BAB II
PELAKSANAAN UJIAN KOMPETENSI

2.1. Tujuan Pelaksanaan Ujian Kompetensi

Ujian kompetensi ini dilaksanakan untuk mengukur kemampuan teoritis dan praktis peserta dalam pengembangan aplikasi web menggunakan framework Laravel dan Bootstrap. Tujuannya adalah memberikan sertifikat resmi sebagai bukti pengakuan atas kompetensi peserta. Sertifikasi ini bertujuan untuk meningkatkan kredibilitas profesional peserta serta membantu mereka memahami dan menguasai teknologi terkini yang relevan dengan kebutuhan industri teknologi.

Tabel 2.1 Silabus Kompetensi.

No	Topik	Pembahasan	Output
1	Dasar – Dasar dan Persiapan	- Penganatar Laravel dan Boostrap. - Instalasi tools. - Slicing template UI menggunakan Boostrap.	Peserta siap dengan lingkungan pengembangan.
	Dasar Laravel dan CRUD	- Membuat CRUD sederhana. - Relasi antar entitas database - Uji coba dan debungging dasar.	Aplikasi CRUD dengan relasi database berjalan dengan baik.
2	Relasi dan Finalisasi	- CRUD relasi master-detail. - Implementasi fitur tambahan. - Uji coba aplikasi.	Aplikasi lengkap dengan fitur relasi dan autentikasi sederhana.

Tabel 2.2 Daftar Unit Kompetensi.

No	Kode Unit	Unit Kompetensi	Jam Pelaksanaan
1	TIK.SM03.001.01	Menentukan arsitektur perangkat keras	1
2	M.702090.005.01	Mengelola kualitas proyek (project quality management)	
3	M.702090.002.01	Mengelola Ruang Lingkup Proyek (Project Scope	2

No	Kode Unit	Unit Kompetensi	Jam Pelaksanaan
		Management)	
4	M.702090.001.01	Mengelola Proyek Secara Terintegrasi (Project Integration Management)	
5	J.62090.018.01	Mengelola Risiko Keamanan Informasi	
6	J.620100.041.01	Melaksanakan cutover aplikasi	2
7	J.620100.045.01	Melakukan pemantauan resource yang digunakan aplikasi	
8	J.620100.025.02	Melakukan debugging	3
9	J.620100.038.01	Melaksanakan pengujian oleh pengguna (UAT)	
10	J.620100.020.02	Menggunakan SQL	5
11	J.620100.044.01	Menerapkan alert notification jika aplikasi bermasalah	
12	J.620100.003.01	Melakukan identifikasi library, komponen atau framework yang diperlukan	
13	J.620100.024.02	Melakukan migrasi ke teknologi baru	
14	J.620100.047.01	Melakukan pembaharuan perangkat lunak	
15	J.620100.039.02	Memberikan petunjuk teknis kepada pelanggan	
16	J.620100.030.02	Menerapkan pemrograman multimedia	
17	J.620100.001.01	Menganalisis tools	5
18	J.620100.002.01	Menganalisis skalabilitas perangkat lunak	
19	J.620100.043.01	Menganalisis dampak perubahan terhadap aplikasi	
20	J.620100.029.02	Menerapkan pemrograman paralel	
21	J.620100.022.02	Mengimplement asikan algoritma pemrograman	
22	J.620100.028.02	Menerapkan pemrograman real time	

2.2. Proses Pendaftaran

Proses pendaftaran dilakukan secara online dengan tahapan sebagai berikut:

1. Peserta dapat mghubungi penyelenggara melalui akun Instragram resmi atau mendaftar langsung melalui situs web penyelenggara.

2. Perseta yang menghubingi melalui Instagram diberikan tautan form Google melalui WhatsApp untuk pengisian data diri, sedangkan yang mendaftar melalui situs web langsung mengisi formular yang tersedia.
3. Setelah mengisi formular, perserta menerima konfirmasi melalui WhatsApp dan dimasukkan ke dalam WhatsApp Group untuk komunikasi lebih lanjut.
4. Perserta diarahkan untuk mengikuti sesi pelatihan secara daring melalui aplikasi Zoom, dengan informasi jadwal yang dibagikan di WhatsApp.

2.3. Tahapan Persiapan Ujian Kompetensi.

2.3.1 Dasar – Dasar dan Persiapan

1. Pengenalan Laravel dan Boostrap.

Laravel adalah framework PHP yang berbasis arsitektur Model-View-Controller (MVC), yang dirancang untuk mempermudah pengembangan aplikasi web dengan fitur-fitur seperti routing, autentikasi, dan pengelolaan database. Sementara itu, Bootstrap adalah framework CSS yang memungkinkan pengembang untuk membuat antarmuka pengguna yang responsif dan modern secara efisien dengan memanfaatkan sistem grid dan komponen UI siap pakai. Integrasi Laravel dan Bootstrap memberikan solusi ideal dalam pengembangan aplikasi web, menggabungkan kekuatan Laravel sebagai backend yang terstruktur dengan fleksibilitas desain Bootstrap untuk menciptakan aplikasi yang fungsional, responsif, dan estetis.

2. Instalasi tools.

Pada tahap awal, peserta diminta untuk mengunduh XAMPP, Composer, dan Visual Studio Code, serta menyiapkan proyek berbasis Laravel dan Bootstrap. Setelah proses instalasi selesai, peserta diwajibkan untuk menyampaikan bukti instalasi berupa versi masing-masing alat yang telah berhasil diinstal.

a. Instalasi XAMPP.

Gambar 2.1 Hasil Instalasi XAMPP.

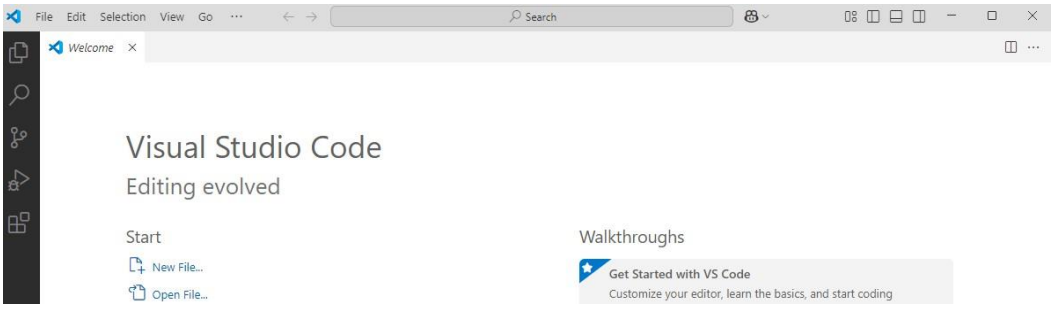
Pada gambar 2.1 merupakan tampilan dashboard hasil instalasi **XAMPP** beserta versinya.

b. Instalasi Composer.

Gambar 2.2 Hasil Instalasi Composer.

Pada gambar 2.2 merupakan hasil pengecekan Composer berserta versinya berhasil terinstal menggunakan perintah “composer -v” pada terminal Visual Studio Code.

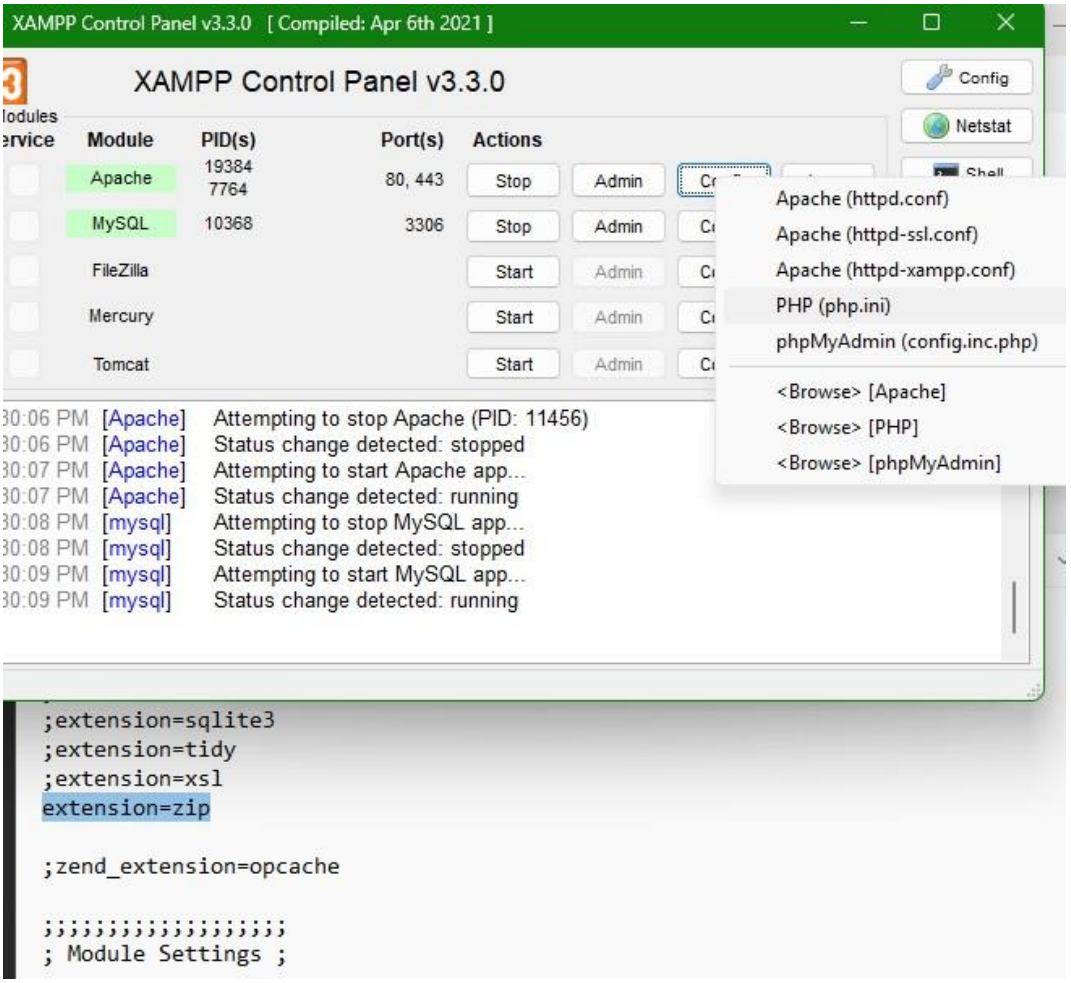
c. Instalasi Visual Studio Code.



Gambar 2.3 Hasil Instalasi VS Code.

Pada gambar 2.3 merupakan Visual Studio Code berhasil terinstal dan dijalankan.

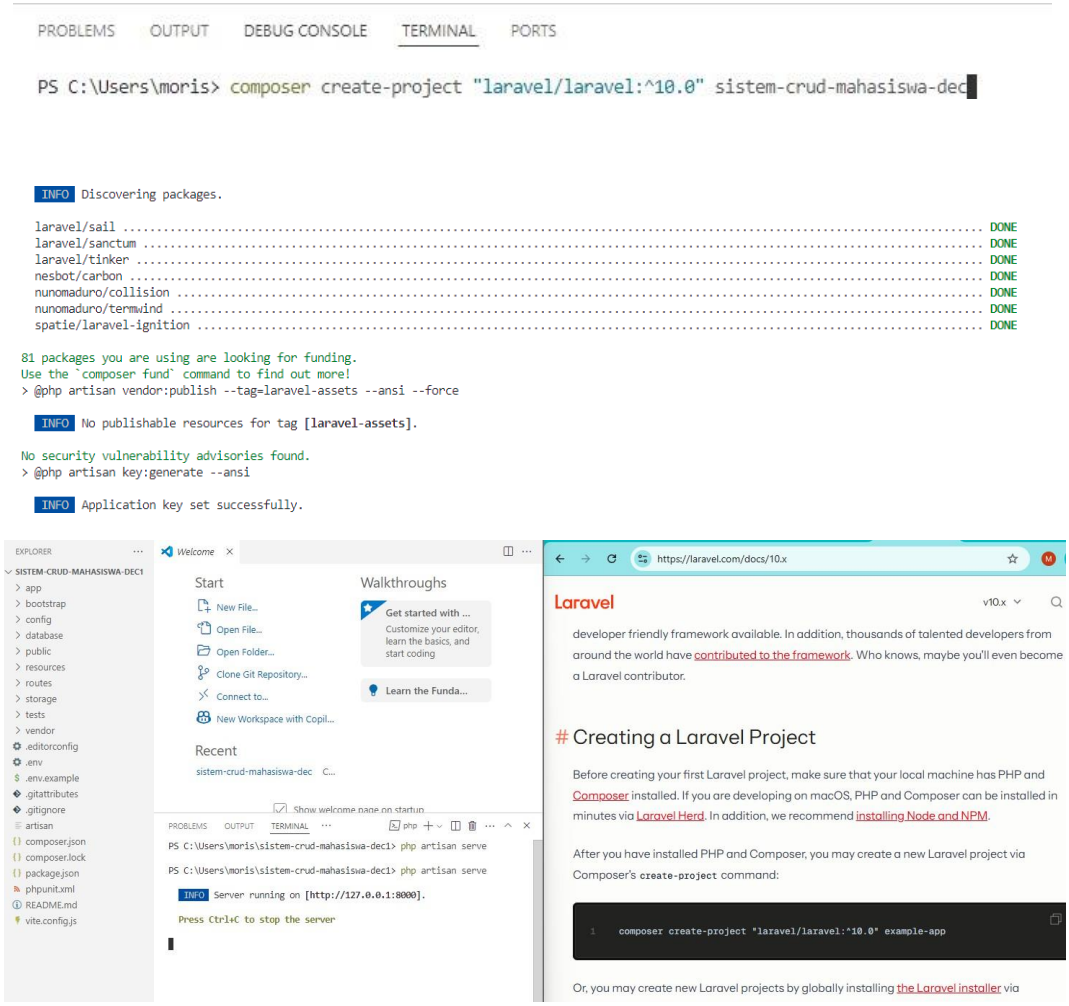
d. Menjalankan XAMPP.



Gambar 2.4 Menjalakan dan Konfigurasi XAMPP.

Pada gambar 2.4 ditunjukan bahwa sebelum memulai proyek Laravel, beberapa langkah persiapan perlu dilakukan terlebih dahulu. Salah satunya adalah menjalankan XAMPP dan memastikan modul Apache serta MySQL sudah diaktifkan. Selanjutnya, melalui menu Config, file konfigurasi php.ini dibuka dan mengaktifkan ekstensi zip dengan menghapus tanda titik koma (;) pada baris extension=zip. Pengaktifan ekstensi ini agar Laravel dapat menjalankan fungsi-fungsi yang berkaitan dengan pengelolaan arsip dan proses instalasi dependensi menggunakan Composer.

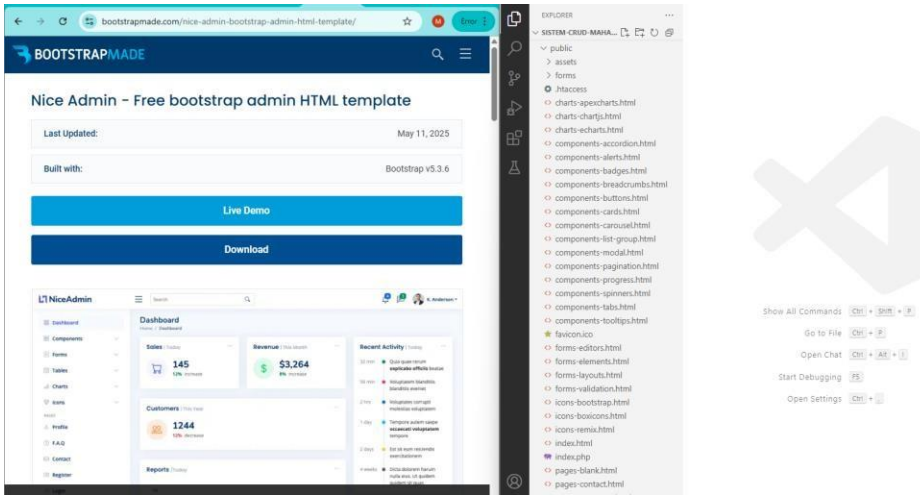
e. Instalasi Laravel dan akses server lokal.



Gambar 2.5 Hasil Installasi Laravel dan Akses Server Local.

Pada gambar 2.5 memperlihatkan tahapan instalasi dan pembuatan proyek Laravel yang dilakukan melalui terminal. Proses tersebut dilaksanakan dengan memanfaatkan perintah composer create-project "laravel/laravel:^10.0" example-app sebagaimana tercantum dalam dokumentasi resmi Laravel (https://laravel.com/docs/10.x). Melalui perintah ini, paket Laravel versi 10 diunduh oleh Composer dan struktur dasar direktori proyek disusun secara otomatis. Setelah instalasi selesai, direktori yang telah terbentuk diakses, dan server lokal dijalankan dengan perintah php artisan serve melalui alamat http://localhost:8000. Apabila proses instalasi berhasil, maka halaman dashboard Laravel akan ditampilkan, yang menjadi indikator bahwa instalasi telah dilakukan dengan baik.

f. Instalasi Boostrap.



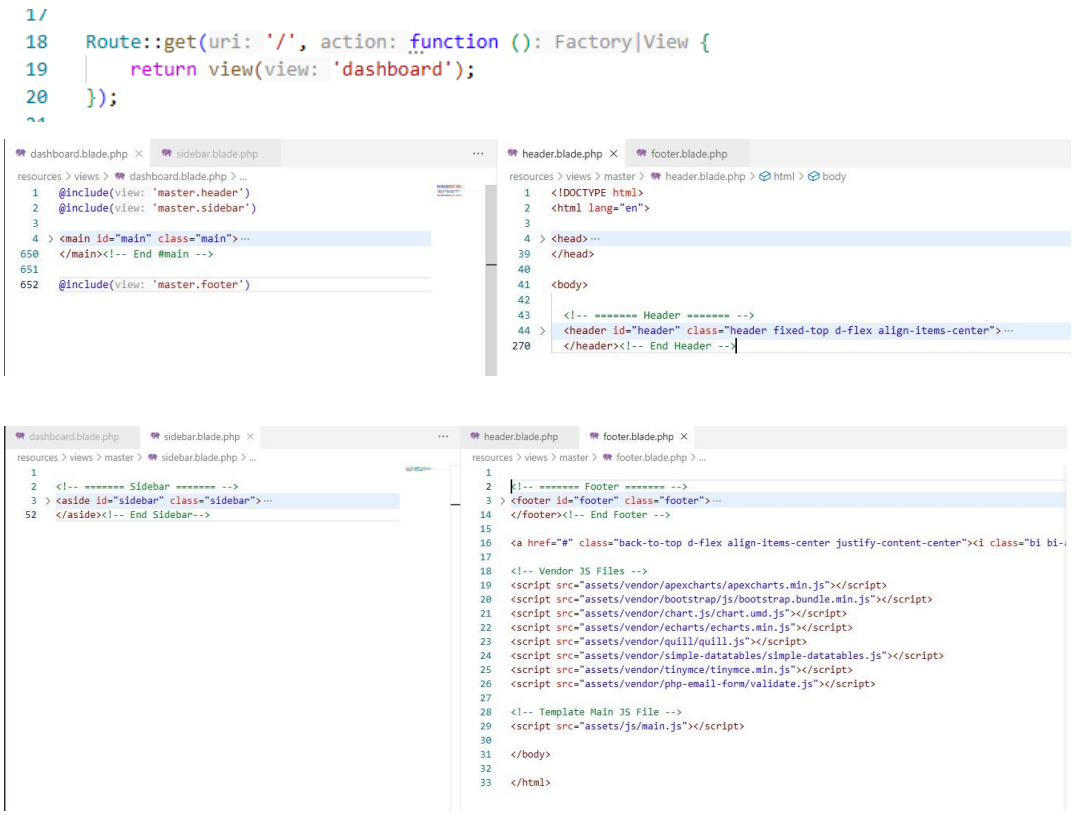
Gambar 2.6 Hasil Instalasi Boostrap.

Pada gambar 2.6 merupakan hasil dari proses instalasi template Bootstrap yang telah berhasil diterapkan. Proses instalasi template ini dilakukan dengan terlebih dahulu mengunduh berkas template dari situs penyedia, kemudian berkas tersebut diekstrak dari bentuk arsipnya. Setelah diekstrak, seluruh isi template disalin ke dalam direktori proyek Laravel, khususnya ke dalam folder public. Dengan demikian, struktur antarmuka pengguna yang telah disediakan oleh template dapat langsung diakses dan digunakan dalam pengembangan.

3. Slicing template UI menggunakan Boostrap

Proses slicing UI menggunakan Bootstrap dimulai dengan membuat dashboard sebagai halaman utama, kemudian mengarahkan routing ke halaman tersebut agar dapat diakses dengan mudah. Setelah itu, dilakukan pemecahan struktur halaman menjadi tiga bagian utama yaitu, header, sidebar, dan footer. Pada tahap slicing, file terpisah dibuat untuk setiap komponen menggunakan template Blade Laravel, seperti header.blade.php, sidebar.blade.php, dan footer.blade.php, yang digunakan sebagai template utama untuk seluruh halaman.

a. Konfigurasi routing dasar dan slicing UI.



Gambar 2.7 Konfigurasi routing dan slicing UI halaman utaman.

Pada gamabar 2.7 merupakan hasil konfigurasi routing serta proses slicing antarmuka pengguna (UI) pada halaman utama. Struktur kode telah dipisahkan ke dalam beberapa file agar lebih terorganisir. File header hanya memuat source kode yang berkaitan dengan bagian awal hingga akhir dari element header, sedangkan file sidebar bersisi keseluruhan kode sidebar, dan file footer memuat bagian footer secara utuh. Pada file utama dashboard, hanya disisakan kode utama (main) bagian konten, sementara header, sidebar, footer disisipkan melalui direktif @include(‘’). Penggunaan @extends tidak diperlukan dalam konteks ini karena pendekatan yang digunakan tidak mengandalkan pewarisan dari layout master, melainkan meggunakan penyisipan secara langsung komponen tampailan secara moduler. Pendekatan ini diterapkan agar proses pengembangan lebih muda, terstruktur, dan mempermudah pemeliharaan kode. Konfigurasi routing juga dilakukan melalui file web.php. secara bawaan, Laravel sudah menyediakan satu rute dasar (root) berupa

Route::get('/', function () { return view('welcome'); });, yang akan menampilkan halaman welcome.blade.php pada saat aplikasi pertama kali diakses. Pada implementasi ini, rute diubah agar mengarahkan pengguna langsung ke halaman utama yang telah dibuat, yaitu dashboard. Perubahan dilakukan dengan mengganti baris kode { return view('welcome'); } menjadi { return view('dashboard'); }. Dengan konfigurasi ini, saat alamat dasar aplikasi diakses maka akan menampilkan tampilan halaman dashboard.

b. Membuat halaman Prodi, Mahasiswa dan konfigurasi routing.

```
17
18  Route::get(uri: '/', action: function (): Factory|View {
19      return view(view: 'dashboard');
20  });
21
22  Route::get(uri: '/create-mahasiswa', action: function (): Factory|View {
23      return view(view: 'mahasiswa.create');
24  });
25
26  Route::get(uri: '/mahasiswa', action: function (): Factory|View {
27      return view(view: 'mahasiswa.index');
28  });
29
30  Route::get(uri: '/create-prodi', action: function (): Factory|View {
31      return view(view: 'prodi.create');
32  });
33
34  Route::get(uri: '/mahasiswa', action: function (): Factory|View {
35      return view(view: 'prodi.index');
36  });
```

index.blade.php

resources > views > prodi > index.blade.php > ...

```
1   @include(view: 'master.header')
2   @include(view: 'master.sidebar')
3
4   <main id="main" class="main">···
60  </main><!-- End #main -->
61
62  @include(view: 'master.footer')
```

create.blade.php

resources > views > mahasiswa > create.blade.php > ...

```
1   @include(view: 'master.header')
2   @include(view: 'master.sidebar')
3
4   <main id="main" class="main">···
70  </main><!-- End #main -->
71
72  @include(view: 'master.footer')
```

Gambar 2.8 Pembuatan halaman Prodi, Mahasiswa , dan konfigurasi routing.

Pada gambar 2.8 merupakan hasil pembuatan dua file baru, yaitu index.blade.php dan create.blade.php, masing-masing untuk entitas Prodi dan Mahasiswa. Setiap file dirancang dengan pendekatan moduler, seperti halaman dashboard yang mana tampilan seperti header, sidebar, dan footer dipisahkan ke dalam file tersendiri dan

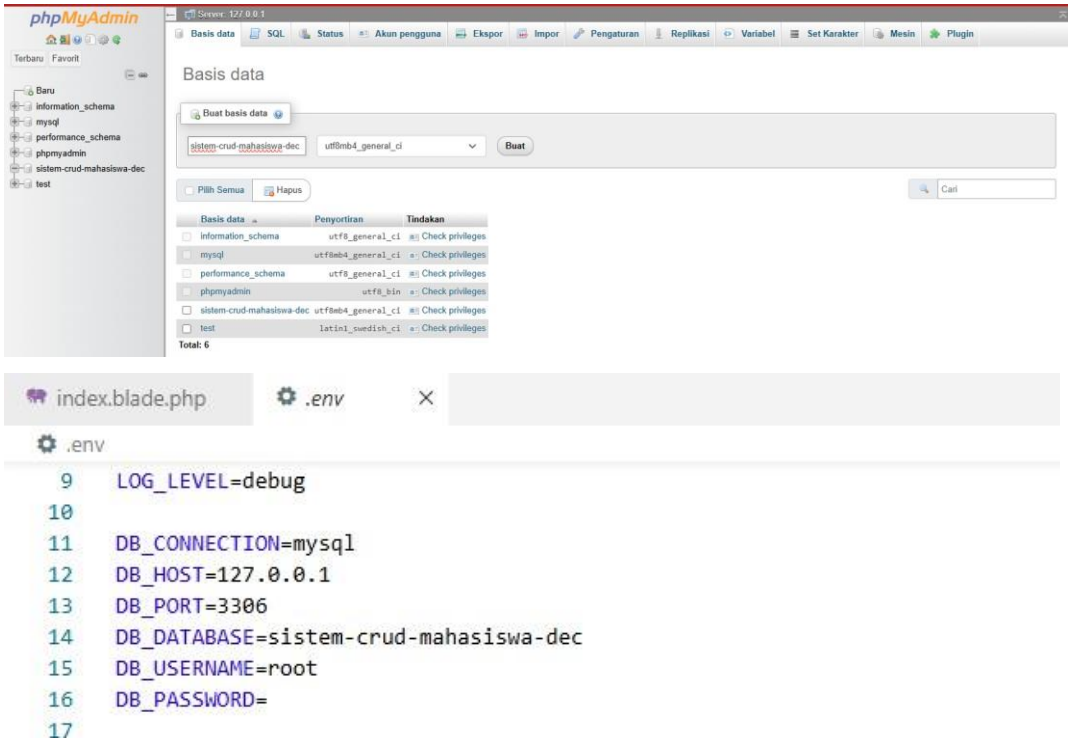
disisipkan ke halaman utama menggunakan direktif @include. Komponen-komponen tersebut mengikuti struktur dari tampilan utama dashboard.blade.php, sehingga keseragaman antarmuka antar halaman dapat terjaga. Pada bagian utama halaman (main), disediakan ruang khusus untuk memuat konten dinamis sesuai kebutuhan. File index.blade.php digunakan untuk menampilkan daftar data dalam bentuk tabel, sedangkan create.blade.php memuat elemen formulir yang digunakan untuk menambahkan data baru. Selain itu, konfigurasi routing juga perlu dilakukan. Masing-masing tampilan diberi rute khusu yang menghubungkan alamat URL dengan file tampilan. Contohnya, rute /create-mahasiswa akan diarahkan ke tampilan mahasiswa.create, dan rute /mahasiswa diarahkan ke tampilan mahasiswa.index. Pola yang sama diterapkan pula untuk entitas Prodi. Dengan konfigurasi ini, Laravel dapat secara otomatis menampilkan halaman yang sesuai berdasarkan URL yang diakses, sehingga navigasi dalam aplikasi menjadi lebih terarah dan mendukung alur CRUD.

2.3.2 Dasar Laravel dan CRUD

1. Membuat CRUD sederhana.

Sebelum memulai proses pembuatan fitur CRUD sederhana, basis data perlu disiapkan terlebih dahulu. Jenis basis data yang digunakan dapat dipilih secara bebas, namun dalam implementasi ini digunakan MySQL. Pemilihan ini disesuaikan dengan penggunaan XAMPP sebagai server lokal, yang telah dilengkapi dengan Apache serta phpMyAdmin sebagai antarmuka grafis untuk pengelolaan basis data. Basis data yang dibuat diberi nama sistem-crud-mahasiswa-dec, dan digunakan untuk menyimpan serta mengelola data entitas program studi dan mahasiswa secara terstruktur. Namun, sebagai bentuk contoh dalam penerapan fitur CRUD, hanya entitas Prodi yang ditampilkan dan dijadikan fokus utama pada tahap pembahasan ini.

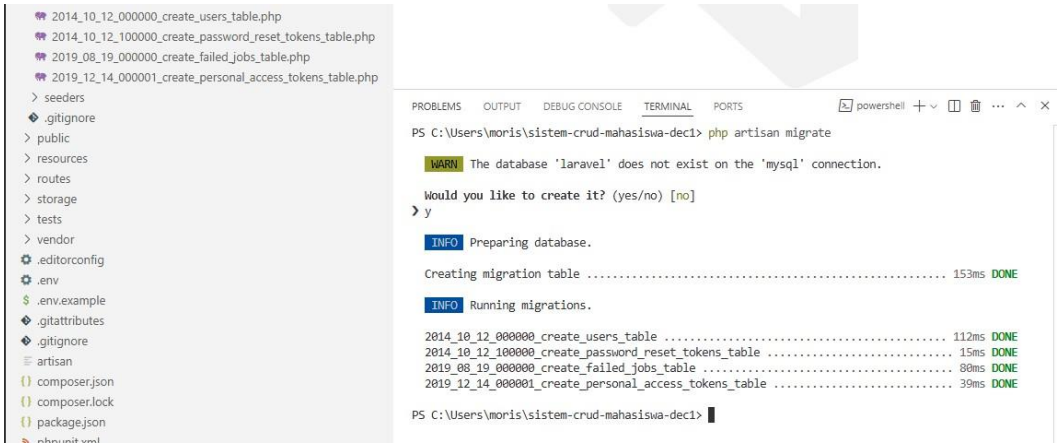
a. Membuat database dan konfigurasi file .env.



Gambar 2.9 Membuat database dan Konfigurasi file .env.

Pada gambar 2.9 merupakan tampilan hasil pembuatan basis data serta konfigurasi pada file .env. Konfigurasi dilakukan agar aplikasi Laravel dapat terhubung dengan basis data yang telah dibuat, yaitu sistem-crud-mahasiswa-dec. Konfigurasi hanya mencankup jenis koneksi basis data (DB_CONNECTION) yaitu MySQL, alamat host (DB_HOST) 127.0.0.1, port (DB_PORT) 3306, dan nama basis data (DB_DATABASE) sistem-crud-mahasiswa-dec. Pada bagian informasi nama pengguna (DB_USERNAME) root dan kata sandi (DB_PASSWORD) dibiarkan secara default (bawaan), karena proyek masih dalam tahap pengembangan dan dijalankan di server lokal, maka kredensial (data login) dibiarkan menggunakan pengaturan bawaan dari XAMPP. Konfigurasi ini cukup dilakukan satu kali di awal proyek dan akan digunakan selama proses pengembangan berlangsung, kecuali jika terjadi perubahan pada struktur koneksi, kredensial, atau pemindahan ke server lain.

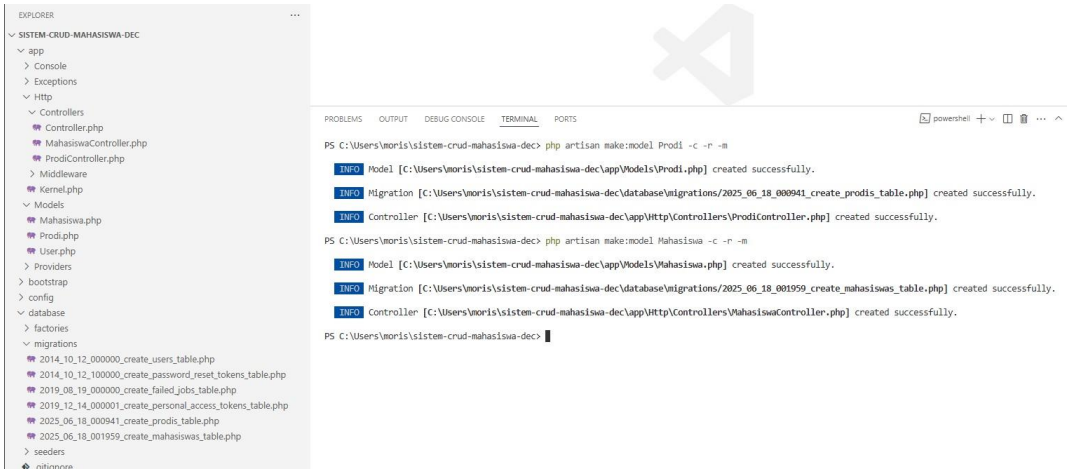
b. Migrasi database.



Gambar 2.10 Hasil migrasi database.

Pada gambar 2.10 merupakan hasil proses migrasi basis data menggunakan Laravel. Secara default (bawaan), Laravel sudah menyediakan beberepa file migrasi awal yang dapat ditemukan di dalam direktori database/migrations. Proses migrasi dijalankan melalui command (perintah) php artisan migrate pada terminal, dan dalam hal ini dilakukan melalui PowerShell. Saat perintah dijalankan, Laravel akan memeriksa apakah basis data yang dituju telah tersedia atau belum. Jika belum, sistem akan menampilkan konfirmasi untuk membuatnya terlebih dahulu. Setelah basis data tersedia, Laravel akan secara otomatis mengeksekusi seluruh file migrasi yang ada, lalu membuat struktur tabel sesuai dengan isi migrasi ke dalam basis data yang telah dikonfigurasi sebelumnya. Selain perintah tersebut, terdapat juga perintah lainya yang sering digunakan apabila terjadi kesalahan dalam proses migrasi dan ingin kembalikan ke keadaan sebelumnya, seperti php artisan migrate:rollback yang digunakan untuk membatalkan migrasi terakhir, yaitu seluruh migrasi yang tergabung dalam batch (satu kelompok).

c. Membuat entitas berserta tabel prodis dan mahasiswas.



Gambar 2.11 Hasil membuat entitas berserta tabel.

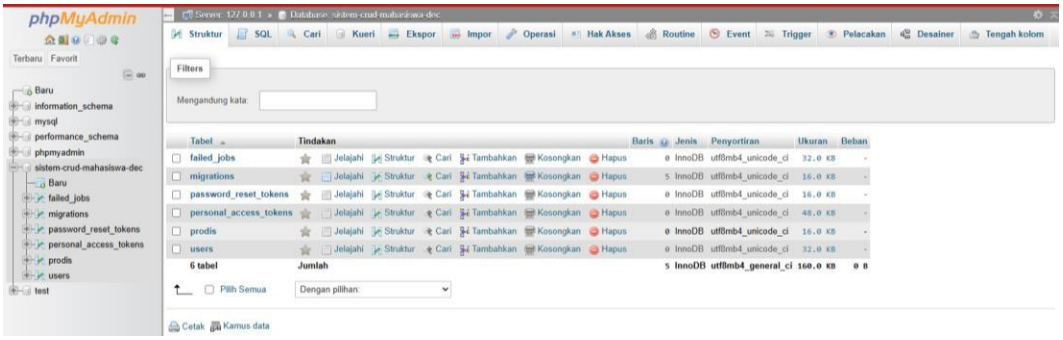
Pada gambar 2.11 merupakan hasil pembuatan entitas bertserta tabel Prodi dan Mahasiswa secara otomatis tanpa harus membuatya langsung melalui MySQL. Proses ini dilakukan dengan menjalankan command (perintah) php artisan make:model Prodi -c -r -m melalui terminal. Perintah tersebut merupakan satu paket dengan beberapa flag, yaitu -c untuk membuat file controller, -r yang menunjukkan bahwa controller akan dibuat sebagai resource controller sehingga secara otomatis menyertakan fungsi-fungsi standar seperti index, create, store, show, edit, update, dan destroy sekaligus mendukung pembuatan rute resource, serta -m untuk menghasilkan file migrasi yang akan digunakan untuk menentukan struktur kolom dan tipe data pada tabel di basis data. Setelah perintah dijalankan, Laravel secara otomatis membuat file model Prodi.php, file controller ProdiController.php, dan file migrasi untuk tabel prodis. Proses serupa juga diterapkan untuk entitas Mahasiswa, dengan menjalankan perintah yang sama namun mengganti nama entitas menjadi “Mahasiswa”. Pendekatan ini mempermudah proses pengembangan karena seluruh komponen dasar telah disiapkan dalam satu langkah terintegrasi.

d. Migrasi tabel prodis.

database > migrations > 2025_06_18_000941_create_prodis_table.php > ...

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create(table: 'prodis', callback: function (Blueprint $table): void {
            $table->id();
            $table->string(column: 'nama');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists(table: 'prodis');
    }
};
```

```
PS C:\Users\moris\sistem-crud-mahasiswa-dec> php artisan migrate --path=/database/migrations/2025_06_18_000941_create_prodis_table.php

  INFO  Running migrations.

  2025_06_18_000941_create_prodis_table ..... 156ms DONE
```



Gambar 2.12 Hasil migrasi tabel prodis.

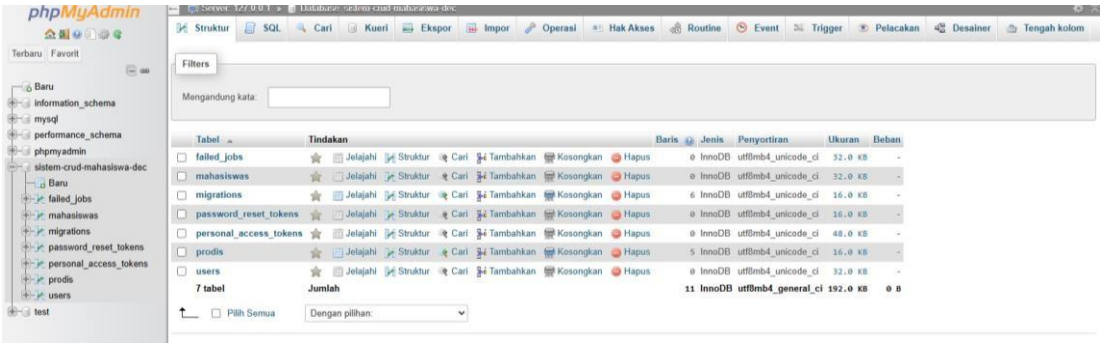
Pada gambar 2.12 merupakan proses dan hasil migrasi tabel prodis yang secara otomatis dihasilakan sebelumnya. Pada file tabel prodi sudah terdapat baris kode dasar seperti $table->id(); untuk membuat kolom ID sebagai primary key dan $table->timestamps(); yang akan menambahkan dua kolom otomatis, yaitu created_at dan updated_at. Agar tabel Prodi memiliki kolom nama program studi, maka perlu ditambahkan baris $table->string('nama'); ke dalam fungsi up(). Dengan

menambahkan perintah ini, maka Laravel akan menyertakan kolom nama bertipe string ke dalam struktur tabel saat proses migrasi dijalankan. Kolom tersebut selain ID dan timestamp. Hal ini memudahkan pengembang agar tidak perlu menulis ulang struktur dari awal, cukup hanya menambahakan kolom yang dibutuhkan sesuai kebutuhan. Berikutnya menjalankan perintah php artisan migrate--path=/database/migrations/, fungsi perintah ini untuk menjalakan migrasi secara spesifik hanya pada file tertentu, bukan seluruh file migrasi yang ada pada direktori database/migrations. Dalam kasus ini, Laravel secara otomatis akan mengeksekusi isi dari file 2025_04_13_092857_create_prodis_table.php dan membuat tabel Prodi didalam basis data sesuai struktur yang telah ditentukan. Perintah ini harus dijalankan dari root direktori proyek Laravel, dan pastikan nama file serta path ditulis tepat seperti yang ada dalam folder. Jika ada kesalahan dalam penulisan path atau nama file, Laravel akan menampilkan error bahwa file migrasi tidak ditemukan.

e. Migrasi table mahasiswas.

```
    */
    public function up(): void
    {
        Schema::create(table: 'mahasiswas', callback: function (Blueprint $table): void {
            $table->id();
            $table->string(column: 'nama');
            $table->unsignedBigInteger(column: 'prodi_id');
            $table->foreign(columns: 'prodi_id')->references(columns: 'id')->on(table: 'prodis')->onDelete(action: 'cascade');
            $table->string(column: 'whatsapp');
            $table->timestamps();
        });
    }
```

```
PS C:\Users\moris\sistem-crud-mahasiswa-dec> php artisan migrate --path=/database/migrations/2025_06_18_001959_create_mahasiswas_table.php

  INFO  Running migrations.

  2025_06_18_001959_create_mahasiswas_table ..... 98ms DONE
```



Gambar 2.13 Hasil migrasi tabel mahasiswas.

Pada gambar 2.13 merupakan hasil migrasi tabel Mahasiswas. Struktur dasar seperti $table->id();, $table->timestamps();, dan $table->string('nama'); tetap

dipertahankan. Namun, pada tabel mahasiswa dilakukan penambahan beberapa kolom baru. Kolom $table->unsignedBigInteger('prodi_id'); digunakan untuk menyimpan ID dari program studi yang menjadi relasi. Setelah itu, relasi antar tabel ditentukan dengan $table->foreign('prodi_id')->references('id')->on('prodis')->onDelete('cascade'); yang berarti kolom prodi_id pada tabel mahasiswa akan menjadi foreign key yang merujuk ke kolom id di tabel prodis. Kata kunci onDelete('cascade') digunakan agar jika suatu data pada tabel prodis dihapus, maka seluruh mahasiswa yang terkait dengannya akan ikut terhapus otomatis. Selain itu, ditambahkan juga $table->string('whatsapp'); untuk menyimpan nomor kontak mahasiswa. Dengan pendekatan ini, struktur tabel menjadi lebih rapi dan relasi antar entitas bisa terkelola dengan baik.

f. Penyederhanaan routing.

```
18  Route::get(uri: '/', action: function (): Factory|View {
19      return view(view: 'dashboard');
20  });
21
22  // Route::get('/create-mahasiswa', function () {
23  //     return view('mahasiswa.create');
24  // });
25
26  // Route::get('/mahasiswa', function () {
27  //     return view('mahasiswa.index');
28  // });
29
30  Route::resource(name: 'mahasiswa', controller: MahasiswaController::class);
31
32  // Route::get('/create-prodi', function () {
33  //     return view('prodi.create');
34  // });
35
36  // Route::get('/mahasiswa', function () {
37  //     return view('prodi.index');
38  // });
39
40  Route::resource(name: 'prodi', controller: ProdiController::class);
```

```
PS C:\Users\moris\sistem-crud-mahasiswa-dec> php artisan route:list

  GET|HEAD        / ..........................................................................................................
  POST            _ignition/execute-solution ..... ignition.executeSolution › Spatie\LaravelIgnition › ExecuteSolutionController
  GET|HEAD        _ignition/health-check ..... ignition.healthCheck › Spatie\LaravelIgnition › HealthCheckController
  POST            _ignition/update-config ..... ignition.updateConfig › Spatie\LaravelIgnition › UpdateConfigController
  GET|HEAD        api/user ...................................................................................................
  GET|HEAD        mahasiswa ..... mahasiswa.index › MahasiswaController@index
  POST            mahasiswa ..... mahasiswa.store › MahasiswaController@store
  GET|HEAD        mahasiswa/create ..... mahasiswa.create › MahasiswaController@create
  GET|HEAD        mahasiswa/{mahasiswa} ..... mahasiswa.show › MahasiswaController@show
  PUT|PATCH       mahasiswa/{mahasiswa} ..... mahasiswa.update › MahasiswaController@update
  DELETE          mahasiswa/{mahasiswa} ..... mahasiswa.destroy › MahasiswaController@destroy
  GET|HEAD        mahasiswa/{mahasiswa}/edit ..... mahasiswa.edit › MahasiswaController@edit
  GET|HEAD        prodi ..... prodi.index › ProdiController@index
  POST            prodi ..... prodi.store › ProdiController@store
  GET|HEAD        prodi/create ..... prodi.create › ProdiController@create
  GET|HEAD        prodi/{prodi} ..... prodi.show › ProdiController@show
  PUT|PATCH       prodi/{prodi} ..... prodi.update › ProdiController@update
  DELETE          prodi/{prodi} ..... prodi.destroy › ProdiController@destroy
  GET|HEAD        prodi/{prodi}/edit ..... prodi.edit › ProdiController@edit
  GET|HEAD        sanctum/csrf-cookie ..... sanctum.csrf-cookie › Laravel\Sanctum › CsrfCookieController@show

                                                                                  Showing [20] routes
```

Gambar 2.14 Penyederhanaan konfigurasi routing.

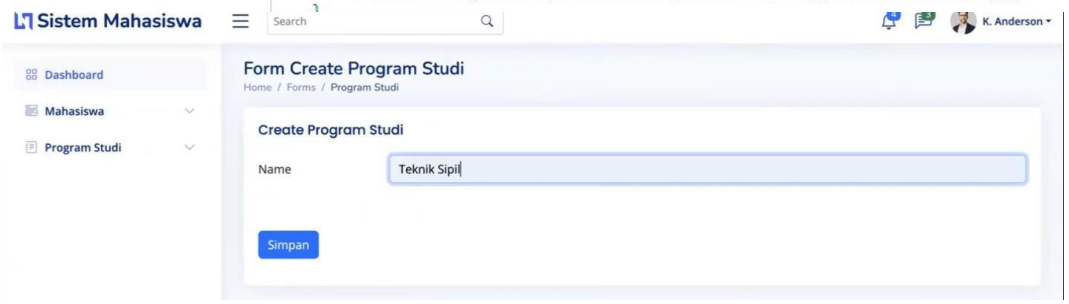
Pada gambar 2.14 merupakan penyederhanaan konfigurasi routing menggunakan fitur Route::resource() dalam Laravel. Sebelumnya, setiap rute untuk operasi CRUD perlu ditulis secara manual, seperti rute untuk menampilkan form create, menampilkan data, dan sebagainya. Penggunaan Route::resource('mahasiswa', MahasiswaController::class); dan Route::resource('prodi', ProdiController::class);, seluruh rute untuk operasi dasar CRUD dapat dihasilkan secara otomatis hanya dengan satu baris per entitas. Penyederhanaan ini tidak hanya mempersingkat kode, tetapi juga memastikan bahwa struktur rute lebih konsisten. Selanjutnya, perintah php artisan route:list digunakan untuk menampilkan seluruh daftar rute yang aktif dan terdaftar didalam aplikasi Laravel untuk entitas Mahasiswa maupun Prodi. Secara otomatis Laravel sudah membuat tujuh rute dasar, masing-masing dengan keperluan seperti index, create, store, show, edit, update, dan destroy. Hal ini karena Route::resource() adalah salah satu kelebihan utama Laravel yang menyediakan cara efisien dalam mendefenisikan routing CRUD. Dengan adanya fitur ini, seluruh rute tidak harus didefinisikan satu per satu, yang tentu akan memakan waktu dan rentan kesalahan.

g. Membuat From Create dan Read untuk entitas Prodi.

```
<!-- General Form Elements -->
<form action="{{ route(name: 'prodi.store')}}" method="POST">
    @csrf
    <div class="row mb-3">
        <label for="inputText" c
        <div class="col-sm-10">
        <input type="text" name=
        </div>
    </div>
    <br>
    <br>
    <div class="row mb-3">
        <div class="col-sm-10">
        <button type="submit" class="btn btn-primary">Simpan</button>
        </div>
    </div>
</form>
<!-- End General Form E
```

```
0 references | 0 overrides
public function store(Request $request): Redirector|RedirectResponse
{
    // dd(vars: $request->all());
    Prodi::create(attributes: [
        'nama' => $request->nama
    ]);

    return redirect(to: 'prodi');

}
```

```
public function index(): Factory|View
{
    $data = Prodi::all();
    return view(view: "prodi.index",data: compact(var_name: 'data'));
```



Gambar 2.15 Form Create untuk entitas Prodi.

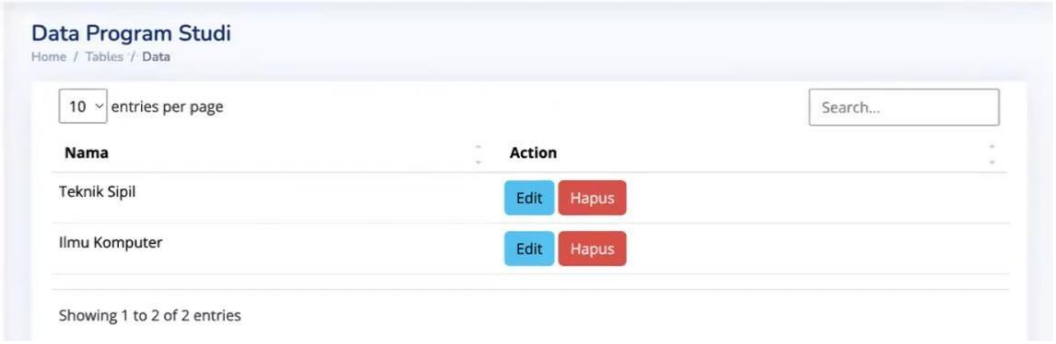
Pada gambar 2.15 merupakan pembuatan dan hasil konten utama (main) untuk entitas Prodi dalam bentuk form input yang disusun di dalam file create.blade.php menggunakan Blade Laravel. Form ini dirancang untuk mengirim data ke route prodi.store dengan metode POST, dan dilengkapi proteksi @csrf sebagai fitur keamanan. Struktur form memanfaatkan sistem grid Bootstrap dengan satu input untuk nama program studi serta tombol bertipe submit, dibungkus dalam komponen card agar tampilan lebih terstruktur. Laravel mewajibkan penyertaan token CSRF untuk semua permintaan yang mengubah data, termasuk metode POST, PUT, PATCH, dan DELETE. Karena HTML hanya mendukung GET dan POST, maka untuk metode lain Laravel menggunakan teknik spoofing, yaitu menyisipkan direktif seperti @method('PUT') pada form agar Laravel mengenali maksud operasinya. Setelah data dikirim, method store() pada ProdiController akan menangani penyimpanan ke database menggunakan perintah Prodi::create, dan pengguna kemudian diarahkan kembali ke halaman utama entitas Prodi melalui redirect('prodi'). Untuk menampilkan data yang telah tersimpan, digunakan method

index() yang memuat seluruh data dari tabel prodis menggunakan Prodi::all(), lalu mengirimkannya ke view prodi.index melalui fungsi compact, sehingga alur proses input hingga penampilan data dapat berlangsung secara terpadu, aman, dan efisien.

h. Membuat Fitur Delete pada entitas Prodi.

```
<tbody>
    @foreach ($data as $row)
    <tr>
        <td>{{ $row->nama }}</td>
        <td>
        <form action="{{ route(name: 'prodi.destroy', parameters: $row->id) }}" method="POST">
            @method('DELETE')
            @csrf
        <a href= "{{ route(name: 'prodi.edit', parameters: $row->id) }}" class="btn btn-info">Edit</a>
        <button type="submit" class="btn btn-danger">Hapus</button>
        </form>
        </td>
    </tr>
    @endforeach
</tbody>
```

```
    public function destroy(Prodi $prodi): Redirector|RedirectResponse
    {
        $prodi ->delete();
        return redirect(to: 'prodi');
    }
```



Gambar 2.16 Membuat Fitur Delete pada entitas Prodi.

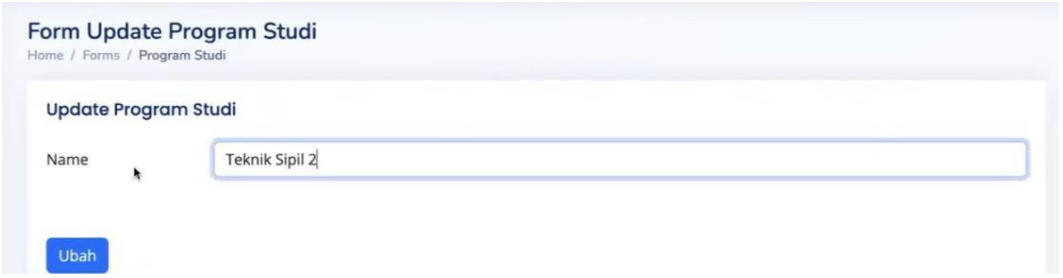
Pada gambar 2.16 merupakan hasil implementasi fitur penghapusan data (delete) pada Laravel yang mencakup dua bagian utama, yaitu tampilan antarmuka (Blade) dan logika backend pada controller. Di bagian tampilan, data entitas Prodi disusun dalam bentuk tabel dengan menggunakan perulangan @foreach, di mana setiap baris menyajikan informasi program studi beserta dua tombol aksi, yaitu Edit dan Hapus. Tombol Edit disusun melalui elemen <a> yang mengarah ke route prodi.edit dengan menyertakan parameter ID, sedangkan tombol Hapus ditempatkan dalam elemen <form> yang diarahkan ke route prodi.destroy menggunakan metode POST,

namun ditetapkan secara eksplisit sebagai DELETE melalui direktif @method('DELETE'), serta dilengkapi dengan proteksi @csrf guna menjamin keamanan permintaan. Pada sisi controller, method destroy() dimanfaatkan untuk menangani proses penghapusan data, di mana Laravel secara otomatis mencocokkan ID dari form dengan data model Prodi melalui mekanisme route model binding. Setelah data yang dimaksud ditemukan, perintah $prodi->delete() dijalankan untuk menghapus data dari basis data, dan pengguna diarahkan kembali ke halaman utama entitas Prodi dengan perintah return redirect('prodi');.

i. Membuat Fitur Update pada entitas Prodi.

```
<form action="{{ route(name: 'prodi.update', parameters: $prodi->id)}}" method="POST">
    @method('PUT')
    @csrf
    <div class="row mb-3">
        <label for="inputText" class="col-sm-2 col-form-label">Nama</label>
        <div class="col-sm-10">
            <input type="text" name="nama" class="form-control" value="{{$prodi->nama}}">
        </div>
    </div>
    <br>
    <br>
    <div class="row mb-3">
        <div class="col-sm-10">
            <button type="submit" class="btn btn-primary">Ubah</button>
        </div>
    </div>
</form>
```

```
public function update(Request $request, Prodi $prodi): Redirector|RedirectResponse
{
    $prodi->update(attributes: [
        'nama' => $request->nama
    ]);

    return redirect(to: 'prodi');
}
```



Gambar 2.17 Membuat Fitur Update pada entitas Prodi.

Pada Gambar 2.17 ditampilkan implementasi fitur pembaruan data (update) untuk entitas Prodi, yang mencakup bagian tampilan (Blade) dan logika di controller. Di

sisi tampilan, form pembaruan pada file edit.blade.php diarahkan ke route prodi.update dengan parameter ID. Karena HTML hanya mendukung metode GET dan POST, Laravel menggunakan @method('PUT') untuk menandai proses update, serta menyertakan @csrf sebagai perlindungan terhadap CSRF. Struktur form mirip dengan form pembuatan data (create), namun terdapat dua perbedaan. Penggunaan @method('PUT'), dan nilai input yang sudah terisi otomatis menggunakan value="{{ $prodi->nama }}", agar dapat diedit. Di sisi controller, method update() menerima data dari form melalui objek Request dan model Prodi yang memungkinkan Laravel secara otomatis mencocokkan parameter di URL dengan data model yang sesuai di database (binding) otomatis. Data kemudian diperbarui di database menggunakan $prodi->update(['nama' => $request->nama]);, lalu pengguna diarahkan kembali ke halaman prodi. Terakhir, agar alur aplikasi lebih konsisten dan mudah dipahami, penamaan route pada sidebar juga sebaiknya disesuaikan dengan nama route yang terdaftar, sehingga navigasi pengguna menjadi lebih terarah dan sesuai dengan rute yang berlaku di sistem.

2. Relasi antar entitas database.

Pada bagian sebelumnya, relasi antara tabel mahasiswas dan prodis sudah ditetapkan sebagaimana ditampilkan pada gambar 13, dimana struktur tabel Mahasiswas telah dilengkapi dengan kolom prodi_id yang mana berperan sebagai foreign key untuk menghubungkan kedua entitas. Pada bagian ini, pembahasan akan difokuskan lebih lanjut pada konsep relasi antar entitas dalam basis data, tidak hanya dari sisi struktur migrasi, tetapi juga dari implementasinya dalam model Laravel. Selain itu, akan dijelaskan bagaimana relasi ini memengaruhi proses pengambilan data serta menjaga keterkaitan dan konsistensi informasi antar tabel yang saling berelasi. Penyesuaian kode juga dilakukan pada masing-masing model untuk mendukung integrasi relasi antar entitas secara optimal.

a. Konfigurasi model Prodi.

```
6 references | 0 implementations
8   class Prodi extends Model
9   {
10      use HasFactory;
        0 references
11      protected $guarded = ['id'];
12  }
13
```

Gambar 2.18 Konfigurasi model Prodi.

Pada gambar 2.18 merupakan konfigurasi model Prodi. sebelumnya telah dijelaskan bahwa model ini dibuat menggunakan perintah php artisan make:model Prodi -c -r -m, yang secara otomatis menghasilkan file model lengkap dengan struktur dasar serta file controller, resource, dan migrasinya. Untuk menyepurmakan model ini, ditambahankan baris kode protected $guarded = ['id'];. Baris tersebut berfungsi untuk melindungi atribut id dari proses mass assignment, karena kolom tersebut bersifat auto increment artinya nilainya akan dihasilkan secara otomatis oleh sistem basis data, bukan oleh input pengguna. Dengan demikian, Laravel tidak akan mengizinkan pengisian langsung pada kolom tersebut melalui operasi seperti create() atau update() secara massal.

b. Konfigurasi model Mahasiswa.

```
9   class Mahasiswa extends Model
10  {
11      use HasFactory;
        0 references
12      protected $guarded = ['id'];
        0 references | 0 overrides
13      public function prodi():BelongsTo{
14          return $this->belongsTo(related: Prodi::class,foreignKey: 'prodi_id');
15      }
16  }
```

Gambar 2.19 Konfigurasi model Mahasiswa.

Pada gambar 2.19 merupakan konfigurasi model Mahasiswa. Seperti halnya pada model Prodi, properti $guarded juga ditetapkan dengan nilai ['id'] untuk mencegah pengisian kolom ID secara massal. Namun, pada model Mahasiswa ini ditambahkan pula sebuah method relasi antarentitas, yaitu public function prodi(): BelongsTo. Method tersebut mengatur bahwa setiap data mahasiswa memiliki keterkaitan langsung dengan satu data program studi, melalui relasi bertipe belongsTo. Relasi

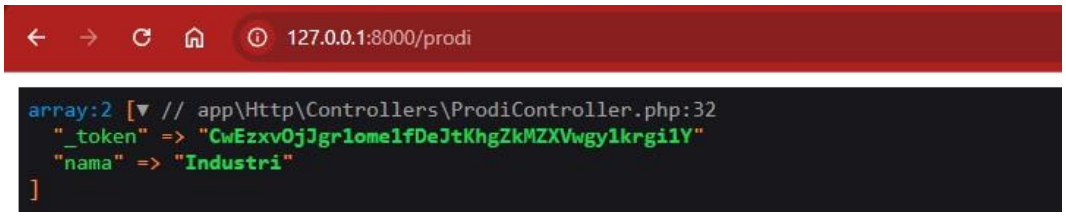
ini mengandalkan kolom prodi_id sebagai foreign key, yang merujuk ke ID dalam tabel Prodi. Dengan menyisipkan baris return $this->belongsTo(Prodi::class, 'prodi_id');, maka model Mahasiswa dapat mengakses data program studi yang bersangkutan melalui pemanggilan $mahasiswa->prodi. Konfigurasi ini mendukung implementasi relasi many-to-one pada basis data, dan sangat membantu dalam menjaga keterhubungan antarentitas pada lapisan model Laravel secara efisien dan konsisten.

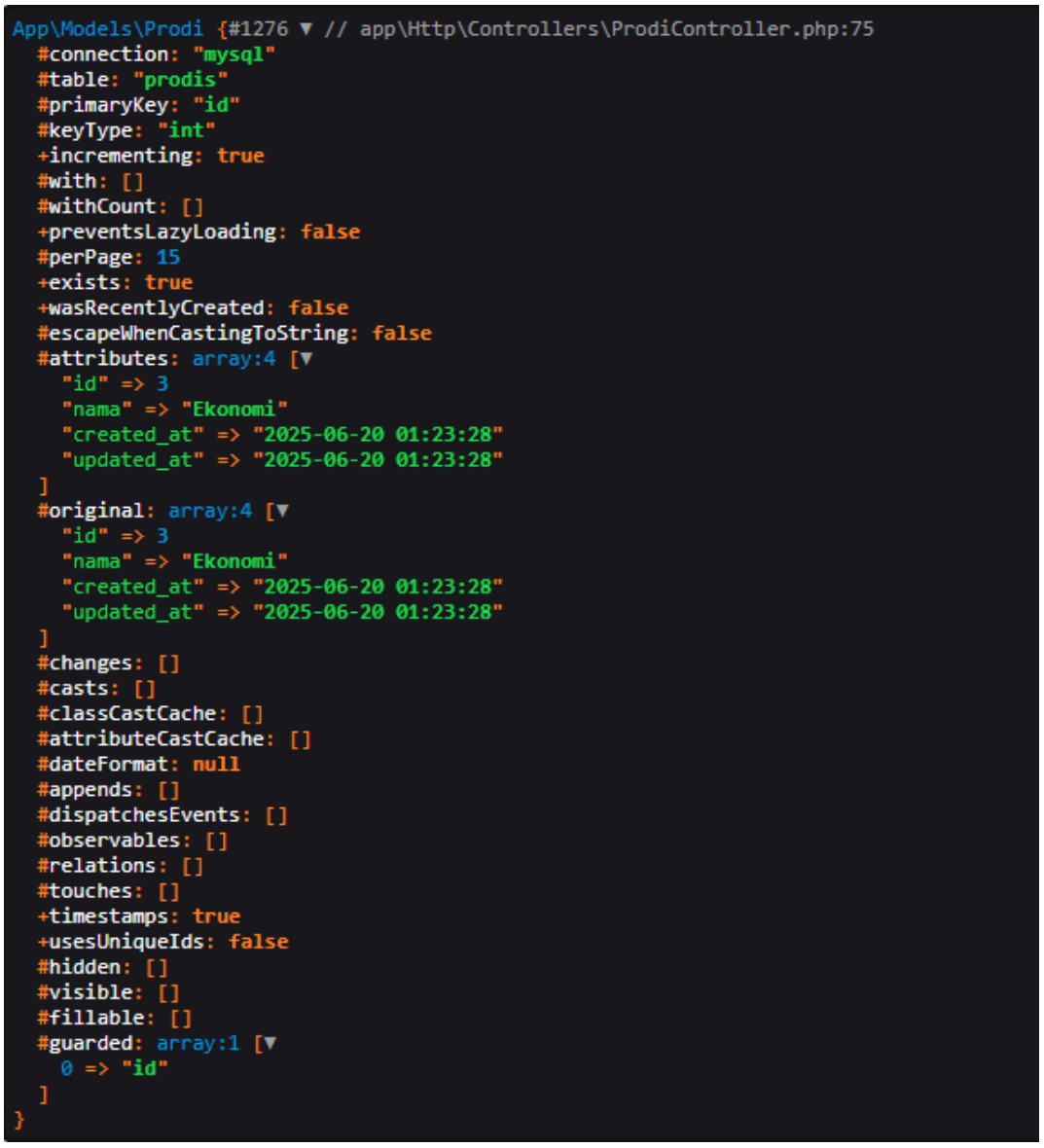
3. Uji coba dan debungging dasar.

Setelah tahap pengembangan selesai, proses uji coba dan debugging dilakukan baik selama proses pengembangan maupun setelahnya. Tujuannya adalah untuk memastikan bahwa seluruh operasi berjalan dengan baik dan sistem telah terhubung dengan basis data secara benar. Proses debugging difokuskan pada bagian controller dari setiap entitas, seperti Mahasiswa dan Prodi, untuk memastikan bahwa setiap fungsinya baik itu untuk pembuatan (create), pembaruan (update), penghapusan (delete), maupun penampilan data (read)—berjalan sebagaimana mestinya. Pengujian juga dapat dilakukan secara bertahap, dengan memeriksa masing-masing operasi secara terpisah sesuai kebutuhan. Dengan demikian, kesalahan logika atau kendala koneksi dapat diidentifikasi dan diperbaiki sebelum sistem dijalankan secara penuh.

a. Uji coba dan debugging.

```
                0 references | 0 overrides
29              public function store(Request $request): never
30              {
31                  dd(vars: $request->all());
32              }
33
```





Gambar 2.20 Hasil Uji coba dan Debugging

Pada Gambar 2.20 ditampilkan hasil dari proses debugging dasar yang dilakukan dalam method store() pada ProdiController. Proses ini bertujuan untuk memastikan bahwa data yang dikirimkan dari form di file create.blade.php melalui metode POST ke route prodi.store telah diterima dengan benar oleh controller. Pada tahap awal, pengecekan dilakukan menggunakan fungsi dd($request->all()); yang akan menampilkan seluruh data form dalam bentuk array dan menghentikan eksekusi program. Data yang ditampilkan biasanya meliputi _token sebagai token keamanan CSRF Laravel, serta nama yang merupakan input pengguna, misalnya “Industri”.

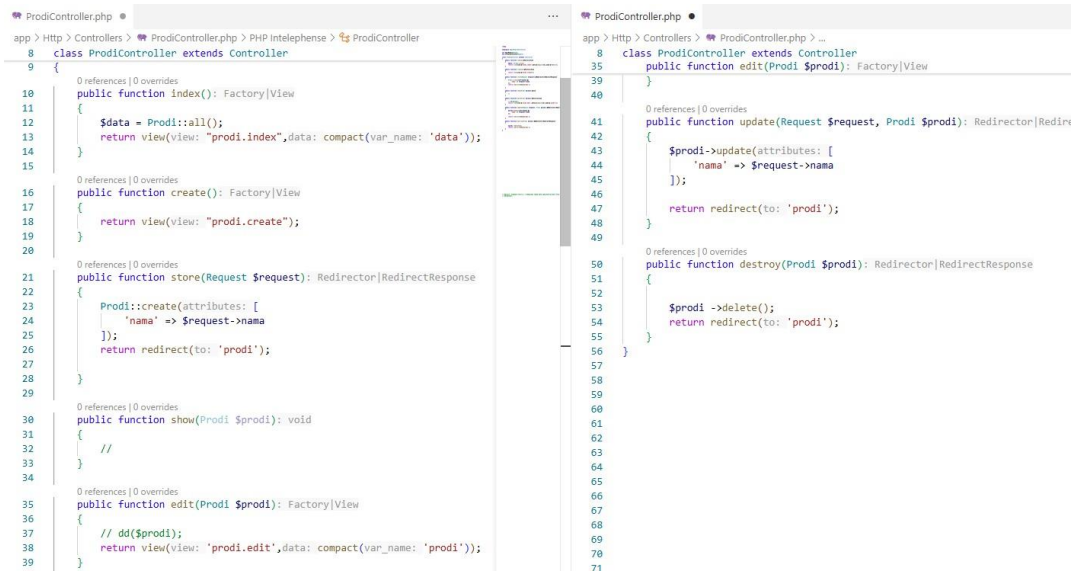
Setelah objek model terbentuk, pengecekan lanjutan dapat dilakukan menggunakan dd($prodi); untuk menampilkan isi lengkap dari objek Prodi yang telah dibentuk sebelum data disimpan ke dalam basis data. Hasil debugging ini menampilkan informasi penting seperti atribut id, nama, serta waktu created_at dan updated_at dalam bagian #attributes, yang menandakan bahwa data telah berhasil terbentuk dan siap diproses. Bagian #original menunjukkan data asli dari database, sedangkan #changes tetap kosong apabila belum ada perubahan sejak pemuatan. Informasi tambahan lainnya seperti koneksi database (mysql), nama tabel (prodis), dan konfigurasi pengisian massal (#fillable dan #guarded) juga turut disertakan. Seluruh proses ini sangat penting dalam tahap pengembangan karena tidak hanya membantu memastikan validitas dan alur data dari form ke controller, tetapi juga memberikan gambaran menyeluruh tentang struktur model yang digunakan, sehingga pengembangan aplikasi dapat berjalan secara terstruktur dan minim kesalahan.

2.3.3 Relasi dan Finalisasi

1. CRUD relasi master-detail.

Relasi master-detail pada Laravel menggambarkan hubungan antar entitas di mana satu entitas berperan sebagai induk dan lainnya sebagai rincian dari entitas tersebut. Dalam konteks ini, entitas Prodi merupakan entitas master yang dapat memiliki banyak entitas Mahasiswa sebagai detailnya. Untuk mengatur keterkaitan ini, digunakan konfigurasi relasi pada model Laravel hasMany pada model Prodi dan belongsTo pada model Mahasiswa. Selain itu, struktur basis data pun harus mencantumkan foreign key pada tabel Mahasiswa yang menunjuk ke entitas Prodi. Pendekatan ini memungkinkan integrasi data yang konsisten dan terstruktur, serta memudahkan dalam pengelolaan antarmuka dan proses CRUD untuk masing-masing entitas yang saling terhubung.

a. Implementasi CRUD pada Prodi (Relasi Master-Detail).

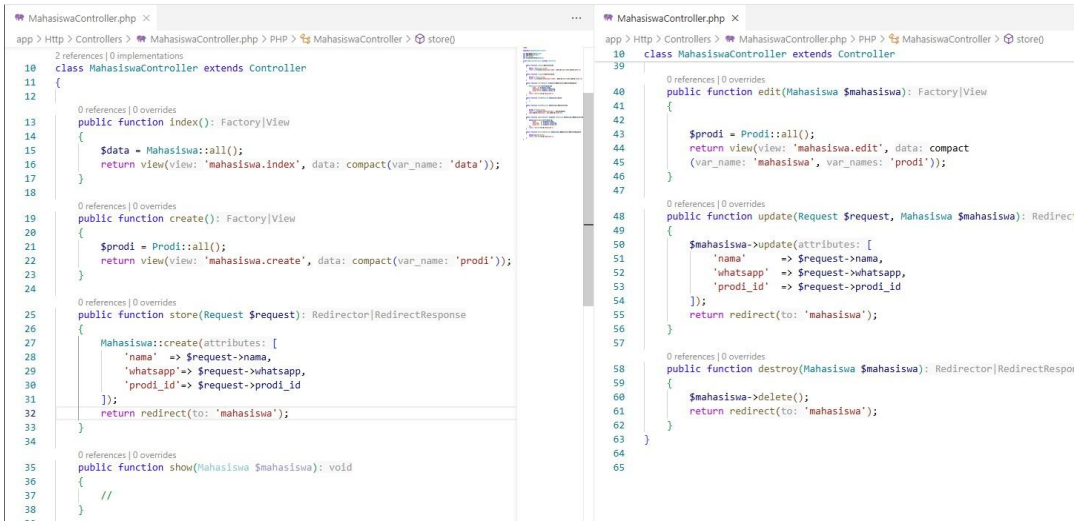


Gambar 2.21 Implementasi relasi CRUD pada Prodi.

Pada gambar 2.21 merupakan proses CRUD pada entitas Prodi diawali melalui method index() pada ProdiController, di mana seluruh data Prodi diambil menggunakan Prodi::all() dan dikirimkan ke tampilan prodi.index dengan compact. Tampilan ini menyajikan data dalam bentuk tabel, lengkap dengan tombol aksi untuk mengedit dan menghapus entri. Method create() akan memunculkan form kosong untuk entri baru, sedangkan data yang dikirim melalui form akan ditangani oleh method store() yang menyimpan data menggunakan perintah Prodi::create(['nama' => $request->nama]);. Laravel secara otomatis mencatat waktu pembuatan dan pembaruan data melalui fitur timestamp. Untuk proses pembaruan, method edit() akan memuat data tertentu berdasarkan ID dan menampilkannya di form yang telah terisi sebelumnya. Setelah diperbarui, method update() akan menyimpan perubahan dengan memanggil $prodi->update([...]). Adapun proses penghapusan dilakukan melalui method destroy() dengan menjalankan $prodi->delete();. Seluruh form dalam proses ini menggunakan proteksi @csrf untuk keamanan, serta @method('PUT') pada form pembaruan sebagai bentuk spoofing metode HTTP karena HTML hanya mendukung GET dan POST. Secara umum, struktur form update dan create hampir identik, yang membedakan hanya pada nilai input yang telah diisi sebelumnya dan penggunaan

method spoofing untuk pembaruan data.

b. Implementasi CRUD pada Mahasiswa (Relasi Master-Detail).



```
MahasiswaController.php
app > Http > Controllers > MahasiswaController.php > PHP > MahasiswaController > store()
2 references | 0 implementations
class MahasiswaController extends Controller
{
    0 references | 0 overrides
    public function index(): Factory|View
    {
        $data = Mahasiswa::all();
        return view(view: 'mahasiswa.index', data: compact(var_name: 'data'));
    }

    0 references | 0 overrides
    public function create(): Factory|View
    {
        $prodi = Prodi::all();
        return view(view: 'mahasiswa.create', data: compact(var_name: 'prodi'));
    }

    0 references | 0 overrides
    public function store(Request $request): Redirector|RedirectResponse
    {
        Mahasiswa::create(attributes: [
            'nama'  => $request->nama,
            'whatsapp'=> $request->whatsapp,
            'prodi_id'=> $request->prodi_id
        ]);
        return redirect(to: 'mahasiswa');
    }

    0 references | 0 overrides
    public function show(Mahasiswa $mahasiswa): void
    {
        //
    }
```

```
MahasiswaController.php
app > Http > Controllers > MahasiswaController.php > PHP > MahasiswaController > store()
class MahasiswaController extends Controller

    0 references | 0 overrides
    public function edit(Mahasiswa $mahasiswa): Factory|View
    {

        $prodi = Prodi::all();
        return view(view: 'mahasiswa.edit', data: compact
        (var_name: 'mahasiswa', var_names: 'prodi'));
    }

    0 references | 0 overrides
    public function update(Request $request, Mahasiswa $mahasiswa): Redirec
    {
        $mahasiswa->update(attributes: [
            'nama'     => $request->nama,
            'whatsapp'  => $request->whatsapp,
            'prodi_id'  => $request->prodi_id
        ]);
        return redirect(to: 'mahasiswa');
    }

    0 references | 0 overrides
    public function destroy(Mahasiswa $mahasiswa): Redirector|RedirectRespo
    {
        $mahasiswa->delete();
        return redirect(to: 'mahasiswa');
    }
}
```

Gambar 2.22 Implementasi CRUD pada Mahasiswa.

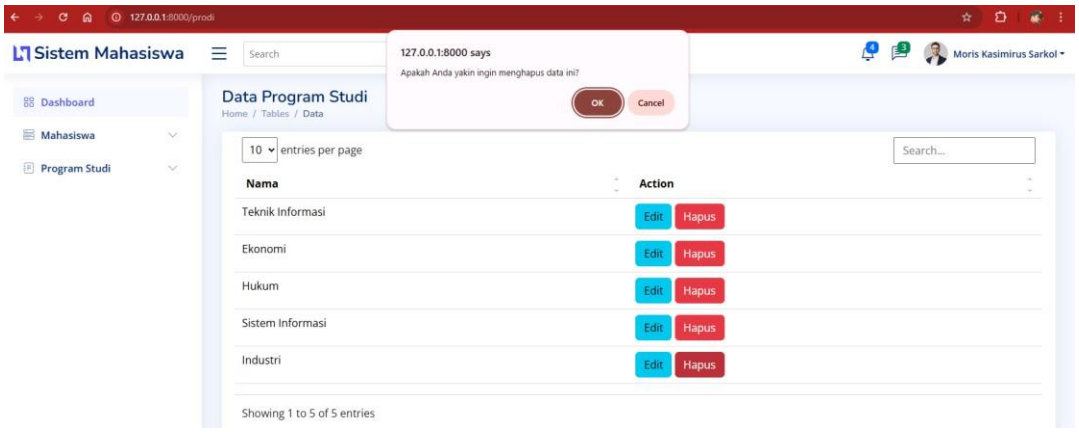
Pada gambar 2.22 ditampilkan implementasi lengkap fitur CRUD (Create, Read, Update, Delete) pada entitas Mahasiswa memiliki keterkaitan langsung dengan Prodi melalui atribut prodi_id. Pada proses pembuatan data baru, method create() dalam MahasiswaController memanggil seluruh data Prodi dengan Prodi::all() dan mengirimkannya ke tampilan mahasiswa.create, yang menyajikan program studi dalam bentuk dropdown. Saat pengguna mengisi data mahasiswa dan memilih program studi, data tersebut akan dikirim dan diproses oleh method store() untuk disimpan ke dalam basis data menggunakan perintah Mahasiswa::create([...]), termasuk nama, nomor WhatsApp, dan ID Prodi. Untuk pembaruan data, method edit() akan memuat data mahasiswa dan seluruh daftar Prodi ke tampilan mahasiswa.edit. Setelah diperbarui, method update() menyimpan perubahan menggunakan $mahasiswa->update([...]). Karena model Mahasiswa telah dikonfigurasi menggunakan belongsTo, maka data nama program studi yang terkait dapat langsung ditampilkan pada tampilan index.blade.php menggunakan sintaks $row->prodi->nama. Proses penghapusan mahasiswa dilakukan oleh method destroy(), dan data yang terhapus tidak lagi ditampilkan pada daftar. Dengan struktur ini, seluruh proses CRUD pada Mahasiswa telah memanfaatkan relasi master-detail secara optimal dan sesuai standar Laravel.

2. Implementasi fitur tambahan.

Sebagai bagian dari peningkatan fungsionalitas serta kenyamanan dalam penggunaan sistem, telah diterapkan fitur tambahan berupa sistem peringatan (alert) sebelum proses penghapusan data dijalankan. Penambahan fitur ini dimaksudkan untuk mencegah terjadinya penghapusan data secara tidak sengaja dan memberikan kendali yang lebih besar kepada pengguna dalam mengelola data yang tersimpan.

a. Penambahan fitur alert.

```
<a href= "{{ route(name: 'prodi.edit', parameters: $row->id) }}" class="btn btn-info">Edit</a>
<button type="submit" class="btn btn-danger" onclick="return confirm('Apakah Anda yakin ingin menghapus data ini?')">Hapus</button>
</form>
</td>
```



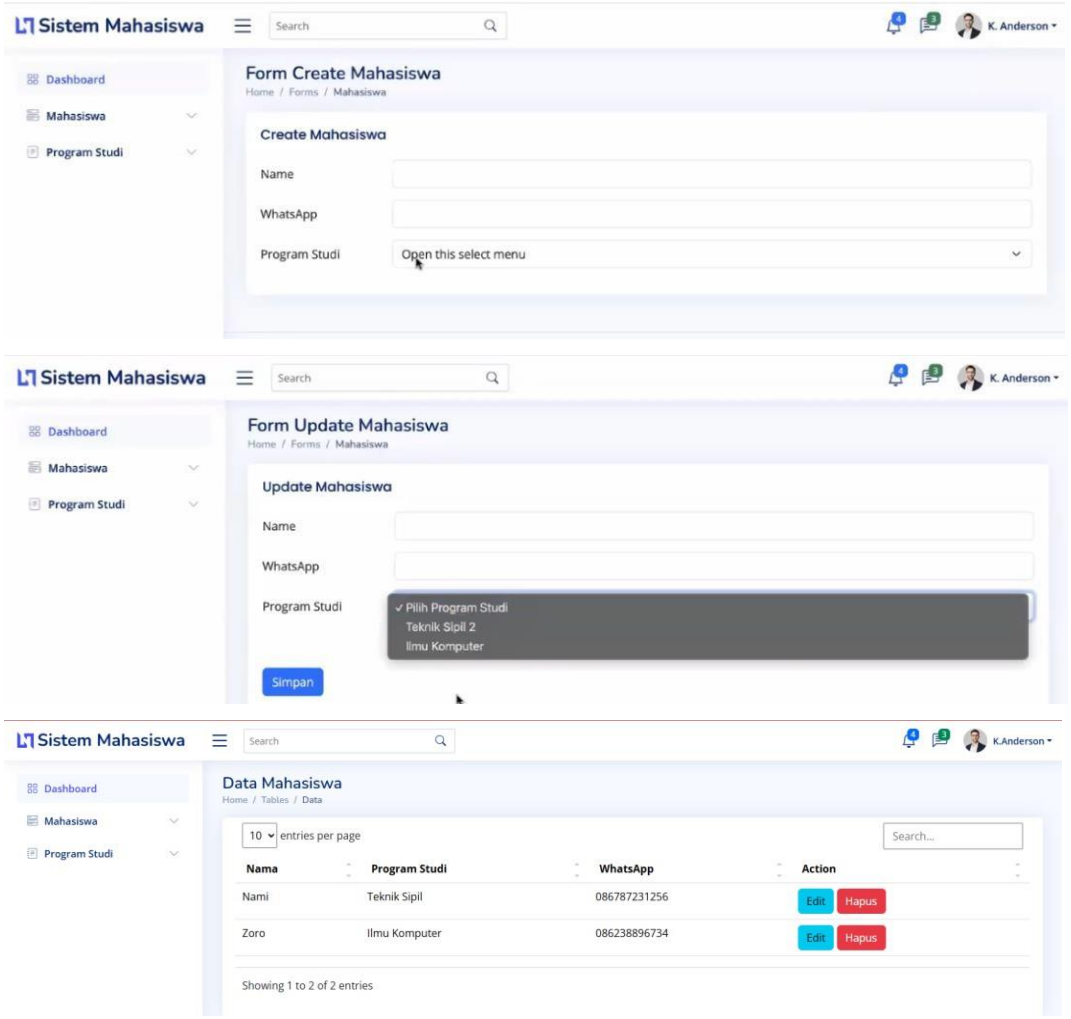
Gambar 2.23 Penambahan fitur alert

Pada gambar 2.23 ditampilkan implementasi tombol hapus yang telah dilengkapi dengan mekanisme konfirmasi menggunakan JavaScript. Kode yang digunakan adalah <button type="submit" class="btn btn-danger" onclick="return confirm('Apakah Anda yakin ingin menghapus data ini?')">Hapus</button>, di mana fungsi confirm() akan memunculkan jendela dialog berisi pesan “Apakah Anda yakin ingin menghapus data ini?”. Apabila pengguna menekan tombol "Ok", maka nilai yang dikembalikan adalah true sehingga form akan dikirimkan ke route prodi.destroy untuk diproses oleh controller. Sebaliknya, apabila tombol "Cancel" dipilih, maka nilai false akan dikembalikan dan pengiriman form dibatalkan. Dengan demikian, sistem ini berfungsi sebagai lapisan verifikasi tambahan guna memastikan bahwa proses penghapusan dilakukan secara sadar dan disengaja oleh pengguna.

3. Uji coba aplikasi.

Uji coba sistem dilakukan untuk memastikan bahwa seluruh fitur dasar telah berfungsi sesuai dengan kebutuhan serta antarmuka dapat digunakan secara intuitif oleh pengguna. Pengujian difokuskan pada dua entitas utama, yaitu Prodi sebagai entitas induk dan Mahasiswa sebagai entitas detail yang memiliki keterkaitan langsung melalui relasi basis data. Hasil uji coba menunjukkan bahwa interaksi pengguna dengan sistem berjalan lancar, dan setiap fungsi CRUD dapat dijalankan tanpa kendala berarti.

a. Hasil uji coba Mahasiswa.

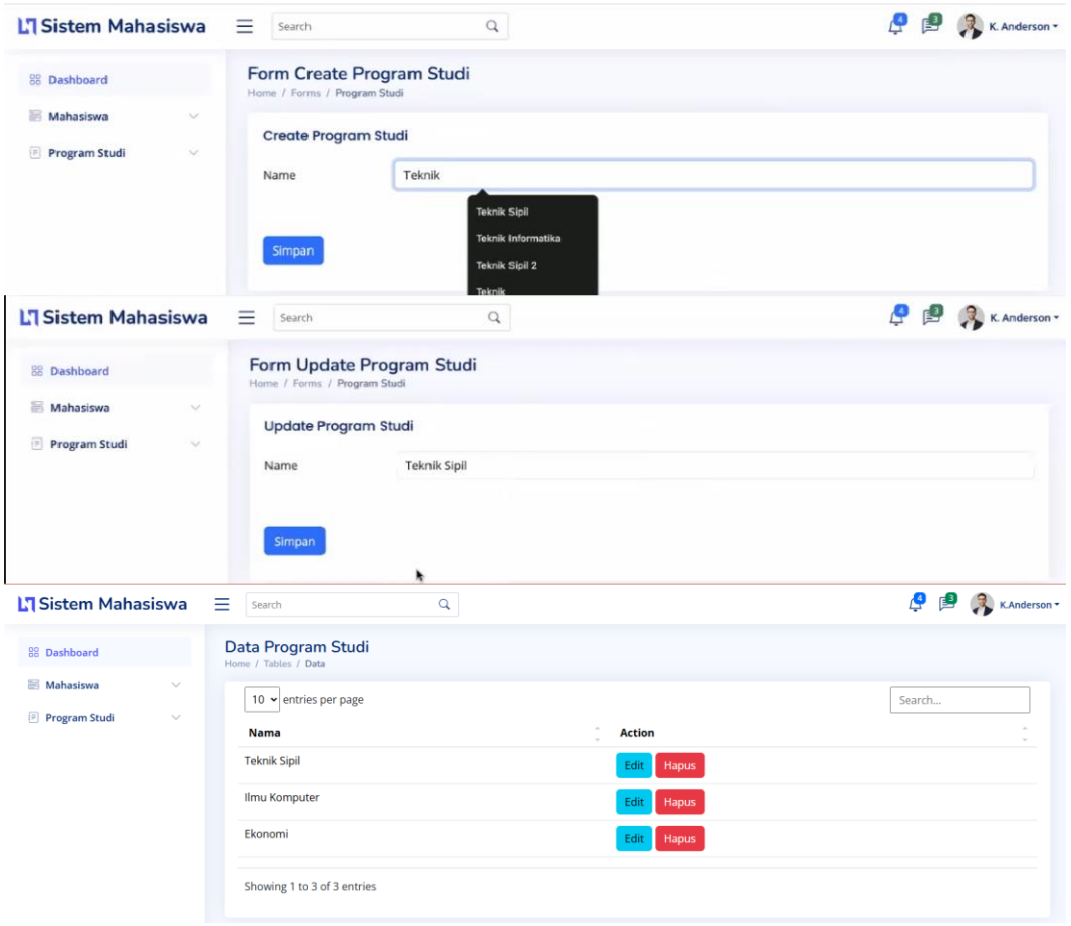


Gambar 2.24 Hasil uji mahasiswa.

Pada gambar 2.24 memperlihatkan hasil uji coba fitur Mahasiswa yang mencakup proses input data, pemilihan program studi, dan tampilan data yang telah disimpan.

Pada bagian atas, ditampilkan form untuk menambahkan data mahasiswa baru, terdiri atas kolom nama, nomor WhatsApp, dan dropdown untuk memilih program studi. Dropdown ini terhubung langsung dengan data dari entitas Prodi, sehingga pilihan program studi yang tersedia bersumber dari data yang sudah ada. Setelah data disimpan, bagian bawah gambar menampilkan daftar mahasiswa dalam bentuk tabel. Setiap baris menampilkan nama mahasiswa, program studi yang dipilih, serta nomor WhatsApp. Selain itu, tersedia tombol Edit dan Hapus untuk memudahkan pengelolaan data. Penampilan nama program studi menunjukkan bahwa relasi antar entitas telah berhasil diterapkan. Tampilan ini menandakan bahwa sistem telah mampu menjalankan seluruh fungsi CRUD pada Mahasiswa secara utuh dan sesuai dengan relasi yang dibangun sebelumnya.

b. Hasil uji coba Prodi.



Gambar 2.25 Hasil uji Prodi.

Pada gambar 2.25 menampilkan hasil uji coba fitur Program Studi yang mencakup

proses pembuatan, pembaruan, dan penampilan data. Pada bagian pertama, terlihat form untuk membuat data program studi baru, yang terdiri atas satu kolom input nama. Setelah data dimasukkan, pengguna dapat menekan tombol Simpan untuk mengirimkan data tersebut ke server. Bagian kedua menunjukkan form pembaruan data. Form ini secara otomatis menampilkan nama program studi yang sudah ada, sehingga pengguna dapat langsung mengedit nilai yang diperlukan tanpa harus mengisi ulang seluruh data. Setelah perubahan dilakukan, tombol Simpan akan memperbarui data yang telah ada. Pada bagian akhir ditampilkan daftar seluruh program studi yang telah tersimpan dalam bentuk tabel. Setiap entri memiliki dua tombol aksi, yaitu Edit dan Hapus, yang memungkinkan pengguna untuk mengelola data secara langsung dari tampilan utama. Penataan data dilakukan secara rapi dan responsif, serta interaksi pengguna difasilitasi secara optimal melalui antarmuka yang sederhana namun fungsional.

2.4 Hasil Uji Kompetensi

Berdasarkan rangkaian evaluasi yang telah dilaksanakan, dapat disimpulkan bahwa peserta menunjukkan pemahaman yang baik terhadap materi serta penguasaan perintah teknis dalam pengembangan aplikasi web. Evaluasi disusun secara sistematis dan terbagi ke dalam beberapa tahapan, yang masing-masing dirancang untuk mengukur kemampuan teoritis sekaligus keterampilan teknis peserta dalam membangun aplikasi berbasis Laravel.

a. Hasil dan nilai ujikom Teori.

Gambar 2.1 Hasil Pre-Test, Pre-Test, dan Ujikom Teori.

Pada gambar 2.26 merupakan proses evaluasi dilaksanakan melalui tiga tahapan

utama, yaitu pre-test, post-test, dan ujian kompetensi lapangan (ujikom). Pada tahap pre-test, peserta mengerjakan 50 soal pilihan ganda dalam waktu 30 menit dengan skor yang diperoleh sebesar 82 dari 100. Hasil ini menunjukkan penguasaan awal terhadap materi dasar seperti framework CSS, pengelolaan proyek, serta pemahaman awal tentang migrasi teknologi. Setelah pelatihan intensif dilaksanakan, peserta mengikuti post-test dan memperoleh skor 92 dari 100. Skor tersebut menunjukkan peningkatan signifikan pada penguasaan aspek teknis, meliputi penggunaan SQL, pemahaman routing, dan penerapan prinsip pemrograman berbasis real-time. Tahap akhir berupa ujian kompetensi lapangan diselenggarakan secara tertutup dengan pengawasan dari pihak penguji. Peserta diminta menyelesaikan simulasi proyek aplikasi nyata, dan berhasil meraih skor 94 dari 100. Dalam pengujian ini, aspek yang dinilai meliputi keberhasilan pembuatan fitur CRUD, konektivitas aplikasi dengan database MySQL maupun PostgreSQL, serta pemahaman perbedaan antara keduanya. Selain itu, peserta diuji dalam penggunaan fitur keamanan seperti penyisipan token @csrf, serta teknik debugging ketika terjadi error. Peserta diharapkan mampu menjelaskan serta menerapkan langkah-langkah penanganan kesalahan secara sistematis, mulai dari membaca pesan kesalahan, mengecek rute dan controller, hingga mengamankan form input.

b. Sertifikat BNSP Web Developer.

13459132

BADAN NASIONAL
SERTIFIKASI PROFESI
INDONESIAN PROFESSIONAL CERTIFICATION AUTHORITY

SERTIFIKAT KOMPETENSI
CERTIFICATE OF COMPETENCE

No. 62019 3514 0 0002281 2024

Dengan ini menyatakan bahwa,
This is to certify that,

Moris Kasimirus Sarkol

No. Reg. TIK 2324 03351 2024

Telah kompeten pada bidang:
Is competent in the are of:

WEB DEVELOPER
Web Developer

Dengan kualifikasi / kompetensi:
With qualification / competency:

WEB DEVELOPER
Web Developer

Sertifikat ini berlaku untuk : 3 (tiga) tahun
This certificate is valid for : 3 (three) years

Cirebon, 19 Desember 2024

Atas nama Badan Nasional Sertifikasi Profesi (BNSP)
On Behalf of Indonesian Professional Certification Authority

Lembaga Sertifikasi Profesi Digital Teknologi Informasi Indonesia
Professional Certification Body for Indonesian Information Technology Digital

Askarno, S.E., M.M
Direktur
Director

Daftar Unit Kompetensi

List of Unit(s) of Competency

No.	Kode Unit Kompetensi *Code of Competency Unit*	Judul Unit Kompetensi *Title of Competency Unit*
1	J.620100.041.01	Melaksanakan cutover aplikasi *Implementing Application Cutover*
2	J.620100.045.01	Melakukan pemantauan resource yang digunakan aplikasi *Monitoring Application Resource Usage*
3	J.620100.025.02	Melakukan debugging *Debugging*
4	J.620100.038.01	Melaksanakan pengujian oleh pengguna (UAT) *Conducting Test by user (UAT)*
5	J.62090.018.01	Mengelola risiko keamanan Informasi *Managing Information security risks*
6	J.620100.020.02	Menggunakan SQL *Using SQL*
7	J.620100.044.01	Menerapkan alert notification jika aplikasi bermasalah *Implementing an alert notification if the application has a problem*
8	J.620100.003.01	Melakukan identifikasi library, komponen atau framework yang diperlukan *Identifying Required Libraries, Components, or Frameworks*
9	J.620100.024.02	Melakukan migrasi ke teknologi baru *Migrating to new technology*
10	J.620100.047.01	Melakukan pembaharuan perangkat lunak *Updating software*
11	J.620100.039.02	Memberikan petunjuk teknis kepada pelanggan *Providing technical instructions to customers*
12	J.620100.030.02	Menerapkan pemrograman multimedia *Implementing multimedia programming*
13	TIK.SM03.001.01	Menentukan arsitektur perangkat keras *Determining Hardware Architecture*
14	M.702090.001.0	Mengelola proyek secara terintegrasi *Project integration management*
15	J.620100.001.01	Menganalisis tools *Analyzing tools*
16	J.620100.002.01	Menganalisis skalabilitas perangkat lunak *Analyzing software scalability*
17	J.620100.043.01	Menganalisis dampak perubahan terhadap aplikasi *Analyzing the impact of changes to applications*
18	J.620100.029.02	Menerapkan pemrograman paralel *Implementing parallel programming*
19	M.702090.005.0	Mengelola kualitas proyek *Project quality management*
20	J.620100.022.02	Mengimplement asikan algoritma pemrograman *Programming algorithms Implementation*
21	J.620100.028.02	Menerapkan pemrograman real time *Implementing real-time programming*
22	M.702090.002.0	Mengelola ruang lingkup proyek *Project scope management*

Cirebon, 19 Desember 2024

Lembaga Sertifikasi Profesi Digital Teknologi Informasi Indonesia

Indonesian Information Technology Digital Professional Certification Institute

Moris Kasimirus Sarkol
Tanda tangan pemilik
Signature of holder

Ikhsan Nendi, S.E., M.M
Tanda Tangan Manajer Sertifikasi
Signature Manager of Certification

BNSP

Gambar 2.27 Sertifikat BNSP Web Developer

Pada gambar 2.27 memperlihatkan bukti sebagai hasil akhir dari seluruh rangkaian pelatihan dan evaluasi, peserta dinyatakan kompeten dan berhasil memperoleh

sertifikat resmi dari Badan Nasional Sertifikasi Profesi (BNSP) dengan skema Junior Web Developer. Sertifikasi ini diterbitkan oleh Lembaga Sertifikasi Profesi (LSP) yang terlisensi BNSP dan menjadi bukti bahwa peserta telah memenuhi standar kompetensi kerja nasional di bidang pengembangan aplikasi web. Di dalam sertifikat tersebut tercantum unit-unit kompetensi sesuai dengan Kerangka Kualifikasi Nasional Indonesia (KKNI), termasuk di antaranya pemrograman web, pengelolaan basis data, penggunaan framework seperti Laravel, serta penerapan prinsip keamanan aplikasi. Sertifikat ini berlaku secara nasional selama tiga tahun dan dapat digunakan sebagai legitimasi kompetensi dalam dunia kerja profesional maupun proses seleksi kerja yang memerlukan bukti keahlian resmi.

2.5 Manfaat

Pelaksanaan program sertifikasi kompetensi Web Developer memberikan manfaat nyata dalam meningkatkan keterampilan teknis serta pemahaman konseptual peserta di bidang pengembangan aplikasi web. Materi yang diberikan mencakup instalasi Laravel, konfigurasi proyek, pembuatan fitur CRUD dengan relasi master-detail, serta penyusunan antarmuka responsif menggunakan Bootstrap. Selain itu, peserta dibekali pemahaman terhadap mekanisme keamanan seperti penggunaan token @csrf, proses debugging saat terjadi error, dan pemetaan relasi basis data. Evaluasi bertahap melalui pre-test, post-test, dan ujikom memberikan ruang untuk mengukur perkembangan dan kesiapan peserta dalam menghadapi situasi pengembangan aplikasi secara nyata. Penerapan langsung pada simulasi proyek membuat peserta terbiasa dengan alur kerja profesional, serta meningkatkan kepercayaan diri dan kesiapan memasuki industri digital.