sebagai pemateri, seperti nama lengkap, URL profil LinkedIn, kategori keahlian, dan mengunggah file CV. Formulir ini diatur untuk mengirimkan datanya ke method applyAsSpeaker() pada User controller menggunakan metode POST dan enctype="multipart/form-data" untuk mendukung unggahan file.

Implementasi Halaman createPost.php



- **URL Akses Form:** `index.php?c=Todos&m=create` (hanya untuk peran 'speaker')
- **URL Aksi Form:** `index.php?c=Todos&m=store` (method POST)
- **View File (Form):** `views/createPost.php`

Controller (Todos.class.php**), Method:** create()

```
function create() {
   $userId = 1; // Simulasi user login Joe (ID 1, speaker). Sesuaikan dengan user ID pemateri.
   $userModel = $this->loadModel('UserModel');
   $user = $userModel->getUserById($userId);

   if ($user && $user['role'] === 'speaker') {
      $this->loadView('createPost.php', ['user' => $user, 'userRole' => $user['role']]);
   } else {
      header('Location: ?c=Todos&m=grow&error=access_denied');
      exit();
   }
}
```

Method `create()` pada `Todos` controller bertugas menampilkan formulir pembuatan event. Ia memverifikasi peran pengguna melalui `UserModel`; jika pengguna adalah 'speaker', data pengguna (termasuk peran untuk logika footer dinamis) diteruskan ke view `createPost.php`. Pengguna yang bukan 'speaker' akan dialihkan.

Controller (Todos.class.php), Method: store()

```
function store() {
    $userId = 2;

    $userModelForFile = $this->loadModel('UserModel');
    $eventCreatorUser = $userModelForFile->getUserById($userId);
    $userNameForFile = 'unknown_user';
    if ($eventCreatorUser) {
        $userNameForFile = strtolower($eventCreatorUser['user_name']);
        $userNameForFile = preg_replace('/[^a-z0-9_-]+/', '-', $userNameForFile);
        $userNameForFile = trim($userNameForFile, '-');
    }

    $title = $_POST['title'] ?? '';
    $topic = $_POST['topic'] ?? 'General';
    $description = $_POST['description'] ?? '';
    $imageFile = $_FILES['image'] ?? null;
    $keySumFile = $_FILES['keySum'] ?? null;

    $uploadFile = function($file, $subDir, $baseNamePrefix = 'file') use ($userId, $userNameForFile) {
        if ($file && $file['error'] === UPLOAD_ERR_OK) {
            $baseUploadDir = 'uploads/';
            $uploadDir = $baseUploadDir . $subDir . '/';
            if (!is_dir($uploadDir)) {
                mkdir($uploadDir, 0755, true);
            }
            $fileExtension = pathinfo($file['name'], PATHINFO_EXTENSION);
            $timestamp = date('YmdHis');
            $uniqueComponent = uniqid();
            $newFilename = $baseNamePrefix . "_" . $userId . "_" . $userNameForFile . "_" . $timestamp . "_" .
$uniqueComponent . "." . $fileExtension;
            $filePath = $uploadDir . $newFilename;
            if (move_uploaded_file($file['tmp_name'], $filePath)) {
                return $filePath;
            }
        }
        return '';
    };

    $imagePath = $uploadFile($imageFile, 'images', 'eventimg');
    $keySumPath = $uploadFile($keySumFile, 'pdfs', 'keysum');

    $eventModel = $this->loadModel('EventModel');
    $success = $eventModel->createEvent($userId, $title, $topic, $description, $imagePath, $keySumPath);

    if ($success) {
        header('Location: ?c=Todos&m=grow&status=event_created');
    } else {
```

```
        header('Location: ?c=Todos&m=create&error=create_failed');
    }
    exit();
}
```

Method `store()` dalam `Todos` controller bertanggung jawab untuk memproses data yang dikirim dari formulir `createPost.php`. Ia mengambil detail event dari `$_POST` dan file yang diunggah dari `$_FILES`. Sebelum memproses unggahan file, ia mengambil data pengguna (pembuat event) dari `UserModel` untuk digunakan dalam pembuatan nama file yang unik dan deskriptif. Terdapat fungsi anonim `$uploadFile` yang menangani logika unggahan file secara umum, termasuk pembuatan nama file unik (menggunakan ID pengguna, nama pengguna, timestamp lengkap, dan `uniqid()`), pembuatan direktori tujuan (`uploads/images/` atau `uploads/pdfs/`) jika belum ada, dan pemindahan file ke server. Setelah path file yang berhasil diunggah didapatkan, Controller memanggil method `createEvent()` pada `EventModel` untuk menyimpan semua informasi event baru, termasuk path file, ke dalam database. Terakhir, pengguna diarahkan kembali ke halaman Beranda dengan parameter status yang mengindikasikan keberhasilan atau kegagalan pembuatan event.

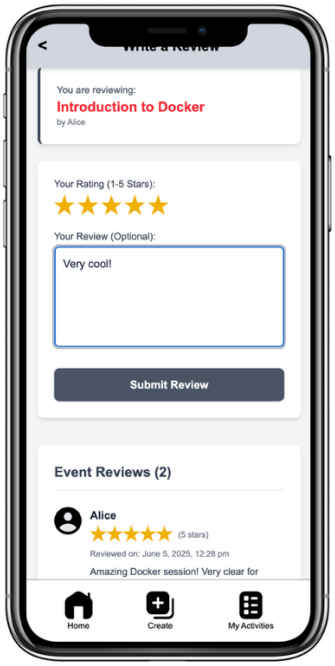
Model (EventModel.class.php**), Method:** createEvent()

```
public function createEvent($userId, $title, $topic, $description, $imageUrl, $keySumPath) {
    $sql = "INSERT INTO events (user_id, title, topic, description, image_url, key_summary_path)
            VALUES (?, ?, ?, ?, ?, ?)";
    $stmt = $this->db->prepare($sql);
    // 'isssss' -> integer, string, string, string, string, string
    $stmt->bind_param('isssss', $userId, $title, $topic, $description, $imageUrl, $keySumPath);
    return $stmt->execute();
}
```

Method createEvent() dalam EventModel adalah komponen kunci yang berinteraksi langsung dengan database untuk menyimpan event baru. Model menerima semua detail event yang telah diproses dan disiapkan oleh Controller (termasuk ID pengguna pembuat, judul, topik, deskripsi, serta path untuk gambar event dan file ringkasan). Method ini kemudian menjalankan perintah SQL INSERT ke dalam tabel events menggunakan prepared statement untuk memastikan keamanan dari SQL injection dan integritas data. Keberhasilan operasi penyimpanan data ini dikembalikan ke Controller dalam bentuk boolean.

View (createPost.php) View createPost.php menyediakan antarmuka formulir HTML yang lengkap dan terstruktur bagi pemateri untuk memasukkan semua detail yang diperlukan saat membuat event baru. Ini mencakup input untuk judul, topik, deskripsi event, serta field untuk mengunggah file gambar event dan file PDF ringkasan materi. Nama pemateri yang sedang login juga ditampilkan di bagian atas formulir untuk memberikan konteks. Formulir ini dirancang untuk mengirimkan datanya menggunakan metode POST ke method store() pada Todoscontroller dan menggunakan enctype="multipart/form-data" untuk memastikan fungsionalitas unggahan file berjalan dengan benar. Terdapat juga footer navigasi standar.

Implementasi Halaman reviewForm.php



- **URL Akses Form:** `index.php?c=Todos&m=showReviewForm&event_id={event_id}`
- **URL Aksi Form:** `index.php?c=Todos&m=storeReview` (method POST)
- **View File (Form & Daftar Ulasan):** `views/reviewForm.php`
- **Controller (`Todos.class.php`), Method: `showReviewForm()`**

```
function showReviewForm() {
    $eventId = $_GET['event_id'] ?? 0;
    if ($eventId <= 0) {
        header('Location: ?c=Todos&m=grow&error=invalid_event_for_review');
        exit();
    }

    $currentUserId = 2; // Simulasi user login
    $userModel = $this->loadModel('UserModel');
    $currentUser = $userModel->getUserById($currentUserId);
    $userRole = $currentUser ? $currentUser['role'] : 'user';

    $eventModel = $this->loadModel('EventModel');
    $eventDetails = $eventModel->getEventById($eventId);

    if (!$eventDetails) {
```

```
        header('Location: ?c=Todos&m=grow&error=event_not_found_for_review');
        exit();
    }

    $hasReviewed = $eventModel->hasUserReviewedEvent($currentUserId, $eventId);
    $reviews = $eventModel->getReviewsByEventId($eventId);

    $this->loadView('reviewForm.php', [
        'event' => $eventDetails,
        'userRole' => $userRole,
        'currentUserId' => $currentUserId,
        'hasReviewed' => $hasReviewed,
        'reviews' => $reviews
    ]);
}
```

Method `showReviewForm()` pada `Todos` controller menangani tampilan halaman untuk memberikan ulasan. Ia mengambil `event_id` dari URL, memuat detail event terkait dan semua ulasan yang sudah ada untuk event tersebut menggunakan `EventModel`. Controller juga memeriksa apakah pengguna saat ini sudah pernah memberikan ulasan untuk event ini melalui method `hasUserReviewedEvent()` dari `EventModel`. Semua data ini (detail event, ulasan yang ada, status ulasan pengguna, dan info pengguna) kemudian diteruskan ke view `reviewForm.php`.

Controller (`Todos.class.php`), Method: `storeReview()`

```
function storeReview() {
    $userId = $_POST['user_id'] ?? 0; // Idealnya dari sesi
    $eventId = $_POST['event_id'] ?? 0;
    $rating = $_POST['rating'] ?? null;
    $reviewText = trim($_POST['review_text'] ?? '');

    if ($userId <= 0 || $eventId <= 0 || $rating === null || $rating < 1 || $rating > 5) {
        header('Location: ?c=Todos&m=showReviewForm&event_id=' . $eventId .
'&error=invalid_review_data');
        exit();
    }

    $eventModel = $this->loadModel('EventModel');
    $success = $eventModel->addReview($eventId, $userId, $rating, $reviewText);

    if ($success) {
        header('Location: ?c=Todos&m=showEvent&id=' . $eventId . '&status=review_added');
    } else {
        header('Location: ?c=Todos&m=showReviewForm&event_id=' . $eventId .
'&error=review_failed_or_exists');
    }
    exit();
}
```

Method `storeReview()` memproses data yang dikirim dari formulir ulasan. Ia menerima `event_id`, `user_id`, `rating`, dan `review_text` dari `$_POST`. Setelah validasi dasar, Controller memanggil method `addReview()` pada `EventModel` untuk menyimpan ulasan baru ke database. Pengguna kemudian diarahkan kembali ke halaman detail event atau halaman ulasan dengan status yang sesuai.

Model (EventModel.class.php)
Method: addReview($eventId, $userId, $rating, $reviewText)

```
public function addReview($eventId, $userId, $rating, $reviewText) {
   if ($this->hasUserReviewedEvent($userId, $eventId)) {
      return false;
   }
   $sql = "INSERT INTO event_reviews (event_id, user_id, rating, review_text) VALUES (?, ?, ?, ?)";
   $stmt = $this->db->prepare($sql);
   $stmt->bind_param('iiis', $eventId, $userId, $rating, $reviewText);
   return $stmt->execute();
}
```

Method `addReview()` dalam `EventModel` bertugas menyimpan ulasan baru ke tabel `event_reviews`. Sebelum menyimpan, ia memanggil `hasUserReviewedEvent()` untuk memastikan pengguna belum pernah mengulas event yang sama, guna menjaga integritas data sesuai constraint `UNIQUE KEY` di database.

Method: hasUserReviewedEvent($userId, $eventId)

```
public function hasUserReviewedEvent($userId, $eventId) {
   $sql = "SELECT id FROM event_reviews WHERE user_id = ? AND event_id = ? LIMIT 1";
   $stmt = $this->db->prepare($sql);
   $stmt->bind_param('ii', $userId, $eventId);
   $stmt->execute();
   $result = $stmt->get_result();
   return $result->num_rows > 0;
}
```

Method `hasUserReviewedEvent()` adalah fungsi utilitas di Model yang memeriksa apakah sudah ada ulasan dari pengguna tertentu untuk event tertentu di tabel `event_reviews`.

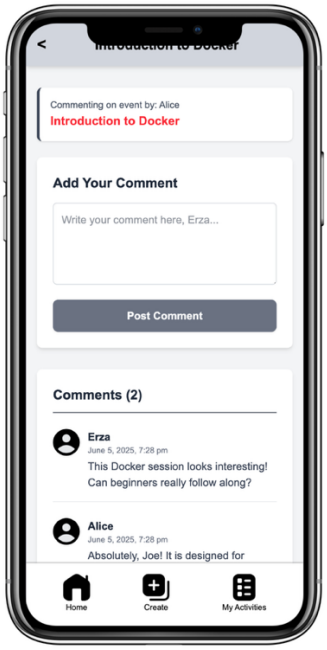
```
public function getReviewsByEventId($eventId) {
   $sql = "SELECT er.*, u.user_name
         FROM event_reviews er
         JOIN users u ON er.user_id = u.id
         WHERE er.event_id = ?
         ORDER BY er.created_at DESC";
   $stmt = $this->db->prepare($sql);
   $stmt->bind_param('i', $eventId);
```

```
    $stmt->execute();
    $result = $stmt->get_result();
    return $result->fetch_all(MYSQLI_ASSOC);
}
```

Method getReviewsByEventId() mengambil semua ulasan untuk suatu event, melakukan JOIN dengan tabel usersuntuk mendapatkan nama pengguna yang memberikan ulasan. Model ini mengagregasi data dari dua tabel untuk menyediakan informasi ulasan yang lengkap kepada Controller.

View (`reviewForm.php`) View `reviewForm.php` menampilkan informasi singkat tentang event yang diulas. Jika pengguna belum memberikan ulasan (`!$hasReviewed`), formulir input rating (bintang interaktif) dan teks ulasan akan ditampilkan.Jika sudah, pesan konfirmasi yang akan muncul. Di bawah bagian formulir/pesan tersebut, view ini juga menampilkan daftar semua ulasan yang sudah ada untuk event tersebut, yang diambil dari variabel `$reviews`. Setiap ulasan menampilkan nama pengulas, rating bintang, tanggal, dan teks ulasannya.

Implementasi Halaman eventComments.php



- **URL Akses Halaman**

Komentar: `index.php?c=Todos&m=showComments&event_id={event_id}`

- **URL Aksi Form Komentar:** `index.php?c=Todos&m=storeComment` (method POST)
- **View File (Form & Daftar Komentar):** `views/eventComments.php`

Controller (Todos.class.php**), Method:** showComments()

```
function showComments() {
    $eventId = $_GET['event_id'] ?? 0;
    if ($eventId <= 0) {
        header('Location: ?c=Todos&m=grow&error=invalid_event_id_for_comments');
        exit();
    }

    $currentUserId = 2;
    $userModel = $this->loadModel('UserModel');
    $currentUser = $userModel->getUserById($currentUserId);
    $userRole = $currentUser ? $currentUser['role'] : 'user';
    $currentUserName = $currentUser ? $currentUser['user_name'] : 'Guest';

    $eventModel = $this->loadModel('EventModel');
    $eventDetails = $eventModel->getEventById($eventId);
    if (!$eventDetails) {
        header('Location: ?c=Todos&m=grow&error=event_not_found_for_comments');
        exit();
    }

    $comments = $eventModel->getCommentsByEventId($eventId);

    $this->loadView('eventComments.php', [
        'event' => $eventDetails,
        'comments' => $comments,
        'userRole' => $userRole,
        'currentUserId' => $currentUserId,
        'currentUserName' => $currentUserName
    ]);
}
```

Method `showComments()` pada `Todos` controller mempersiapkan data untuk halaman komentar. Ia mengambil `event_id`dari URL, memuat detail event menggunakan `EventModel` (method `getEventById()`), dan mengambil semua komentar yang terkait dengan event tersebut melalui method `getCommentsByEventId()` dari `EventModel`. Informasi pengguna yang sedang aktif juga diambil untuk digunakan di formulir komentar dan footer. Semua data ini diteruskan ke view `eventComments.php`.

Controller (Todos.class.php**), Method:** storeComment()

```
function storeComment() {
    $userId = $_POST['user_id'] ?? 0;
    $eventId = $_POST['event_id'] ?? 0;
    $commentText = trim($_POST['comment_text'] ?? '');

    if ($userId > 0 && $eventId > 0 && !empty($commentText)) {
        $eventModel = $this->loadModel('EventModel');
        $success = $eventModel->addComment($eventId, $userId, $commentText);
```

```
        if ($success) {
            header('Location: ?c=Todos&m=showComments&event_id=' . $eventId . '&status=comment_added');
            exit();
        }
    }
    header('Location: ?c=Todos&m=showComments&event_id=' . $eventId . '&error=comment_failed');
    exit();
}
```

Method `storeComment()` menangani pengiriman formulir komentar baru. Ia mengambil `event_id`, `user_id`, dan `comment_text` dari `$_POST`. Setelah validasi dasar, ia memanggil method `addComment()` pada `EventModel` untuk menyimpan komentar ke database. Pengguna kemudian diarahkan kembali ke halaman komentar untuk event yang sama, dengan parameter status yang menandakan keberhasilan atau kegagalan.

Model (EventModel.class.php**)**
Method: addComment($eventId, $userId, $commentText)

```
public function addComment($eventId, $userId, $commentText) {
    $sql = "INSERT INTO comments (event_id, user_id, comment_text) VALUES (?, ?, ?)";
    $stmt = $this->db->prepare($sql);
    $stmt->bind_param('iis', $eventId, $userId, $commentText);
    return $stmt->execute();
}
```

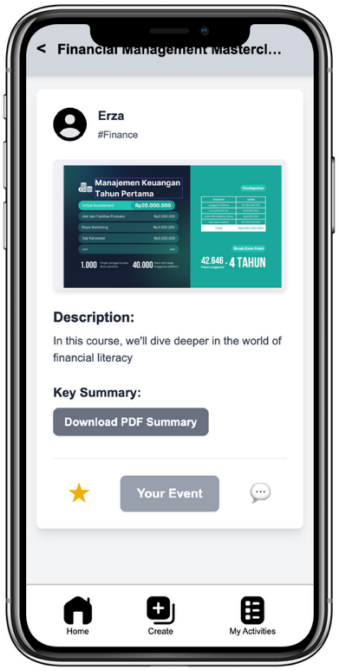
Method `addComment()` di `EventModel` bertanggung jawab untuk menyimpan data komentar baru ke tabel `comments`di database. **Model ini menjalankan perintah SQL `INSERT` menggunakan prepared statement**, memastikan data dimasukkan dengan aman dan efisien.
Method: `getCommentsByEventId($eventId)`

```
public function getCommentsByEventId($eventId) {
    $sql = "SELECT comments.*, users.user_name
            FROM comments
            JOIN users ON comments.user_id = users.id
            WHERE comments.event_id = ?
            ORDER BY comments.created_at ASC";
    $stmt = $this->db->prepare($sql);
    $stmt->bind_param('i', $eventId);
    $stmt->execute();
    $result = $stmt->get_result();
    return $result->fetch_all(MYSQLI_ASSOC);
}
```

Method `getCommentsByEventId()` bertugas mengambil semua komentar untuk suatu event. **Model ini melakukan `JOIN` antara tabel `comments` dan `users` untuk menyertakan nama pengguna yang membuat setiap komentar.**Hasilnya diurutkan berdasarkan tanggal pembuatan dan dikembalikan sebagai array ke Controller.

View (`eventComments.php`) View `eventComments.php` dirancang untuk menampilkan interaksi komentar pada sebuah event. Bagian atas view menampilkan informasi singkat tentang event yang sedang dikomentari. Di bawahnya, terdapat formulir sederhana bagi pengguna untuk menulis dan mengirim komentar baru. Bagian utama halaman kemudian menampilkan daftar semua komentar yang sudah ada untuk event tersebut, di mana setiap komentar menunjukkan foto profil (statis), nama pengguna yang berkomentar, tanggal komentar, dan isi teks komentar.

Implementasi Halaman eventDetail.php



- **URL:** `index.php?c=Todos&m=showEvent&id={event_id}`
- **View File:** `views/eventDetail.php`
- **Controller (`Todos.class.php`), Method: `showEvent()`**

```
function showEvent() {
   $eventId = $_GET['id'] ?? 0;
   if ($eventId <= 0) {
      header('Location: ?c=Todos&m=grow&error=invalid_event_id');
      exit();
   }

   $currentUserId = 2;
   $userModel = $this->loadModel('UserModel');
   $currentUser = $userModel->getUserById($currentUserId);
   $userRole = $currentUser ? $currentUser['role'] : 'user';
```

```
    $eventModel = $this->loadModel('EventModel');
    $eventDetails = $eventModel->getEventById($eventId);

    if (!$eventDetails) {
        header('Location: ?c=Todos&m=grow&error=event_not_found');
        exit();
    }

    $registeredEventIds = [];
    if ($currentUserId && $eventModel) {
        $registeredEventIds = $eventModel->getRegisteredEventIdsForUser($currentUserId);
    }

    $this->loadView('eventDetail.php', [
        'event' => $eventDetails,
        'userRole' => $userRole,
        'currentUserId' => $currentUserId,
        'registeredEventIds' => $registeredEventIds
    ]);
}
```

Method `showEvent()` pada `Todos` controller bertugas menangani permintaan untuk menampilkan halaman detail dari sebuah event. Pertama, ia mengambil `event_id` dari parameter URL. Kemudian, ia mengambil informasi pengguna yang sedang aktif (ID dan peran) menggunakan `UserModel`. Selanjutnya, `EventModel` dimuat, dan method `getEventById()` dipanggil untuk mengambil detail lengkap dari event yang bersangkutan. Jika event tidak ditemukan, pengguna akan diarahkan kembali ke halaman utama. Method ini juga memanggil `getRegisteredEventIdsForUser()` dari `EventModel` untuk menentukan status partisipasi pengguna pada event tersebut, yang berguna untuk logika tampilan tombol aksi.Seluruh data yang terkumpul (detail event, informasi pengguna, dan status partisipasi) kemudian diteruskan ke view `eventDetail.php`.

Model (`EventModel.class.php`), Method: `getEventById($eventId)`

```
public  function getEventById($eventId) {
        $sql  =  "SELECT
            events.*,
            users.user_name
          FROM events
          JOIN users ON events.user_id = users.id
          WHERE events.id = ?
          LIMIT 1";

        $stmt  =  $this->db->prepare($sql);
        $stmt->bind_param('i',  $eventId);
        $stmt->execute();
```

```
    $result = $stmt->get_result();
    if ($result->num_rows === 1) {
        return $result->fetch_assoc();
    }
    return null;
}
```

Method `getEventById()` dalam `EventModel` bertanggung jawab untuk mengambil satu record event spesifik dari database berdasarkan `eventId`. Query SQL yang dijalankan melakukan `JOIN` dengan tabel `users` untuk menyertakan nama pembuat event (`user_name`). Penggunaan prepared statement memastikan keamanan dan efisiensi. Jika event ditemukan, data event dikembalikan sebagai array asosiatif; jika tidak, `null` dikembalikan.

View (`eventDetail.php`) View `eventDetail.php` menerima data detail satu event (`$event`), informasi pengguna saat ini (`$userRole`, `$currentUserId`), dan daftar ID event yang telah diikuti pengguna (`$registeredEventIds`). View ini kemudian menampilkan semua informasi event secara komprehensif, termasuk judul, gambar event yang lebih besar,deskripsi lengkap, nama pembuat, topik event, dan link untuk mengunduh ringkasan materi (jika tersedia). Tombol-tombol aksi seperti "Join Event", "Your Event", atau "✓ Joined" ditampilkan secara kondisional berdasarkan `$currentUserId` dan `$registeredEventIds`. Halaman ini juga menyertakan footer standar dan fungsionalitas pop-up notifikasi.

Use Case: Mengelola Templates GrowHub

Penyusun Use Case: Ailsa Zahra Sahdana - 235150401111017

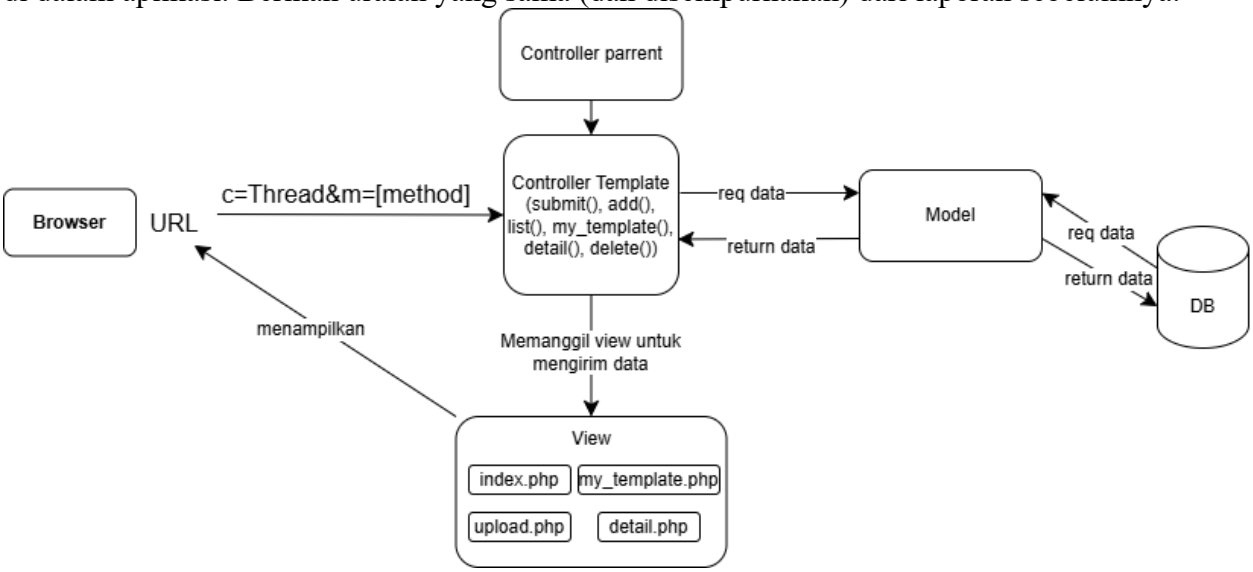
Use case ini dibuat untuk menjelaskan bagaimana pengguna dapat dengan mudah mencari, menyimpan, dan mengunggah template di GrowHub. Pengguna bisa memasukkan kata kunci untuk mencari template yang diinginkan dan melihat detailnya. Jika tidak menemukan template yang sesuai, mereka juga dapat mengunggah template baru agar bisa digunakan oleh pengguna lain. Berikut detail dari use case tersebut:

Actor:	Pengguna.
Description:	Pengguna mencari *template* berdasarkan kata kunci, melihat daftar *template* yang tersedia, mengunggah *template* baru, dan mengunduh *template* yang disukai.
Use Case Goal:	Pengguna mengelola file template dalam *platform* GrowLink.
Preconditions:	Pengguna telah masuk pada *platform* GrowLink.
Main Flow:	Pengguna membuka fitur GrowHub. Pengguna melihat daftar *template* yang tersedia. Pengguna mengetikkan kata kunci pada pencarian untuk mencari *template*. Sistem menampilkan hasil pencarian berdasarkan kata kunci. Pengguna memilih salah satu *template* untuk melihat detail filenya. Pengguna mengunduh file tersebut dengan menekan tombol unduh. Pengguna mengunggah template baru dengan menekan tombol +. Pengguna memasukkan file, judul,dan kategori *template* pada halaman New Template. Pengguna menekan tombol *post* untuk mengunggah *template*. Pengguna melihat template yang telah diunggah di halaman My Template. Pengguna menghapus template yang mereka unggah sendiri pada halaman My Template, saat template yang diunggah salah.
Alternative Flow - 1:	9.a Sistem menampilkan pesan kesalahan dan meminta pengguna mengunggah ulang template jika terdapat kesalahan. 9.b Pengguna mencoba mengisi ulang form unggah *template*.
Alternative Flow - 2:	11.a Pengguna memilih untuk membatalkan menghapus *template*. 5.a.1 Sistem tidak akan menghapus *template* dan kembali ke halaman sebelumnya.

Postconditions:	*Template* yang diunggah akan tersimpan di halaman My Template. Template yang diunggah tersedia untuk pengguna lain di halaman utama GrowHub.

Implementasi Controller dan Model

Gambarkan diagram blok yang memperlihatkan alur proses dan elemen-elemen MVC yang terlibat di dalamnya untuk menampilkan setiap halaman web dari aplikasi yang dirancang di dalam aplikasi. Berikan uraian yang sama (dan disempurnakan) dari laporan sebelumnya.



Pada diagram diatas, setiap komponen memiliki fungsi dan perannya masing masing dalam arsitektur aplikasi web berbasis MVC (Model-View-Controller). Browser berperan sebagai antarmuka pengguna untuk mengakses aplikasi melalui URL tertentu. URL ini membawa parameter yang menunjukkan controller dan method apa yang harus dijalankan oleh server. Controller berfungsi untuk menerima requesrt dari browser, mengatur logika dari aplikasi, dan juga menertukan interaksi anrata Model, Session, dan View. Controller yang digunakan yaitu Controller parent dan Controller Template yang didalamnya memuat method submit(), add(), list(), my_template(), detail(), delete(). Model adalah komponen yang berfungsi untuk mengakses database, baik untuk membaca maupun menyimpan data. View berfungsi untuk membuat dan mengirimkan tampilan (HTML) yang akan diterima oleh browser. File view yang digunakan untuk menampilkan informasi terkait projek, yang pada bagian ini fokusnya pada template. File tersebut yaitu index.php, my_template.php, upload.php, detail.php. Sedangkan database untuk menyimpan data secara permanen, yang nantinya akan dikelola dan diakses melalui Model.

Secara keseluruhan, alur ini menggambarkan bagaimana pengguna berinteraksi melalui browser, lalu permintaan diproses Controller, diteruskan ke Model jika berhubungan dengan database, dan datanya dikembalikan dan ditampilkan ke view. MVC ini akan memisahkan dengan jelas antara logika, data dan tampilan, sehingga jadi lebih terstruktur. Pada bagian template ini menggunakan format URL sebagai berikut:

- Controller Parent

http://localhost/project3/index.php

URL ini menunjukkan bahwa browser akan mengakses file index.php. File ini menjadi awalan untuk masuk ke seluruh aplikasi. Karena di dalam file ini, terdapat kode untuk routing, yaitu menentukan controller dan method mana yang harus dijalankan berdasarkan parameter URL. Index ini akan mengarahkan ke controller parent yang isinya sebagai berikut.

```
<?php

class Controller {
   function loadView($view = '', $data = []) {
     foreach($data as $key => $val)
     $$key = $val;
     include 'view/' . $view;
   }

}
```

Kelas controller ini berfungsi sebagai controller parent yang menyediakan method umum untuk memuat view dan mngirimkan data pada view tersebut. Di dalamnya terdapat fungsi loadView($view = '', $data = []) yang gunanya untuk mengambil data yang dikirim dari controller dalam bentuk array, lalu mengubah setiap elemen array tersebut menjadi variable biasa dengan foreach, sehingga datanya dapat diakses secara langsung. Manfaat lainnya yaitu untuk menyisipkan file view berdasarkan nama file yang diberikan, dengan mencari file tersebut di folder view. Sehingga controller lebih sederhana dengan hanya memanggil loadView(). Berikut controller yang digunakan untuk tampilan dalam web:

1. Halaman Index.php

http://localhost/project2/index.php?c=Templates&m=list

URL ini untuk mengakses web dengan mengarahkan pengguna untuk membuak file index.php dengan parameter c=Templates dan m=list. Parameter c=Templates memberitahu aplikasi bahwa controller yang akan dijalankan adalah Templates, sedangkan m=list menunjukkan bahwa method list() dari controller tersebut yang harus dieksekusi. Ketika URL ini diakses, sistem akan memuat semua data template yang tersimpan di database melalui model, kemudian menampilkannya ke halaman view yang telah disiapkan.

```
// Controller
<?php

class Templates extends Controller {

   function list() {
      session_start();
      $username = $_SESSION['username'] ?? 'Pengunjung';
```

```
        $model = $this->loadModel('TemplateModel');
        $data = $model->getAllTemplates();
        $this->loadView('templates.php', [
            'templates' => $data,
            'username' => $username
        ]);
    }

    // Model
    <?php

    class TemplateModel extends Model {
     function getAllTemplates() {
        $sql = "SELECT * FROM growhub";
        $result = $this->db->query($sql);
        return $result->fetch_all(MYSQLI_ASSOC);
     }
    }
```

Dengan controller tersebut system akan secara otomatis menampilkan semua daftar template yang ada di database dengan rapi sesuai dengan perintah. Pertama tama fungsi list memanggil TemplateModel melalui $this->loadModel('TemplateModel') yang terkoneksi dengan database. Setelah model, fungsi getAllTemplates() dipanggil untuk mengambil semua data template dari tabel growhub. Data yang berhasil diambil disimpan dalam variable $data. Selain itu, nama pengguna yang sedang login diambil dari $_SESSION['username'], dan disimpan dalam variabel $username. Kedua data tersebut dikirimkan ke view menggunakan $this->loadView('templates.php', [...]).

Fungsi getAllTemplates() sendiri berfungsi untuk menjalankan query SELECT * FROM growhub untuk mengambil seluruh data template, lalu mengembalikannya dalam bentuk array asosiatif menggunakan fetch_all(MYSQLI_ASSOC).

2. Halaman Upload.php

```
http://localhost/project2/index.php?c=Templates&m=submit
```

URL ini digunakan untuk membuka halaman dalam aplikasi web lokal yang mengarahkan pengguna ke proses pengisian atau pengunggahan template baru. Pada URL tersebut, file index.php di folder project bertugas menangkap parameter c=Templates untuk memanggil controller Templates, lalu menjalankan method submit().

```
// Controller
<?php

class Templates extends Controller {
      function submit() {
    $username = $_SESSION['username'] ?? 'Pengunjung';
    $this->loadView('upload.php', ['username' => $username]);
```

```
        }

    public function add() {
        session_start();
        if (!isset($_SESSION['user_id'])) {
            header("Location: login.php");
            exit();
        }
        $userId = $_SESSION['user_id'];
        $title = $_POST['title'];
        $category = $_POST['category'];

        $uploadDir = __DIR__ . '/../uploads/';
        if (!is_dir($uploadDir)) mkdir($uploadDir, 0755, true);

        $safeName = preg_replace('/[^A-Za-z0-9_\-\.]/', '_', $_FILES['template_file']['name']);
        $tmpPath  = $_FILES['template_file']['tmp_name'];

        if (!move_uploaded_file($tmpPath, $uploadDir . $safeName)) {
            die('Upload gagal');
        }

        $this->loadModel('TemplateModel')->insert($title, $category, 'uploads/' . $safeName, $userId);
        $this->mytemplate();
    }
}

// Model
<?php

class TemplateModel extends Model {

  function insert($title, $category, $filePath, $userId) {
      $stmt = $this->db->prepare("INSERT INTO growhub (title, category, file_path, user_id) VALUES (?, ?, ?,
?)");
      $stmt->bind_param("sssi", $title, $category, $filePath, $userId);
      return $stmt->execute();
  }
}
```

Method submit akan menampilkan form yang ada di halaman upload.php. Setelah pengguna mengisi form dan mengirimkannya, form tersebut akan dialihkan ke method add. Method add berfungsi untuk memproses upload template ke server dan menyimpan informasi template tersebut ke database. Pertama, method ini memastikan bahwa pengguna sudah login dengan memeriksa $_SESSION['user_id']. Jika belum login, pengguna akan diarahkan ke halaman login. Setelah itu, data title dan category diambil dari $_POST.

Kemudian method ini menentukan lokasi folder upload yaitu di dalam folder uploads/. Jika folder belum ada, maka akan dibuat otomatis menggunakan mkdir(). Nama file yang diunggah disanitasi menggunakan preg_replace() agar hanya mengandung karakter huruf, angka, underscore, strip, dan titik. File kemudian dipindahkan dari lokasi sementara (tmp_name) ke folder tujuan menggunakan move_uploaded_file(). Jika proses ini gagal, maka akan ditampilkan pesan "Upload gagal" dan proses dihentikan.

Jika berhasil, method akan memanggil fungsi insert() dari TemplateModel untuk menyimpan data title, category, file_path, dan user_id ke dalam database. Fungsi insert() di dalam model menggunakan prepared statement (bind_param) untuk mencegah SQL injection dan memastikan data tersimpan dengan aman di tabel growhub. Setelah data berhasil disimpan, method add akan memanggil mytemplate() untuk menampilkan daftar template milik pengguna tersebut.

3. Halaman My_template.php

http://localhost/project2/index.php?c=Templates&m=mytemplate

URL ini digunakan untuk mengakses halaman di aplikasi lokal yang menampilkan daftar template milik pengguna. Pada URL ini, index.php di folder project menerima dua parameter: c=Templates untuk memanggil controller Templates, dan m=mytemplate untuk menjalankan method mytemplate(). Saat method ini dipanggil, controller akan mengambil data semua template dari database melalui model, lalu mengirimkannya ke view my_template.php untuk ditampilkan.

```
// Controller
<?php

class Templates extends Controller {

  function mytemplate() {
    session_start();
    $username = $_SESSION['username'] ?? 'Pengunjung';
    $userId = $_SESSION['user_id'];

    $model = $this->loadModel('TemplateModel');
    $data = $model->getTemplatesByUser($userId);
    $this->loadView('my_template.php', [
      'templates' => $data,
      'username' => $username
    ]);
  }
}

// Model
<?php

class TemplateModel extends Model {
```

```
 function getTemplatesByUser($userId) {
    $sql = "SELECT * FROM growhub WHERE user_id = $userId";
    $result = $this->db->query($sql);
    return $result->fetch_all(MYSQLI_ASSOC);
 }
}
```

Method ini berfungsi untuk menampilkan semua data template yang dimiliki oleh pengguna yang sedang login ke halaman my_template.php. Pertama-tama, method mytemplate() mengambil user_id dan username dari $_SESSION. Lalu, method ini memuat TemplateModel melalui $this->loadModel('TemplateModel') dan memanggil fungsi getTemplatesByUser($userId) untuk mengambil semua data template yang hanya dimiliki oleh pengguna tersebut.

Data template hasil query disimpan dalam variabel $data, kemudian dikirimkan ke view my_template.php bersama dengan nama pengguna (username) melalui $this->loadView(...), sehingga data dapat ditampilkan kepada user yang sedang login.
Pada bagian model, fungsi getTemplatesByUser($userId) menjalankan query SQL untuk mengambil seluruh data dari tabel growhub berdasarkan user_id. Data dikembalikan dalam bentuk array asosiatif (fetch_all(MYSQLI_ASSOC)) dan digunakan oleh controller.

4. Halaman detail_template.php

```
http://localhost/project2/index.php?c=Templates&m=detail&id=52
```

URL ini untuk mengakses sebuah halaman detail dari satu template tertentu di aplikasi lokal kita. Parameter c=Templates memberitahu sistem bahwa controller yang dipanggil adalah Templates, lalu m=detail menunjukkan bahwa method detail() di dalam controller itu yang akan dijalankan. Sementara id=52 adalah data tambahan yang menunjukkan bahwa yang ingin kita lihat detailnya adalah template dengan id 52 di database.

```
// Controller
<?php

class Templates extends Controller {

function detail() {
   if (!isset($_GET['id'])) {
      die("ID tidak ditemukan.");
   }

   $id = intval($_GET['id']);
   $model = $this->loadModel('TemplateModel');

   $template = $model->getTemplateById($id);

   if (!$template) {
      die("Template tidak ditemukan.");
```

```
    }

    session_start();
    $username = $_SESSION['username'] ?? 'Pengunjung';

    $this->loadView('detail_template.php', [
        'template' => $template,
        'username' => $username
    ]);
}

    function delete() {
        $id = $_GET['id'];
        $model = $this->loadModel('TemplateModel');
        $model->delete($id);
        header('location:index.php?c=Templates&m=list');
    }
}

//Model
<?php

class TemplateModel extends Model {
  function getTemplateById($id) {
      $sql = "SELECT * FROM growhub WHERE id = $id";
      $result = $this->db->query($sql);
      return $result->fetch_assoc();
  }

  function delete($id) {
    $sql = "DELETE FROM growhub WHERE id = '$id'";
    return $this->db->query($sql);
  }
}
```

Method detail digunakan untuk menampilkan detail dari satu template berdasarkan id yang dipilih oleh pengguna. ID tersebut diambil dari URL menggunakan $_GET['id'] dan diamankan dengan fungsi intval() agar hanya berupa angka. Setelah itu, method memuat TemplateModel dan memanggil fungsi getTemplateById($id) untuk mengambil data template sesuai ID. Hasil query disimpan dalam variabel $template.

Jika data ditemukan, informasi tersebut akan dikirim ke view detail_template.php dengan nama pengguna (username) yang diambil dari session. Jika tidak ditemukan, proses dihentikan dan pesan error ditampilkan.

Method delete digunakan untuk menghapus data template berdasarkan id yang juga dikirim melalui URL ($_GET['id']). Setelah model dimuat, fungsi delete($id) dipanggil untuk menghapus

data dari database berdasarkan ID. Setelah penghapusan berhasil, pengguna akan langsung diarahkan kembali ke halaman daftar template (list) menggunakan fungsi header('location:...'). Pada bagian model, fungsi getTemplateById($id) menjalankan query untuk mengambil satu baris data template berdasarkan id, sedangkan fungsi delete($id) menjalankan query DELETE untuk menghapus data dari tabel growhub.

Berikut DDL untuk tabel-tabel yang terlibat di dalam database.

```
CREATE TABLE user (
  id INT AUTO INCREMENT PRIMARY KEY,
  username VARCHAR(100) NOT NULL,
  password VARCHAR(100) NOT NULL
);

CREATE TABLE growhub (
  id INT AUTO INCREMENT PRIMARY KEY,
  title VARCHAR(255) NOT NULL,
  category INT,
  file_path VARCHAR(255),
  user_id INT,
  FOREIGN KEY (user_id) REFERENCE user(id)
);
```

Implementasi Halaman Index.php

File View antarmuka HTML/PHP diletakkan di dalam folder view/ lalu di bagian Controller diberikan kode program:

```
//file: Templates.class.php

<?php

class Templates extends Controller {

   function list($view = 'index.php') {
      $model = $this->loadModel('TemplateModel');
      $data = $model->getAllTemplates();
      $this->loadView($view, ['templates' => $data]);
   }

}
```

Pada bagian Model diberikan kode program:

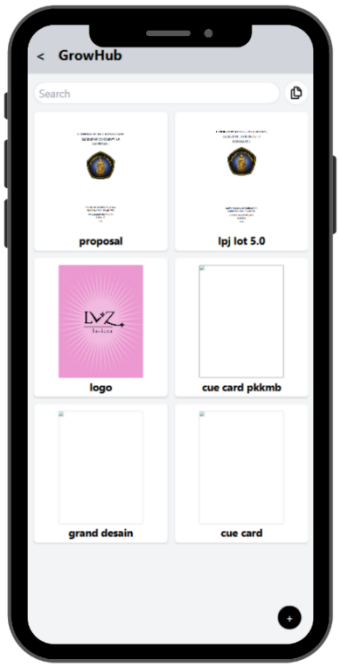
```
      <?php
```

```
class TemplateModel extends Model {
 function getAllTemplates() {
    $sql = "SELECT * FROM growhub";
    $result = $this->db->query($sql);
    return $result->fetch_all(MYSQLI_ASSOC);
 }
}
```

Kemudian melalui browser diakses dengan URL:

http://localhost/project2/index.php?c=Templates&m=list

Sehingga pada jendela browser ditampilkan halaman web yang dimaksud seperti pada Gambar berikut:



Implementasi Halaman Upload.php

File View antarmuka HTML/PHP diletakkan di dalam folder view/ lalu di bagian Controller diberikan kode program:

//file: Templates.class.php

```
<?php

class Templates extends Controller {

   function submit() {
```

```
        $this->loadView('upload.php');
    }

    public function add() {
        $title    = $_POST['title'];
        $category = $_POST['category'];

        $uploadDir = __DIR__ . '/../uploads/';
        if (!is_dir($uploadDir)) mkdir($uploadDir, 0755, true);

        $safeName = preg_replace('/[^A-Za-z0-9_\-\.]/', '_', $_FILES['template_file']['name']);
        $tmpPath  = $_FILES['template_file']['tmp_name'];

        if (!move_uploaded_file($tmpPath, $uploadDir . $safeName)) {
            die('Upload gagal');
        }

        $this->loadModel('TemplateModel')
             ->insert($title, $category, 'uploads/' . $safeName);

        $this->list('my_template.php');
    }

}
```

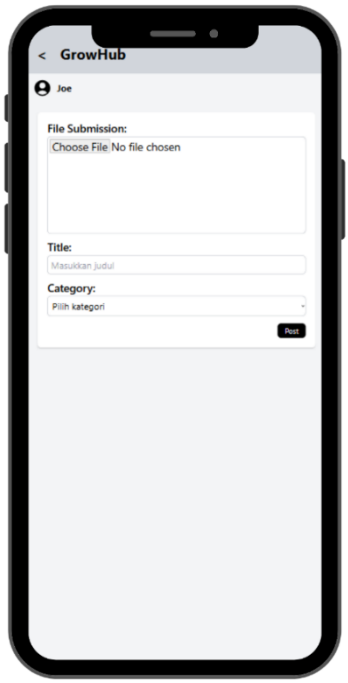
Pada bagian Model diberikan kode program:

```
<?php

class TemplateModel extends Model {

  function insert($title, $category, $filePath, $userId) {
        $stmt = $this->db->prepare("INSERT INTO growhub (title, category, file_path, user_id) VALUES (?, ?, ?,
?)");
        $stmt->bind_param("sssi", $title, $category, $filePath, $userId);
        return $stmt->execute();
  }
}
```

Kemudian melalui browser diakses dengan URL:

http://localhost/project2/index.php?c=Templates&m=submit

Sehingga pada jendela browser ditampilkan halaman web yang dimaksud seperti pada Gambar berikut:



Implementasi Halaman My_template.php

File View antarmuka HTML/PHP diletakkan di dalam folder view/ lalu di bagian Controller diberikan kode program:

```
//file: Templates.class.php

<?php

class Templates extends Controller {

   function mytemplate() {
      $model = $this->loadModel('TemplateModel');
      $data = $model->getAllTemplates();
      $this->loadView('my_template.php', ['templates' => $data]);
   }

}
```

Pada bagian Model diberikan kode program:

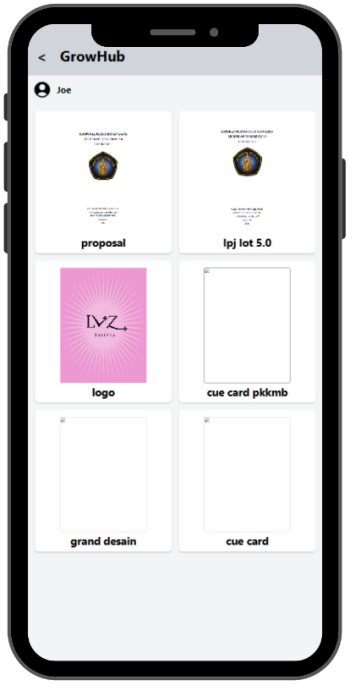
```
      class TemplateModel extends Model {

        function getTemplatesByUser($userId) {
           $sql = "SELECT * FROM growhub WHERE user_id = $userId";
           $result = $this->db->query($sql);
           return $result->fetch_all(MYSQLI_ASSOC);
```

```
    }
}
```

Kemudian melalui browser diakses dengan URL:

http://localhost/project2/index.php?c=Templates&m=mytemplate

Sehingga pada jendela browser ditampilkan halaman web yang dimaksud seperti pada Gambar berikut:



Implementasi Halaman Detail_template.php

File View antarmuka HTML/PHP diletakkan di dalam folder view/ lalu di bagian Controller diberikan kode program:

```
//file: Templates.class.php

<?php

class Templates extends Controller {

  function detail() {
    $id = $_GET['id'];
    $model = $this->loadModel('TemplateModel');
    $template = $model->getTemplate($id);
    $this->loadView('detail_template.php', ['template' => $template]);
  }
```

```
  function delete() {
    $id = $_GET['id'];
    $model = $this->loadModel('TemplateModel');
    $model->delete($id);
    header('location:index.php?c=Templates&m=list');
  }

}
```

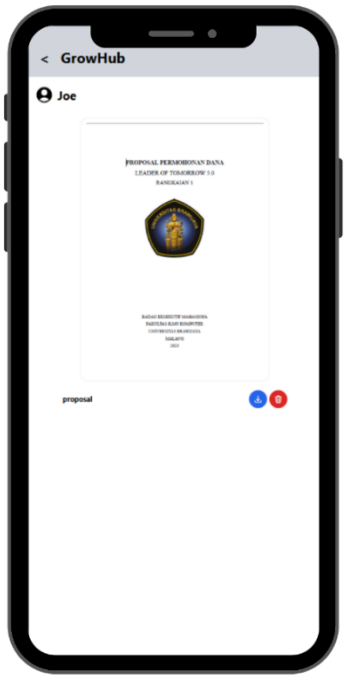
Pada bagian Model diberikan kode program:

```
    <?php

    class TemplateModel extends Model {
     function getTemplateById($id) {
        $sql = "SELECT * FROM growhub WHERE id = $id";
        $result = $this->db->query($sql);
        return $result->fetch_assoc();
     }

     function delete($id) {
       $sql = "DELETE FROM growhub WHERE id = '$id'";
       return $this->db->query($sql);
     }
    }
```

Kemudian melalui browser diakses dengan URL:

http://localhost/project2/index.php?c=Templates&m=detail&id=52

Sehingga pada jendela browser ditampilkan halaman web yang dimaksud seperti pada Gambar berikut:


GrowHub
Joe
PROPOSAL PERMOHONAN DANA
LEADER OF TOMORROW 5.0
BANGKALAN 1
proposal

Use Case: Mengelola Thread

Penyusun Use Case: Calista Aulia Rianka Rachma – 2351504071111032

Use case Mengelola Thread menjelaskan proses interaksi pengguna dengan fitur manajemen thread di platform GrowLink. Pengguna dapat melakukan berbagai aktivitas seperti melihat daftar thread, membuat thread baru, dan menghapus thread yang sudah ada yang dilakukan pada halaman-halaman tertentu. Untuk lebih detail, dapat dilihat pada use case scenario berikut:

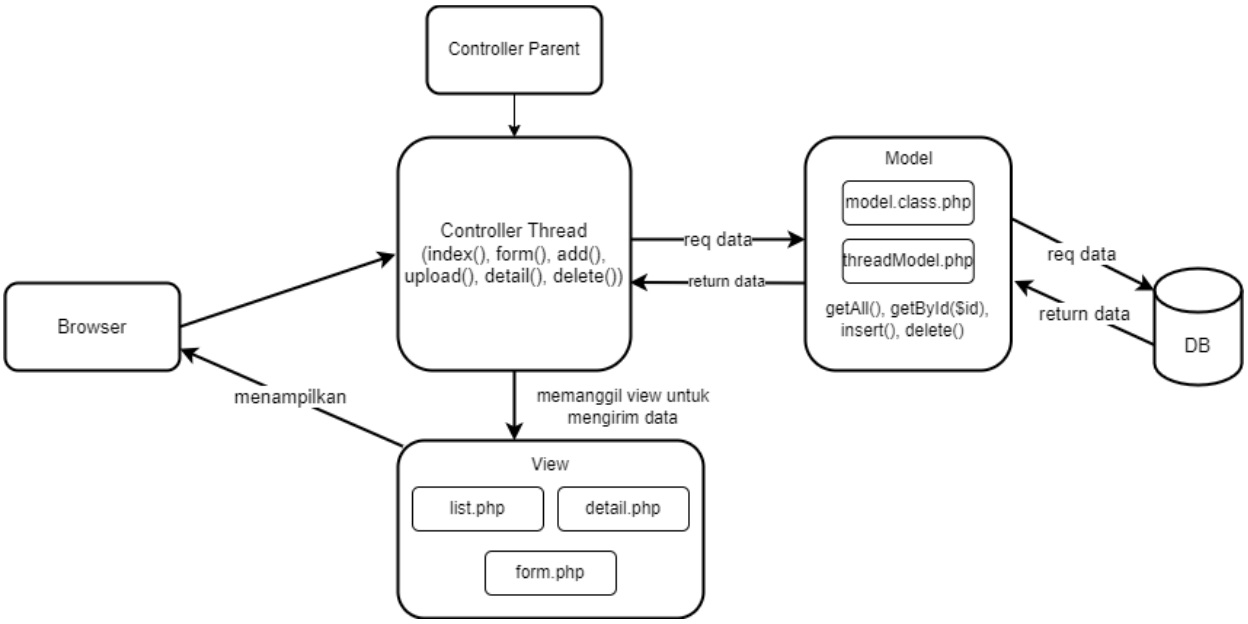
Actor:	Pengguna.
Description:	Pengguna melakukan berbagai aktivitas terkait manajemen thread, seperti membuat, menghapus, dan mengunggah thread.
Use Case Goal:	Pengguna dapat mengelola fitur thread dalam *platform* GrowLink.
Preconditions:	Pengguna telah masuk ke dalam platform GrowLink Halaman home GrowLink sudah ditampilkan
Main Flow:	1. Pengguna memilih halaman thread dengan menekan section thread pada homepage. 2. Sistem menampilkan beranda thread yang berisi daftar thread. 3. Pengguna dapat melihat daftar thread lain yang telah dibuat oleh pengguna lain. 4. Pengguna dapat memilih thread yang ingin dilihat. 5. Sistem menampilkan thread sesuai dengan pilihan pengguna. 6. Pengguna dapat membuat thread baru. 7. Sistem menampilkan form pembuatan thread. 8. Pengguna mengisi dan mengonfirmasi pengunggahan konten thread. 9. Sistem menyimpan thread yang berhasil diunggah. 10. Sistem menampilkan thread pada daftar thread.
Alternative Flow - 1:	8.a Pengguna memilih untuk membatalkan pembuatan thread. 8.a.1 Sistem tidak menyimpan thread dan kembali ke halaman sebelumnya.
Alternative Flow - 2:	9.a Sistem menampilkan informasi bahwa thread tidak dapat diunggah. 9.a.1 Sistem menampilkan pesan kesalahan dan meminta pengguna mencoba ulang pengunggah thread. 9.b Pengguna mencoba lagi pengisian ulang form unggah thread.

Alternative Flow - 3:	10.a Pengguna memilih untuk menghapus thread dengan menekan tombol hapus. 10.a.1 Sistem menghapus thread dari daftar thread.
Postconditions:	Thread dapat divalidasi sehingga dapat dilihat pada daftar thread. Pengguna dapat menghapus thread.

Implementasi Controller dan Model

Pada diagram arsitektur MVC untuk mengelola Thread di platform GrowLink, alur kerja dimulai dari browser mengirimkan permintaan HTTP ke server dengan URL http://localhost/pemwebproject3/index.php?c=Thread&m=index. Untuk index.php tersebut berperan sebagai Front Controller yang menerima semua permintaan dan menganalisis parameter URL untuk menentukan controller dan method yang harus dipanggil. Thread Controller merupakan controller berisi method-method yang menangani seluruh fungsionalitas terkait thread, seperti menampilkan daftar thread, membuat thread baru, dan menghapus thread yang sudah ada. Dalam implementasi ini, database disimulasikan dengan array $threads yang berada di dalam Thread Controller. Array ini berfungsi sebagai model dalam arsitektur MVC, menyimpan, dan mengelola data thread. Pada implementasinya, array ini digantikan dengan koneksi ke database dan model khusus untuk mengakses data. Thread Controller menggunakan method-method seperti index(), form(), add(), upload(), detail(), dan delete() untuk memproses permintaan pengguna dan berinteraksi dengan data thread. Setelah data diproses, controller memanggil fungsi loadView() yang diwarisi dari controller.class.php untuk menampilkan file view yang sesuai, yakni list.php, form.php, detail.php.

Controller.class.php bertindak sebagai class dasar yang menyediakan fungsi umum loadView() untuk semua controller. Fungsi ini menjadikan controller dapat memuat file tampilan dan meneruskan data yang diperlukan ke tampilan tersebut. Tampilan kemudian akan memformat data dan menampilkannya kepada pengguna dalam bentuk halaman website. Arsitektur MVC ini memisahkan dengan jelas antara pengontrolan alur aplikasi secara logika (controller), data (model), dan tampilan (view), sehingga memudahkan pengembangan aplikasi web. Berikut adalah diagram blok alur proses dan elemen-elemen MVC:



Dalam use case Mengelola thread, method-method yang digunakan adalah method dalam controller thread.class.php. Untuk uraian method yang terlibat, dimulai **dengan method index()** yang berguna untuk menampilkan semua thread yang tersimpan dalam database. Lalu **method form()** menampilkan halaman form untuk pengguna yang ingin membuat thread baru. Setelah mengisi content thread, selanjutnya method add() menyimpan thread baru ke database. Untuk **method detail()**, digunakan untuk menampilkan detail thread berdasarkan ID ketika pengguna ingin melihat thread tertentu. Untuk thread milik pengguna itu sendiri (bukan pengguna lain), **terdapat method delete()** yang berfungsi untuk menghapus thread dari database berdasarkan ID.

Model yang digunakan dalam seluruh proses ini adalah threadModel, yang merupakan turunan dari model.class.php. Model ini bertanggung jawab untuk menangani interaksi dengan database, termasuk mengambil semua thread, menyimpan thread baru, mengambil detail berdasarkan ID, dan menghapus thread. Berikut adalah implementasi method dalam threadModel yang saling berhubungan antara controller dengan model:

a) Method index() memanggil getAll()

Controller akan memanggil model untuk mengambil semua data thread dari tabel `growforum`, lalu meneruskannya ke view list.php

```
// Controller
$threads = $this->model->getAll();
$this->loadView('list.php', ['threads' => $threads]);
```

Model menerima data tersebut dan memprosesnya sebagai berikut:

```
// Model
function getAll() {
   $sql = "SELECT * FROM growforum ORDER BY id DESC";
   $result = $this->db->query($sql);
   return $result->fetch_all(MYSQLI_ASSOC);
```

```
}
```

Model akan mengembalikan array yang berisi seluruh thread dari database ke controller.

b) Method add() memanggil insert()

Controller akan memanggil model untuk menyimpan thread baru ke dalam database. Input berasal dari form.

```
// Controller
$content = $_POST['content'] ?? '';
$author = 'Jey';
if (!empty($content)) {
   $this->model->insert($author, $content);
}
header("Location: index.php?c=Thread&m=index");
```

Model menerima data tersebut dan memprosesnya sebagai berikut:

```
// Model
function insert($author, $content) {
   $stmt = $this->db->prepare("INSERT INTO growforum (author, content) VALUES (?, ?)");
   $stmt->bind_param("ss", $author, $content);
   return $stmt->execute();
}
```

Model akan mengembalikan nilai true jika proses insert berhasil, atau false jika gagal.

c) Method detail() memanggil getById()

Controller akan memanggil model untuk mengambil data thread berdasarkan ID tertentu.

```
// Controller
$id = $_GET['id'] ?? null;
if ($id !== null) {
   $thread = $this->model->getById($id);
   if ($thread) {
      $this->loadView('detail.php', ['thread' => $thread]);
      return;
   }
}
echo "Thread tidak ditemukan.";
```

Model menerima data tersebut dan memprosesnya sebagai berikut:

```
// Model
function getById($id) {
   $sql = "SELECT * FROM growforum WHERE id = '$id'";
   $result = $this->db->query($sql);
   return $result->fetch_assoc();
```

```
}
```

Model akan mengembalikan satu baris data thread dalam bentuk associative array.

d) Method delete() memanggil delete()

Controller akan memanggil model untuk menghapus thread berdasarkan ID dan author.

```
// Controller
$id = $_GET['id'] ?? null;
if ($id !== null) {
   $this->model->delete($id, 'Jey');
}
header("Location: index.php?c=Thread&m=index");
```

Model menerima data tersebut dan memprosesnya sebagai berikut:

```
// Model
function delete($id, $author = 'Jey') {
   $stmt = $this->db->prepare("DELETE FROM growforum WHERE id = ? AND author = ?");
   $stmt->bind_param("is", $id, $author);
   return $stmt->execute();
}
```

Model akan mengembalikan nilai true jika proses hapus berhasil, atau false jika gagal.
Model ini mengambil data dari database, berikut adalah DDL atau struktur utama database yang digunakan dalam model-model tersebut.

```
CREATE DATABASE growlink;

USE growlink;

CREATE TABLE growforum (
   id INT AUTO_INCREMENT PRIMARY KEY,
   author VARCHAR(255) NOT NULL,
   content TEXT NOT NULL
);

//simulasi untuk pengguna lain
INSERT INTO growforum (author, content)
VALUES
 ('Joy', 'Halo, ini thread dari Joy '),
 ('Jiv', 'Hai semua, aku Jiv di sini ');
```

Implementasi Halaman Satu: form.php

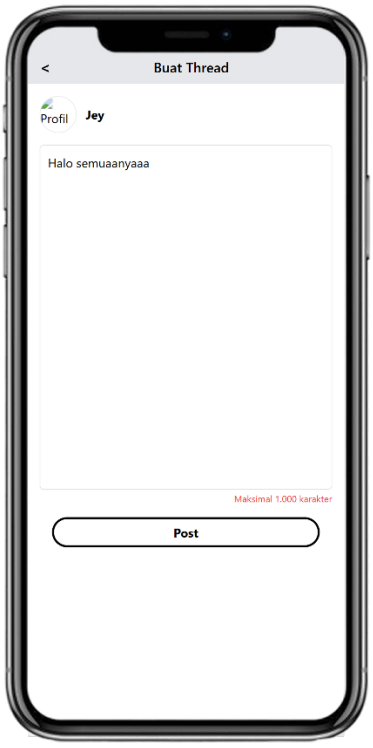
Halaman pertama yang diimplementasikan adalah form.php. Halaman ini menampilkan tampilan form berupa pengisian konten thread untuk mengunggah thread. Halaman ini juga memiliki elemen trigger berupa tombol post yang akan mengirimkan data ke controller untuk diproses dan disimpan ke dalam database. Berikut merupakan kode HTML yang memiliki elemen input dan trigger:

```
 <form method="POST" action="index.php?c=Thread&m=add" class="flex flex-col flex-grow">
<div class="p-4">
   <button type="submit" class="btn-post w-full">Post</button>
</div>
```

Pengguna dapat mengakses halaman ini melalui browser menggunakan URL berikut:

http://localhost/pemwebproject4/index.php?c=Thread&m=form

Sehingga pada jendela browser ditampilkan halaman web yang dimaksud seperti dalam Gambar berikut:



Implementasi Halaman Dua: list.php

Halaman kedua yang diimplementasikan adalah list.php. Halaman ini menampilkan data thread yang telah dikirim oleh pengguna. Data tersebut diperoleh dari database melalui model,

diteruskan oleh controller, dan kemudian ditampilkan oleh view dengan tampilan dari halaman ini berupa daftar thread-thread milik pengguna itu sendiri maupun pengguna lain. Tujuan utama dari halaman ini adalah memberikan umpan balik visual kepada pengguna mengenai thread dari form yang telah mereka isikan.

Untuk menampilkan data konten thread pada halaman list.php, maka terlebih dahulu model mengambil data dari database melalui method add() pada threadModel.class.php yang akan mengambil seluruh data konten thread. Berikut contoh data luarannya.

```
// nilai data luaran dari model fungsi add()
array(3) {
  [0] => array(
    “id” => 1,
    “author” => “Joy”,
    “content” => “Halo, ini thread dari Joy”,
    )
  [1] => array(
    “id” => 2,
    “author” => “Jiv”,
    “content” => “Hai semua, aku Jiv di sini”,
    )
  [2] => array(
    “id” => 3,
    “author” => “Jey”,
    “content” => “Halo semuaanyaaa”,
    )
}
```

Data luaran tersebut kemudian akan diterima oleh controller dan mengirimkannya ke view melalui method index() pada controller Thread.

```
// kode program controller yang menerima data dari Model
$threads = $this->model->getAll();

// dan melempar data yang diterima ke View.
$this->loadView('list.php', ['threads' => $threads]);
```

View list.php akan menerima data array $thread dari controller dan menampilkannya dalam format HTML dengan kode program berikut:

```
// kode program di dalam View yang menampilkan data
<?php foreach ($threads as $thread): ?>
<a href="index.php?c=Thread&m=detail&id=<?php echo $thread['id']; ?>" class="block">
<div class="thread-card bg-white rounded-xl p-4 mx-2 hover:bg-gray-100 transition">
<div class="flex items-start space-x-3">
<img src="profile.png" alt="Profil" class="w-8 h-8 rounded-full border border-black">
        <div>
                <strong><?php echo $thread['author']; ?></strong>
```

```
                    <p class="mt-1 text-sm text-gray-800"><?php echo $thread['content']; ?></p>
                    </div>
            </div>
 </div>
</a>
<?php endforeach; ?>
```

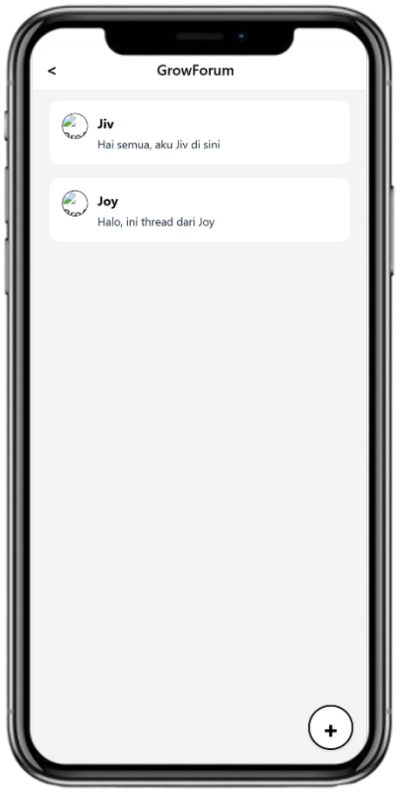
Pengguna dapat mengakses halaman ini melalui browser menggunakan URL berikut:

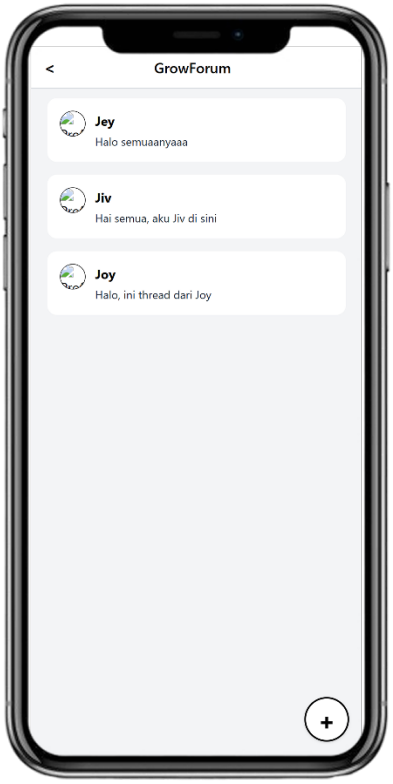
http://localhost/pemwebproject4/index.php?c=Thread&m=index

Halaman ini juga memiliki elemen trigger berupa tombol tambah yang akan mengirimkan data ke controller untuk diproses dan disimpan ke dalam database. Berikut merupakan kode HTML yang memiliki elemen input dan trigger:

```
 <button id="btnAddThread" class="fixed bottom-6 right-4 bg-white border-2 border-black rounded-full w-14
h-14 flex items-center justify-center text-3xl font-extrabold text-black shadow">
        +
    </button>
  </div>

  <script>
    document.getElementById("btnAddThread").onclick = function () {
      window.location.href = "index.php?c=Thread&m=form";
    };
  </script>
```

Sehingga pada jendela browser ditampilkan halaman web yang dimaksud seperti dalam Gambar berikut:





Implementasi Halaman Tiga: detail.php

Halaman ketiga yang diimplementasikan adalah detail.php. Halaman ini ditampilkan saat pengguna memilih thread tertentu pada halaman list. Halaman ini menerima parameter id author dari URL, kemudian controller akan mengambil data thread tersebut dari database menggunakan model, dan menampilkannya dalam halaman detail.

Untuk menampilkan detail thread pada detail.php, maka terlebih dahulu model mengambil data dari database, melalui method getById($id) pada threadModel.class.php yang akan mengambil data thread berdasarkan ID. Berikut contoh data luarannya.

```
// nilai data luaran dari model pada fungsi getClaimById($id)
array(3) {
 ["id"] => int(2)
 ["Author"] => string(3) "Jey"
 ["Content"] => string(15) " Halo semuaanyaaa"
}
```

Data luaran tersebut kemudian akan diterima oleh controller dan mengirimkannya ke view melalui method detail() pada controller Thread.

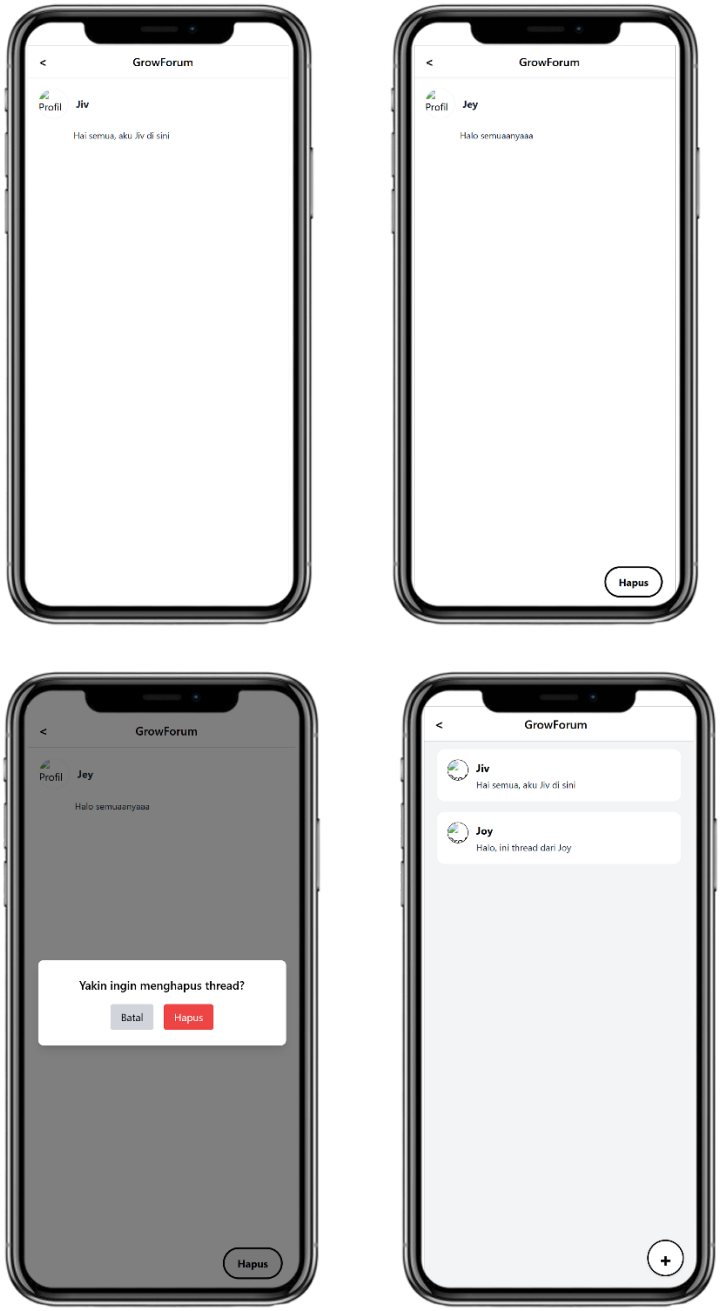
```
// kode program controller yang menerima data dari Model
```

```
$thread = $this->model->getById($id);

// dan melempar data yang diterima ke View.
$this->loadView('detail.php', ['thread' => $thread]);
```

View detail.php akan menerima data array $thread dari controller dan menampilkannya dalam format HTML dengan kode program berikut:

```
// kode program di dalam View yang menampilkan data

<main class="flex-grow p-4">
          <div class="flex items-center space-x-3 mb-4">
           <img src="profile.png" alt="Profil" class="w-12 h-12 rounded-full border border-gray-100">
           <strong class="text-gray-800"><?php echo $thread['author']; ?></strong>
        </div>
        <div class="pl-14 text-gray-800">
           <p class="text-sm"><?php echo nl2br($thread['content']); ?></p>
        </div>
     </main>
```

Sehingga pada jendela browser ditampilkan halaman web yang dimaksud seperti pada Gambar berikut:



Dari kode di atas, selain untuk menampilkan detail thread seperti pada gambar, juga memiliki fungsi untuk menghapus thread berdasarkan ID. Jadi halaman ini juga memiliki elemen trigger berupa tombol untuk menghapus thread miliknya sendiri. Berikut merupakan kode program elemen trigger tersebut:

```
<?php if ($thread['author'] == 'Jey'): ?>
        <button class="btn-delete" onclick="showConfirmModal()">Hapus</button>
<?php endif; ?>
```

Ketika pengguna menekan tombol "Hapus" pada thread miliknya sendiri di halaman detail.php, sistem akan memunculkan pop-up konfirmasi persetujuan pengguna untuk melanjutkan proses penghapusan thread dengan Java Script. Jika pengguna memilih "Hapus" di dalam pop-up tersebut, maka browser akan diarahkan ke URL:

https://localhost/projectpemweb4/index.php?c=Claim&m=delete&id=(nomor id)

URL tersebut memanggil controller Thread dengan method delete(), serta membawa parameter id dari thread yang ingin dihapus. Controller kemudian akan memanggil method delete dari model, untuk menghapus thread berdasarkan ID. Berikut kode program pada controller yang memanggil model tersebut:

```
...
 if ($id !== null) {
        $this->model->delete($id, 'Jey');
      }
      header("Location: index.php?c=Thread&m=index");
      exit;
...
```

dengan kode HTML pada view sebagai berikut:

```
<script>
      function showConfirmModal() {
        document.getElementById('confirmModal').classList.remove('hidden');
      }

      function hideConfirmModal() {
        document.getElementById('confirmModal').classList.add('hidden');
      }

      function confirmDelete() {
        const threadId = <?php echo $thread['id']; ?>;
        window.location.href = 'index.php?c=Thread&m=delete&id=' + threadId;
      }
  </script>

<div id="confirmModal" class="fixed inset-0 bg-black bg-opacity-50 flex items-center justify-center hidden z-
50">
      <div class="bg-white p-6 rounded-lg shadow-lg max-w-sm w-full text-center">
        <h2 class="text-lg font-semibold mb-4">Yakin ingin menghapus thread?</h2>
        <div class="flex justify-center gap-4">
          <button onclick="hideConfirmModal()" class="px-4 py-2 bg-gray-300 rounded hover:bg-gray-
400">Batal</button>
          <button onclick="confirmDelete()" class="px-4 py-2 bg-red-500 text-white rounded hover:bg-red-
600">Hapus</button>
        </div>
      </div>
  </div>
```

Apabila penghapusan disetujui, sistem akan mengarahkan pengguna kembali ke halaman list.php dan jika penghapusan dibatalkan, maka kembali ke halaman detail.php.