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 *create, read, update, delete* (CRUD)	- Membuat *create, read, update, delete* (CRUD) sederhana - Relasi antar entitas database - Uji coba dan *debungging* dasar.	Aplikasi *create, read, update, delete* (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.

Pada tabel 2.1 merupakan penjelasan tentang silabus Pelatihan terkait pengembangan aplikasi menggunakan *Laravel* dan *Bootstrap*. Dimulai dengan

dasar-dasar dan persiapan, akan diperkenalkan pada *Laravel* dan *Bootstrap*, serta diajarkan cara menginstal *tools* dan *slicing template UI*. Selanjutnya, peserta belajar membuat aplikasi *create, read, update, delete* (CRUD) sederhana dengan relasi antar entitas database, serta uji coba dan *debugging* dasar. Terakhir, pada topik finalisasi, akan mempelajari cara membuat *create, read, update, delete* CRUD dengan menambahkan fitur tambahan, dan melakukan uji coba aplikasi hingga menghasilkan aplikasi lengkap dengan fitur 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 Management*)	2
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*	

Pada tabel 2.2 tersebut merupakan daftar unit kompetensi atau serangkaian tugas dalam sebuah proyek teknologi informasi (TI). Tugas-tugas ini mencakup berbagai aspek, mulai dari manajemen proyek (seperti mengelola kualitas, ruang lingkup, dan integrasi), hingga kegiatan teknis seperti melakukan *debugging*, pengujian dengan pengguna (UAT), dan memantau sumber daya aplikasi. Beberapa dari tugas tersebut juga sudah ditentukan alokasi waktunya dalam jam untuk pelaksanaannya.

2.2 Proses Pendaftaran

Proses pendaftaran dilakukan secara online dengan tahapan sebagai berikut:

1. Peserta dapat menghubungi 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 formulir yang tersedia.

3. Setelah mengisi formulir, 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 Waktu pelaksanaan

Masa pelatihan pada LSP Ditekindo sekitar 3 hari untuk masa pelatihan dan 1 hari untuk uji kompetensi, meskipun ini bisa bervariasi tergantung pada skema dan tingkat kesulitan uji kompetensi yang diambil, dan lamanya sangat bergantung pada LSP yang menyelenggarakan serta bidang kompetensi yang diuji.

2.4 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*.

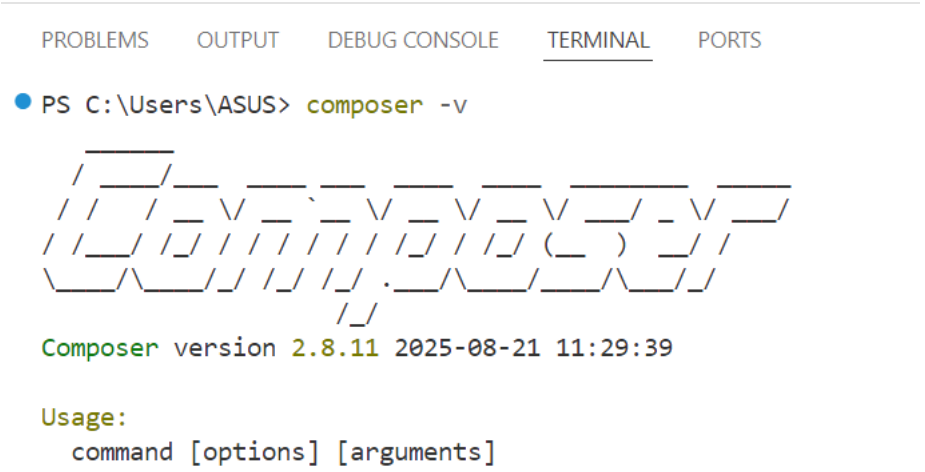
Setelah proses instalasi selesai, peserta diwajibkan untuk menyampaikan bukti instalasi berupa versi masing-masing alat yang telah berhasil diinstal.

a. Instalasi *XAMPP*.

Gambar 2. 1 Hasil Instalasi *XAMPP*

Pada gambar 2.3 merupakan tampilan *dashboard* hasil instalasi *XAMPP* beserta versinya.

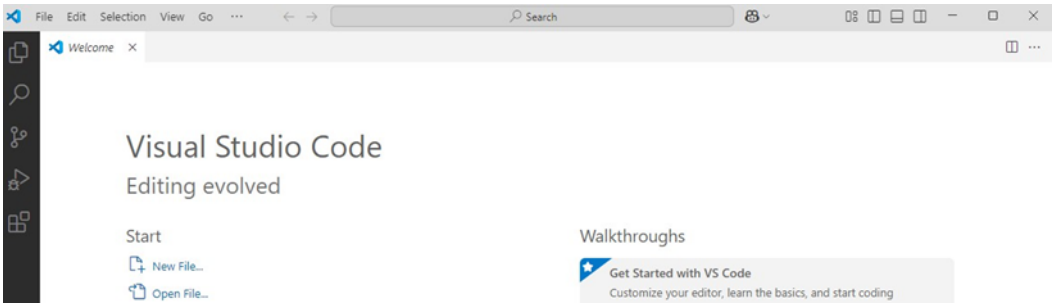
b. Instalasi *Composer*



Gambar 2. 2 Hasil Instalasi *Composer*

Pada gambar 2.4 merupakan hasil pengecekan *Composer* berserta versinya berhasil terinstal menggunakan perintah "*composer -v*" pada terminal *Visual Studio Code.*

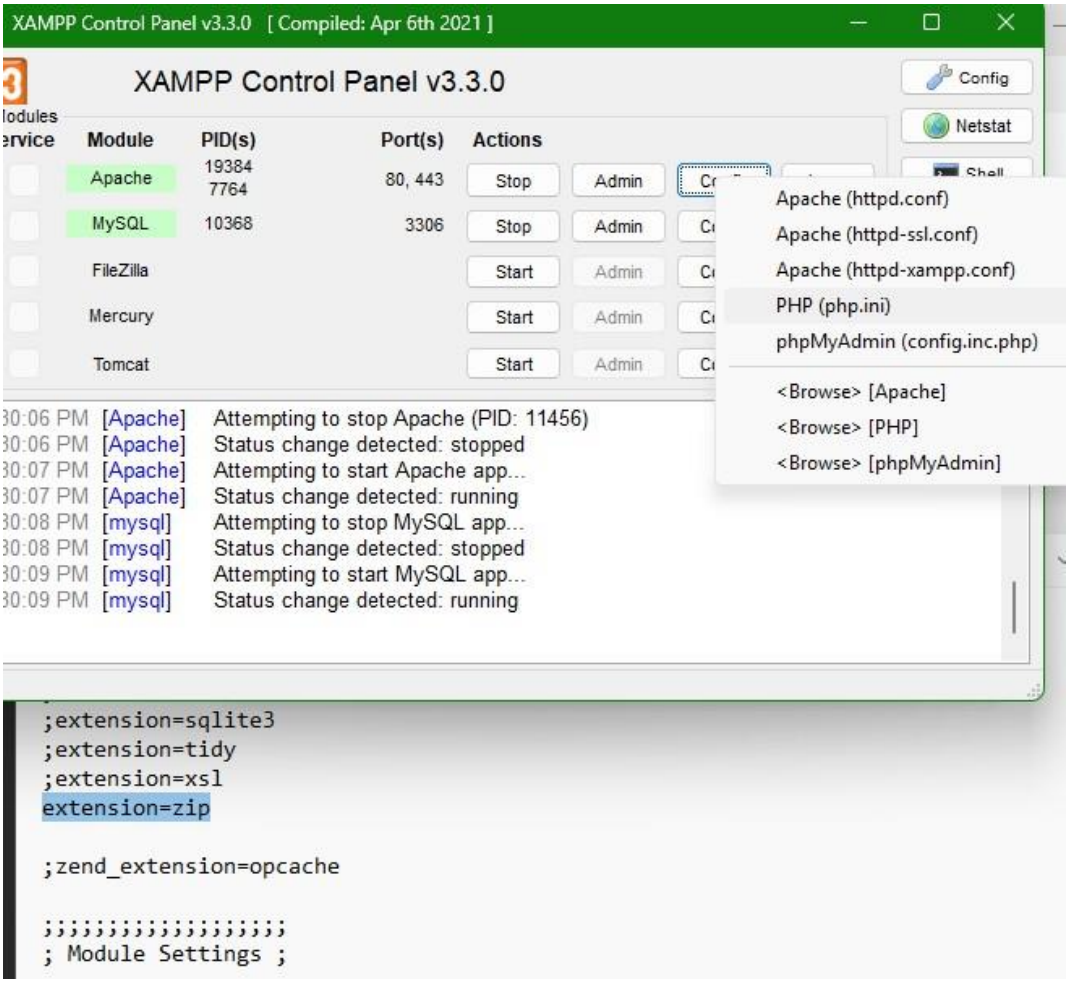
c.* Instalasi *Visual Studio Code



Gambar 2. 3 Hasil Instalasi VS Code

Pada gambar 2.5 merupakan *Visual Studio Code* berhasil terinstal dan dijalankan.

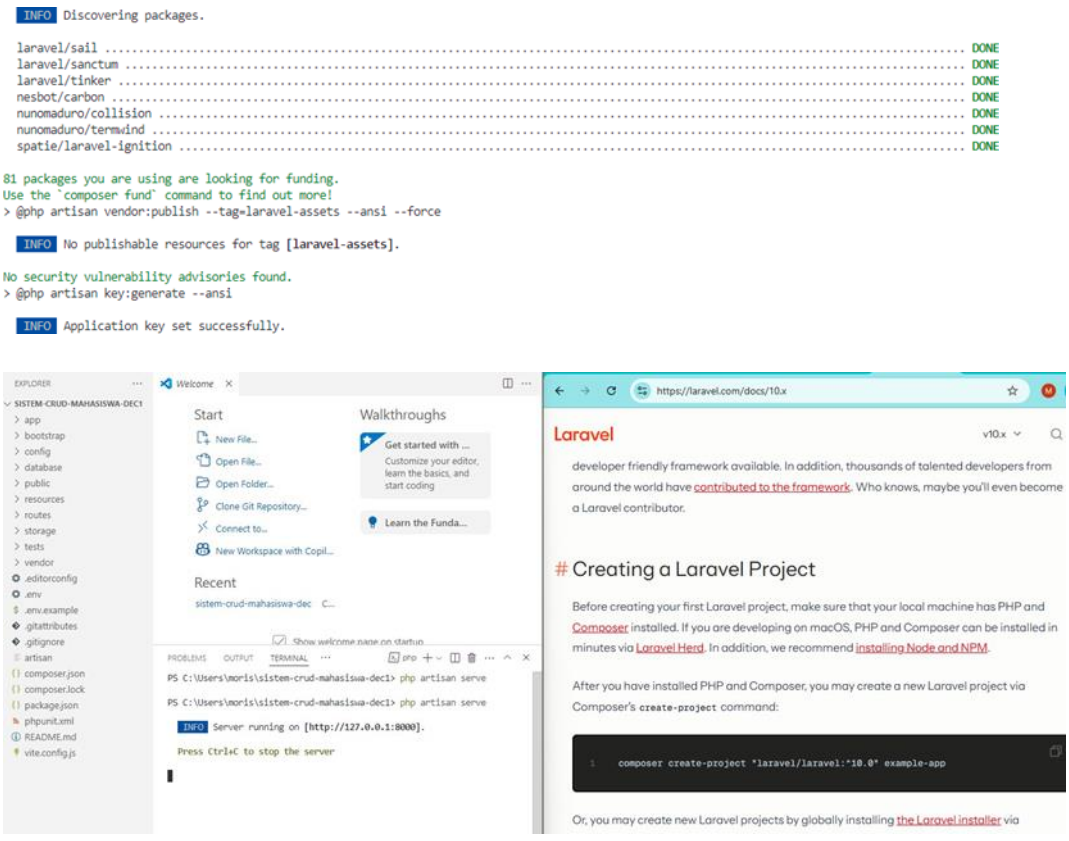
d. Menjalankan *XAMPP.*



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.

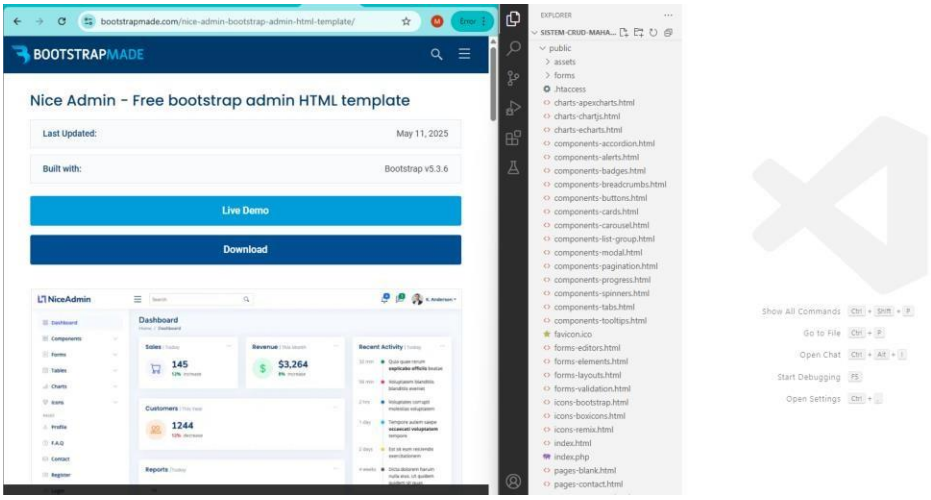


Gambar 2. 5 Hasil Installasi *Laravel* dan Akses Server *Local.*

Pada gambar 2.5 memperlihatkan tahapan installasi 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. Installasi Boostrap



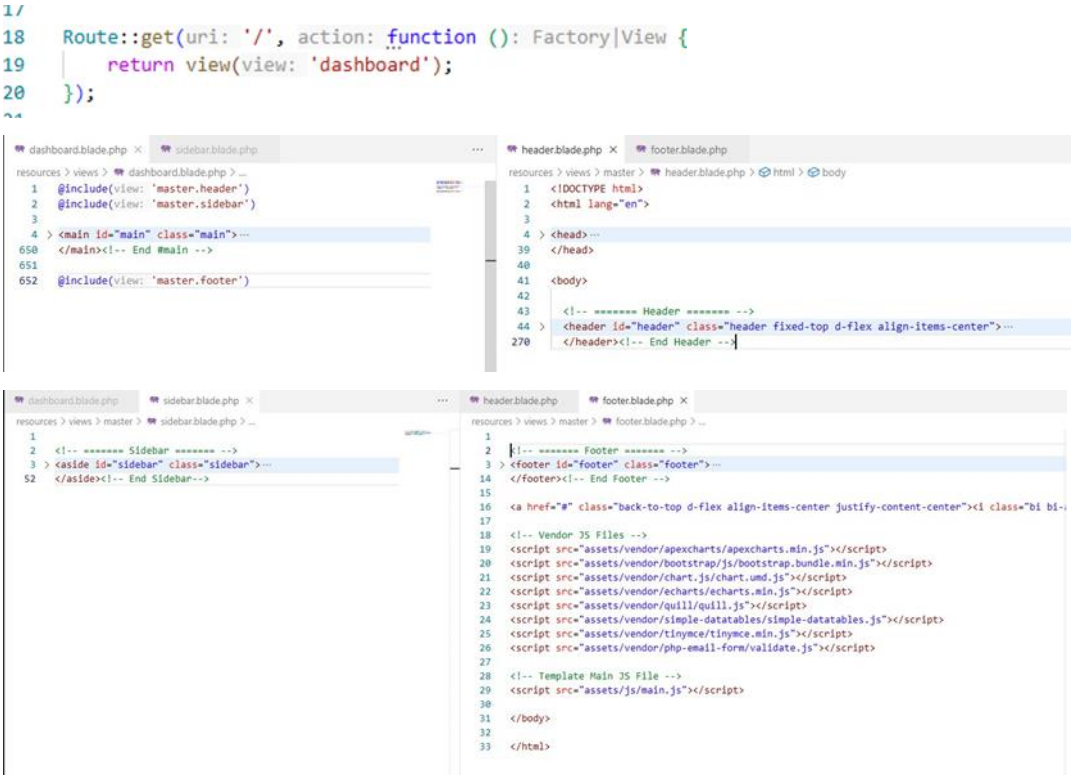
Gambar 2. 6 Hasil *Instalasi Boostrap*.

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*

code 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 mudah, 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 *student* dan konfigurasi *routing.*

```
<section class="section">
 <div class="row">
  <div class="col-lg-8">

   <div class="card">
    <div class="card-body">
     <h5 class="card-title">Form Tambah Buku</h5>

     <form action="{{ route('students.store') }}" method="POST">
     @csrf

     <div class="mb-3">
     <label for="name" class="form-label">Nama</label>
     <input type="text" class="form-control" id="name" name="name" value="{{ old('name') }}" requir
     @error('name') <div class="text-danger">{{ $message}}</div> @enderror
     </div>

<div class="mb-3">
 <label for="email" class="form-label">Email</label>
 <input type="email" class="form-control" id="email" name="email" value="{{ old('email') }}" requir
 @error('email') <div class="text-danger">{{ $message }}</div> @enderror
 </div>
```

index.blade.php × | edit.blade.php | web.php | sidebar.blade.php | StudentApiController.php

niceadmin-Laravel > resources > views > students > index.blade.php > section.section > div.row > div.col-lg-12 >

```
<section class="section">
<div class="row">
<div class="col-lg-12">
<div class="card">
 <div class="card-body">

<a href="{{ route('students.create') }}" class="btn btn-primary mb-3">Tambah Student</a>

<table class="table table-striped">
    <thead>
```

Gambar 2. 8 Pembuatan halaman *student* , 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 *student* 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*

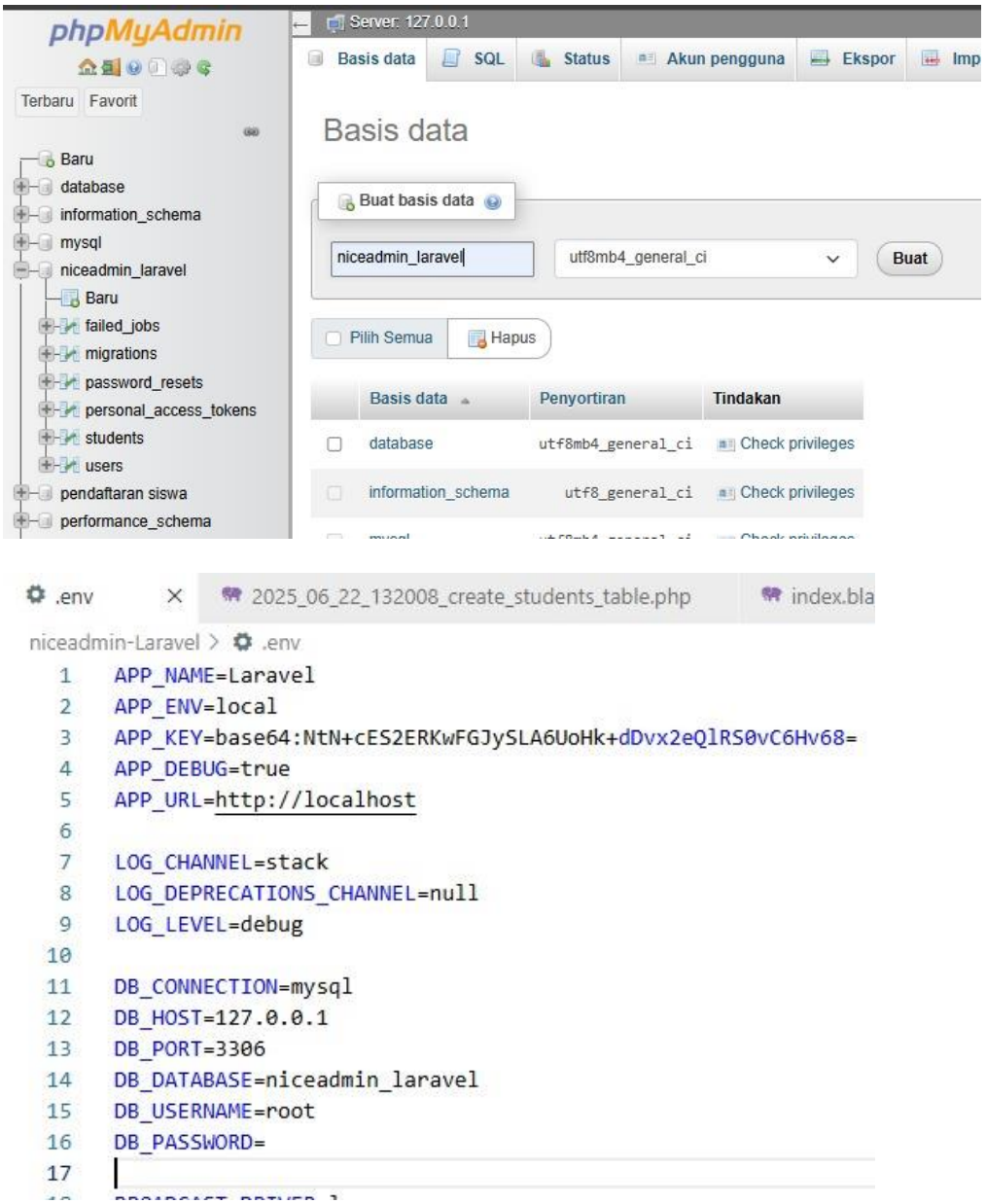
digunakan untuk menampilkan daftar data dalam bentuk tabel, sedangkan *create.blade.php* memuat elemen formulir yang digunakan untuk menambahkan data baru. Selain itu, konfigurasi *routing* juga perlu dilakukan. Masing-masing tampilan diberi rute khusu yang menghubungkan alamat *URL* dengan file tampilan. Contohnya, rute *create-student* akan diarahkan ke tampilan *student.create*, dan rute /student diarahkan ke tampilan *student.index*. Pola yang sama diterapkan pula untuk entitas daftar *student*. 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 *create, read, update, delete* (CRUD).

2.3.2 Dasar *Laravel* dan *CRUD*

1. Membuat *CRUD* sederhana

Sebelum memulai proses pembuatan fitur *create, read, update, delete* (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 *niceadmin_laravel*, dan digunakan untuk menyimpan serta mengelola data entitas program *student* secara terstruktur. Namun, sebagai bentuk contoh dalam penerapan fitur *CRUD*, hanya entitas *student* yang ditampilkan dan dijadikan fokus utama pada tahap pembahasan ini.

a. Membuat database dan konfigurasi file *.env*.

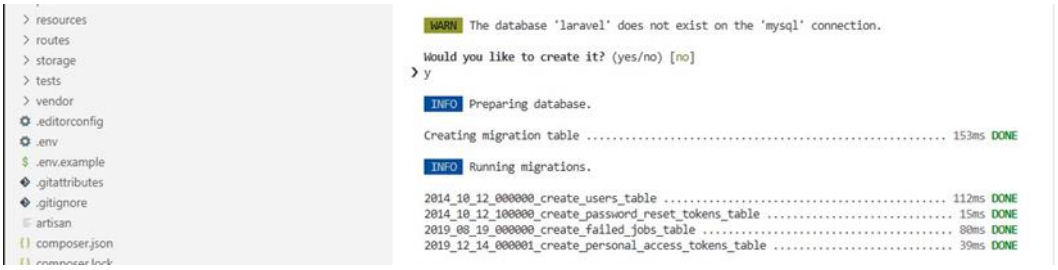


Gambar 2. 9 Membuat database dan Konfigurasi file *.env*.

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 *niceadmin_laravel.* 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*) *niceadmin_laravel*. 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

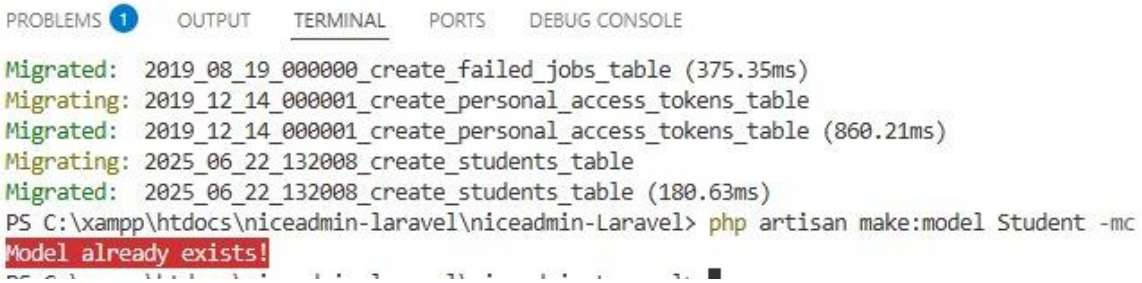


Gambar 2. 10 Hasil migrasi database.

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 *student*



Gambar 2. 11 Hasil membuat entitas berserta tabel.

Pada gambar 2.11 merupakan hasil pembuatan entitas *student* secara otomatis tanpa harus membuatya langsung melalui *MySQL*. Proses ini dilakukan dengan menjalankan *command* (perintah) *php artisan make:model Student -mc* melalui terminal. Perintah tersebut merupakan satu paket dengan beberapa *flag*, yaitu -c untuk membuat file *controller*, 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 Student.php*, file *controller StudentController.php*, dan file migrasi untuk tabel *student*. Proses serupa juga diterapkan untuk entitas *student*, dengan menjalankan perintah yang sama namun mengganti nama entitas menjadi "*Student*". Pendekatan ini mempermudah proses pengembangan karena seluruh komponen dasar telah disiapkan dalam satu langkah terintegrasi.

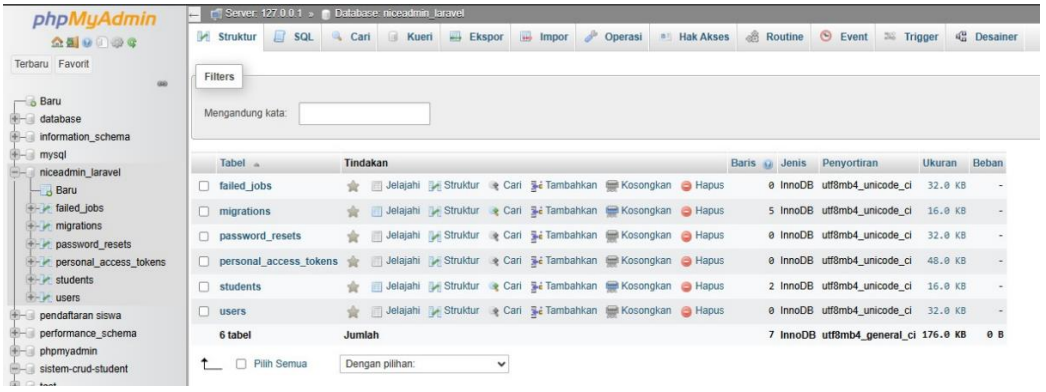
d. Migrasi tabel *student.*

niceadmin-Laravel > database > migrations > 2025_06_22_132008_create_students_table.php > CreateStudent

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;
use PhpParser\Node\NullableType;
use Ramsey\Uuid\Type\Integer;
use SebastianBergmann\Type\NullType;

class CreateStudentsTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('students', function (Blueprint $table) {
            $table->id();
            $table->string ('name');
            $table->string ('email')->uniqid();
            $table->string ('phone')->nullable();
            $table->timestamps();
        });
```



Gambar 2. 12 Hasil migrasi tabel *student*.

Pada gambar 2.12 merupakan proses dan hasil migrasi tabel *student* yang secara otomatis dihasilakan sebelumnya. Pada file tabel *student* 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 student memiliki kolom nama, 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_06_22_132008_create_students_table.php* dan membuat tabel *student* 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. Penyederhanaan *routing*.

```
7   Route::get('/', function () {
8   return view('welcome');
9   });
10
11  Route::get('/admin/dashboard', [AdminController::class, 'dashboard']);
12  Route::resource('students', StudentController::class);
```

```
PS C:\xampp\htdocs\niceadmin-laravel\niceadmin-Laravel> php artisan route:list
+--------+-----------+-------------------------+-----------------+-----------------------------------------------------------------------------------------
------------------------+
| Domain | Method    | URI                     | Name            | Action
                       |
+--------+-----------+-------------------------+-----------------+-----------------------------------------------------------------------------------------
------------------------+
|        | GET|HEAD  | /                       |                 | Closure
                       |
|        | GET|HEAD  | admin/dashboard         |                 | App\Http\Controllers\AdminController@dashboard
                       |
|        | GET|HEAD  | api/user                |                 | Closure
                       |
|        |           |                         |                 |
are\Authenticate:sanctum |
|        | GET|HEAD  | sanctum/csrf-cookie     |                 | Laravel\Sanctum\Http\Controllers\CsrfCookieController@show
                       |
|        | GET|HEAD  | students                | students.index  | App\Http\Controllers\StudentController@index
                       |
|        | POST      | students                | students.store  | App\Http\Controllers\StudentController@store
                       |
|        | GET|HEAD  | students/create         | students.create | App\Http\Controllers\StudentController@create
                       |
|        | GET|HEAD  | students/{student}      | students.show   | App\Http\Controllers\StudentController@show
                       |
|        | PUT|PATCH | students/{student}      | students.update | App\Http\Controllers\StudentController@update
                       |
|        | DELETE    | students/{student}      | students.destroy | App\Http\Controllers\StudentController@destroy
                       |
|        | GET|HEAD  | students/{student}/edit | students.edit   | App\Http\Controllers\StudentController@edit
                       |
```

Gambar 2. 13 Penyederhanaan konfigurasi *routing*.

Pada gambar 2.14 merupakan penyederhanaan konfigurasi *routing* menggunakan fitur *Route::resource*() dalam *Laravel*. Sebelumnya, setiap rute untuk operasi *create, read, update, delete* (CRUD) perlu ditulis secara manual, seperti rute untuk menampilkan *form create*, menampilkan data, dan sebagainya. Penggunaan *Route::resource('student', StudentController::class);* seluruh rute untuk operasi dasar *create, read, update, delete* (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 *Student*. 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 create, read, update, delete* (CRUD). Dengan adanya fitur ini, seluruh rute tidak harus didefinisikan satu per satu, yang tentu akan memakan waktu dan rentan kesalahan.

f. Membuat *From Create Student.*

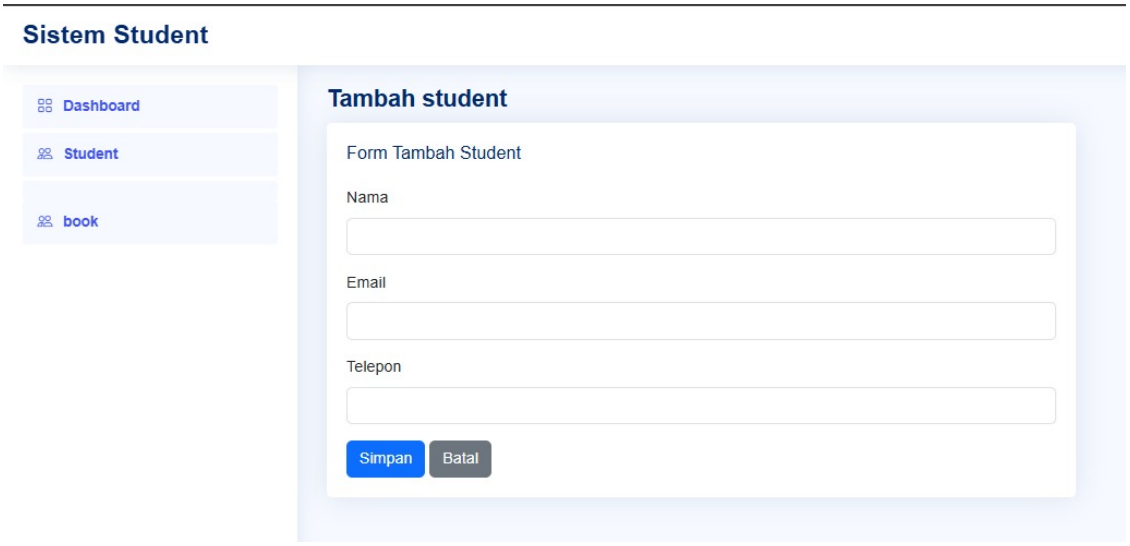
```
<section class="section">
 <div class="row">
  <div class="col-lg-8">

   <div class="card">
    <div class="card-body">
     <h5 class="card-title">Form Tambah Student</h5>

     <form action="{{ route('students.store') }}" method="POST">
     @csrf

     <div class="mb-3">
     <label for="name" class="form-label">Nama</label>
     <input type="text" class="form-control" id="name" name="name" value="{{ old('name') }}" required>
     @error('name') <div class="text-danger">{{ $message}}</div> @enderror
     </div>

<div class="mb-3">
 <label for="email" class="form-label">Email</label>
 <input type="email" class="form-control" id="email" name="email" value="{{ old('email') }}" required>
 @error('email') <div class="text-danger">{{ $message }}</div> @enderror
 </div>
```



Gambar 2. 14 Form *Create* untuk entitas *Student*.

Pada gambar 2.14 merupakan pembuatan dan hasil konten utama (main) untuk entitas *Student* dalam bentuk form *input* yang disusun di dalam file *create.blade.php* menggunakan *Blade Laravel. Form* ini dirancang untuk mengirim data ke *route student.store* dengan metode *POST*, dan dilengkapi proteksi *@csrf* sebagai fitur keamanan. Struktur form memanfaatkan sistem

grid Bootstrap dengan satu input untuk nama, email dan telepon serta tombol bertipe simpan dan batal, 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 *StudentController* akan menangani penyimpanan ke database menggunakan perintah *Student::create*, dan pengguna kemudian diarahkan kembali ke halaman utama entitas *Student* melalui *redirect('student').* Untuk menampilkan data yang telah tersimpan, digunakan *method index*() yang memuat seluruh data dari tabel *student* menggunakan *Student::all*(), lalu mengirimkannya ke *view student.index* melalui fungsi *compact,* sehingga alur proses input hingga penampilan data dapat berlangsung secara terpadu, aman, dan efisien.

g. Membuat Fitur *Delete* pada entitas *Student*.

```
<td>{{ $student->email }}</td>

<td>{{ $student->phone }}</td>

<td>

<a href="{{ route('students.edit', $student->id) }}" class="btn btn-sm btn-warning">Edit</a>

<form action="{{ route('students.destroy', $student->id) }}" method="POST"

class="d-inline" onsubmit="return confirm('Yakin ingin menghapus data ini?')">

@csrf

@method('DELETE')

<button class="btn btn-sm btn-danger">Hapus</button>

</form>
```

Gambar 2. 15 Membuat Fitur *Delete* pada entitas *Student*.

Pada gambar 2.15 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 *student* disusun dalam bentuk tabel dengan menggunakan perulangan *@foreach*, di mana setiap baris menyajikan informasi *student* beserta dua tombol aksi, yaitu Edit dan Hapus. Tombol Edit disusun melalui elemen <a> yang mengarah ke *route student.edit* dengan menyertakan parameter ID, sedangkan tombol Hapus ditempatkan dalam elemen <form> yang diarahkan ke route *student.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 student melalui mekanisme *route* model binding. Setelah data yang dimaksud ditemukan, perintah $*student->delete*() dijalankan untuk menghapus data dari basis data, dan pengguna diarahkan kembali ke halaman utama entitas student dengan perintah *return redirect('student');*.

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 *Student* merupakan entitas master sebagai detailnya. Untuk mengatur keterkaitan ini, digunakan konfigurasi relasi pada model *Laravel hasMany* pada model *student*. Selain itu, struktur basis data pun harus mencantumkan *foreign key* pada tabel *Student*. Pendekatan ini memungkinkan integrasi data yang konsisten dan terstruktur, serta memudahkan dalam pengelolaan antarmuka dan proses *create, read, update, delete* (CRUD) untuk masing- masing entitas yang saling terhubung.

a. Implementasi CRUD pada *student*.

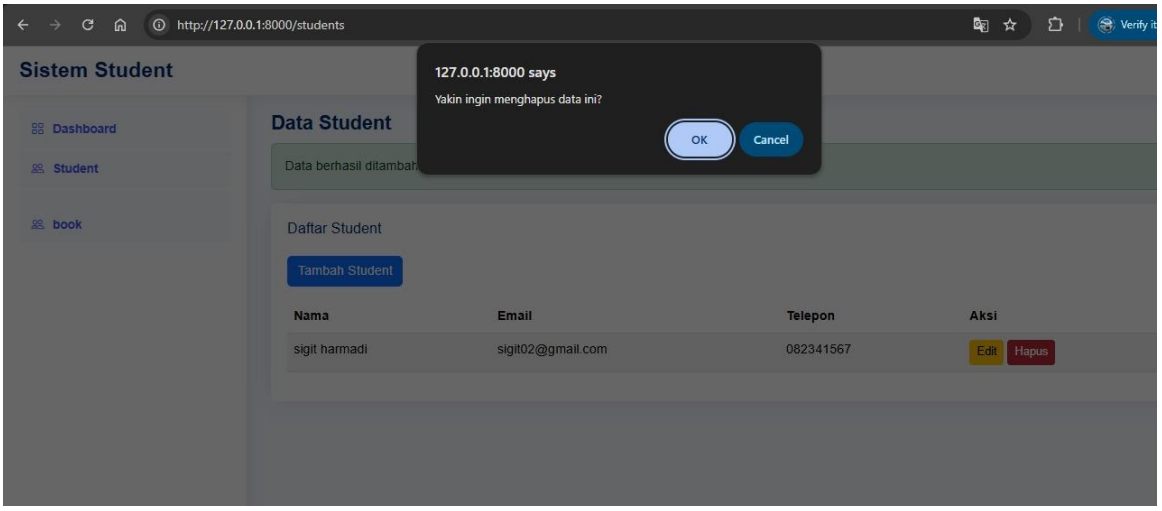
Gambar 2. 16 Implementasi relasi *CRUD* pada *Student*.

Pada gambar 2.16 merupakan proses *create, read, update, delete* (CRUD) pada entitas Student diawali melalui *method index*() pada *StudentController,* di mana seluruh data *Student* diambil menggunakan *Student::all*() dan dikirimkan ke tampilan *student.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 *Student::create('name' => 'required'*,). *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 $*student->update*([...]). Adapun proses penghapusan dilakukan melalui *method destroy*() dengan menjalankan $*student->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.

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.

```
54
55  <a href="{{ route('students.edit', $student->id) }}" class="btn btn-sm btn-warning">Edit</a>
56
57  <form action="{{ route('students.destroy', $student->id) }}" method="POST"
58
59  class="d-inline" onsubmit="return confirm('Yakin ingin menghapus data ini?')">
60
61  @csrf
62
63  @method('DELETE')
64
65  <button class="btn btn-sm btn-danger">Hapus</button>
66
67  </form>
```



Gambar 2. 17 Penambahan fitur *alert*.

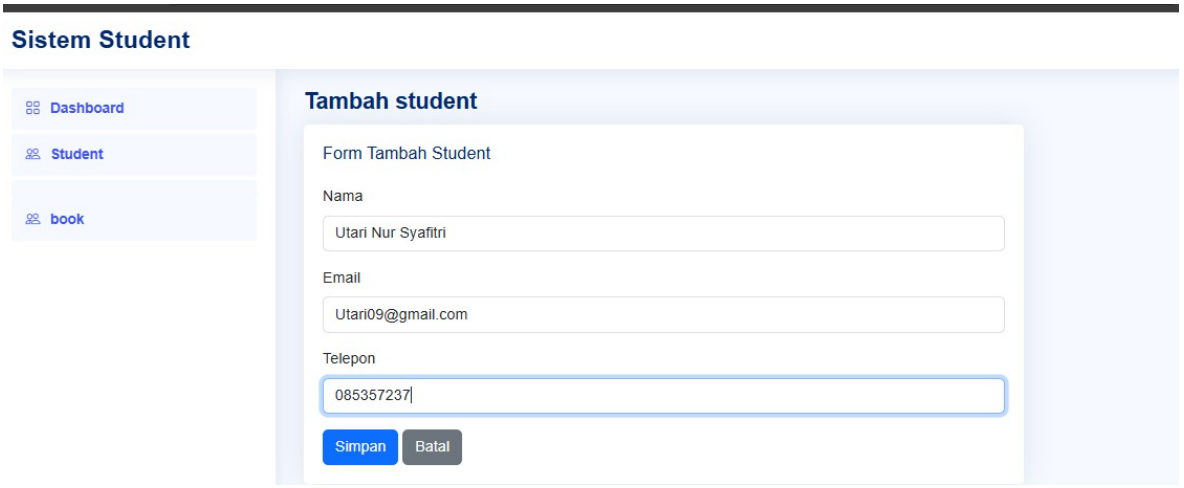
Pada gambar 2.23 ditampilkan implementasi tombol hapus yang telah dilengkapi dengan mekanisme konfirmasi menggunakan *JavaScript.* Kode yang digunakan adalah *<button type="submit" class="btn btn-danger" onclick="return confirm*('Apakah Anda yakin ingin menghapus data ini?')">Hapus*</button>*, di mana 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 student.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

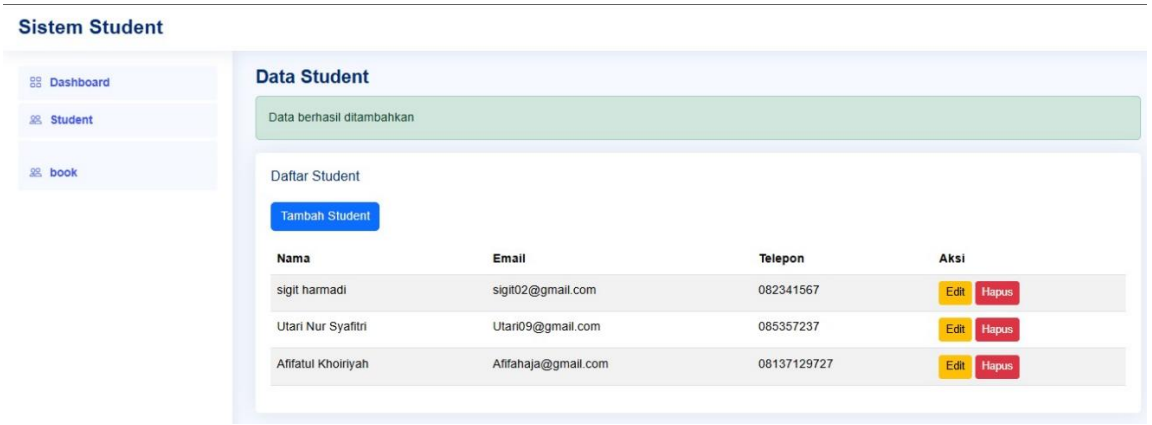
penghapusan dilakukan secara sadar dan disengaja oleh pengguna.

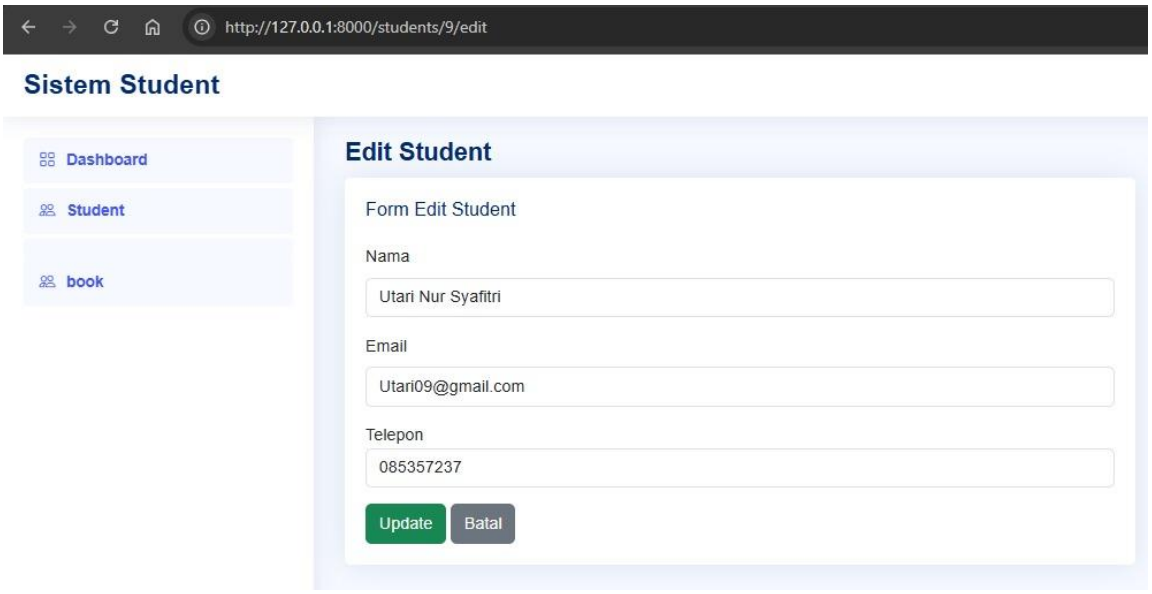
3. Uji coba aplikasi.

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 *student* melalui relasi basis data. Hasil uji coba menunjukkan bahwa interaksi pengguna dengan sistem berjalan lancar, dan setiap fungsi *create, read, update, delete* (CRUD) dapat dijalankan tanpa kendala berarti.

a. Hasil uji coba







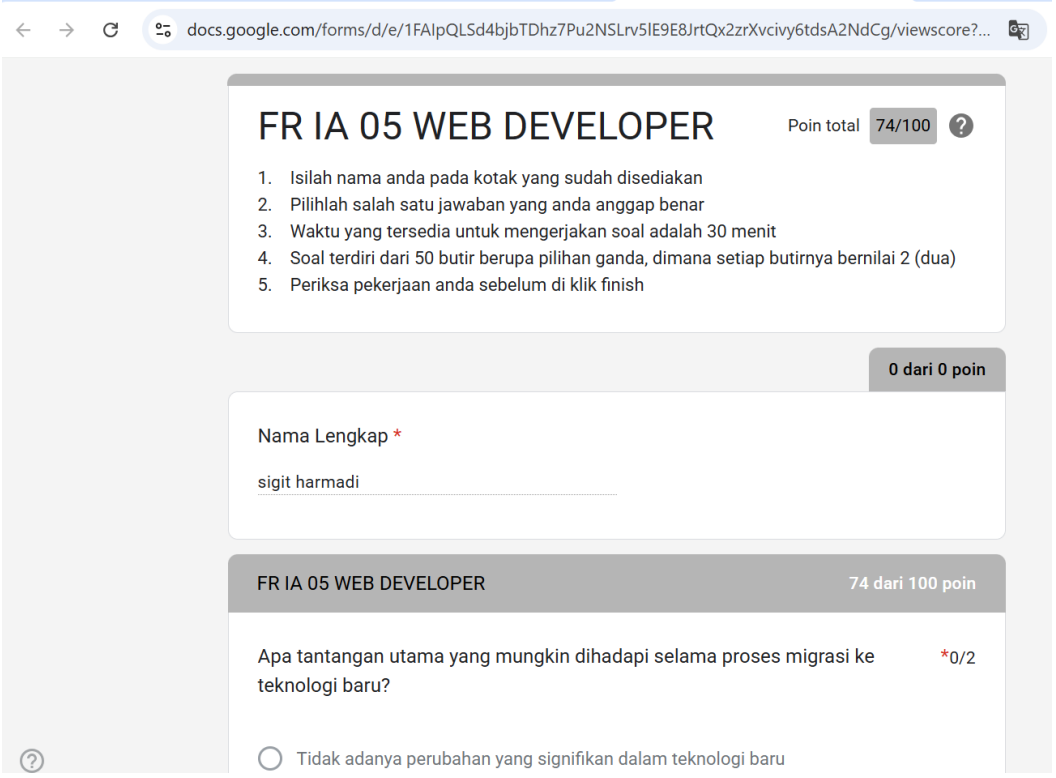
Gambar 2. 18 Hasil uji *student*.

Pada gambar 2.18 memperlihatkan hasil uji coba fitur *Student* yang mencakup proses *input* data, dan tampilan data yang telah disimpan. Pada bagian atas, ditampilkan form untuk menambahkan data *student* baru, terdiri atas kolom nama, email, telepon. Setelah data disimpan, bagian bawah gambar menampilkan daftar student dalam bentuk tabel. Setiap baris menampilkan nama student, serta email dan telepon. Selain itu, tersedia tombol Edit dan Hapus untuk memudahkan pengelolaan data. Tampilan ini menandakan bahwa sistem telah mampu menjalankan seluruh fungsi *create, read, update, delete* (CRUD) pada *student* secara utuh.

2.5 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. 19 Hasil Ujikom Teori.

Pada gambar 2.19 merupakan proses evaluasi dilaksanakan ujian kompetensi lapangan (ujikom). Peserta mengerjakan 50 soal pilihan ganda dalam waktu 30 menit dengan skor yang diperoleh sebesar 74 dari 100. Hasil ini menunjukkan penguasaan awal terhadap materi dasar seperti *framework CSS,* pengelolaan proyek, serta pemahaman awal tentang migrasi teknologi.

3 Sertifikat BNSP Web Developer

13686828

BADAN NASIONAL
SERTIFIKASI PROFESI
INDONESIAN PROFESSIONAL CERTIFICATION AUTHORITY

SERTIFIKAT KOMPETENSI
CERTIFICATE OF COMPETENCE

No. 62019 3514 0 0003632 2025

Dengan ini menyatakan bahwa,
This is to certify that,

Sigit Harmadi

No. Reg. TIK. 2324 01316 2025

Telah kompeten pada bidang:
Is competent in the area 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, 25 Juni 2025

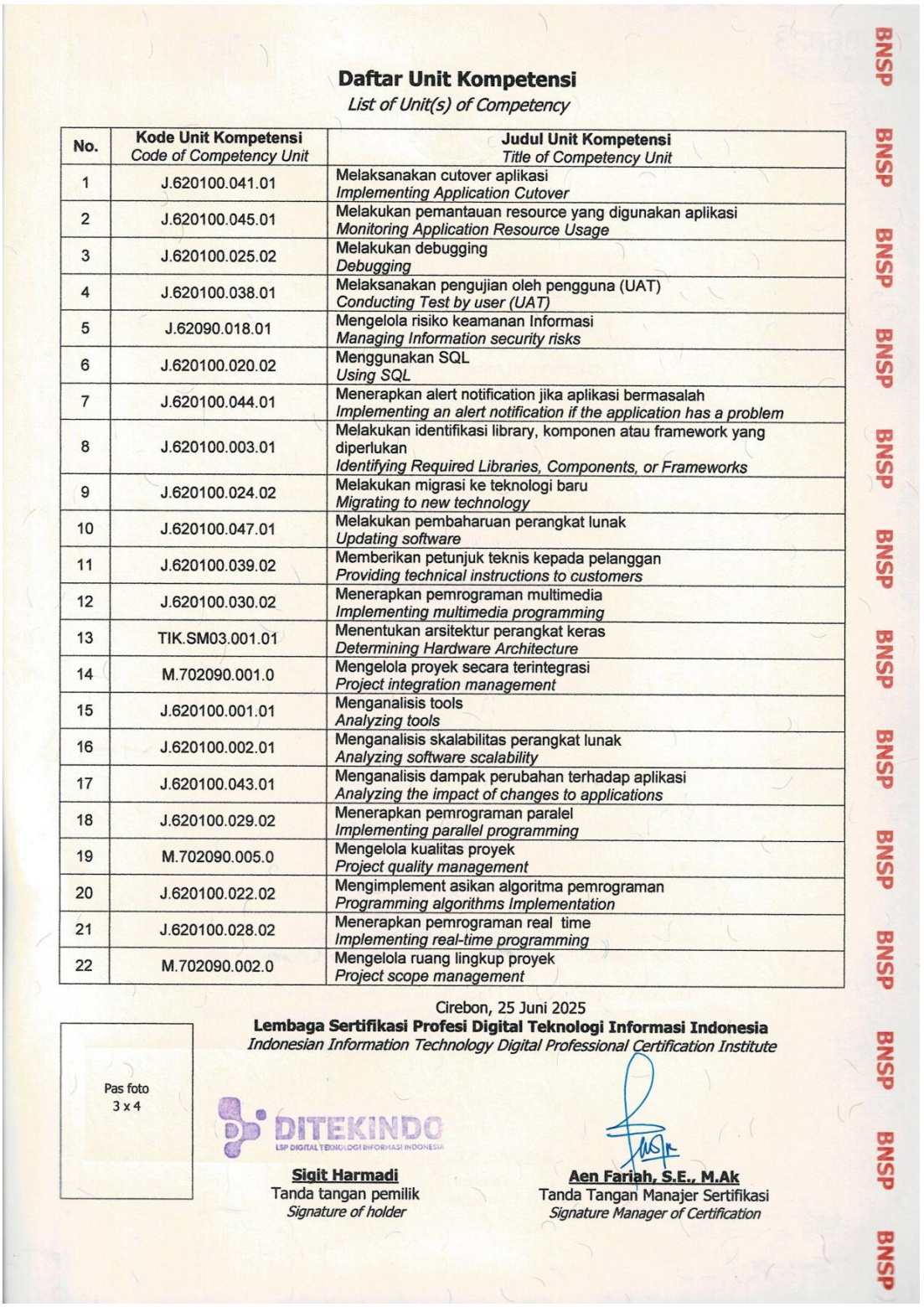
Atas nama Badan Nasional Sertifikasi Profesi (BNSP)
On Behalf of Indonesian Professional Certification Authority

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

DIGIMANDO
LSP DIGITAL TEKNOLOGI INFORMASI INDONESIA

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

BNSP
BADAN NASIONAL SERTIFIKASI PROFESI



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, 25 Juni 2025

Lembaga Sertifikasi Profesi Digital Teknologi Informasi Indonesia

Indonesian Information Technology Digital Professional Certification Institute

Pas foto
3 x 4

DITEKINDO
LSP DIGITAL TEKNOLOGI INFORMASI INDONESIA

Sigit Harmadi
Tanda tangan pemilik
Signature of holder

Aen Fariah, S.E., M.Ak
Tanda Tangan Manajer Sertifikasi
Signature Manager of Certification

BNSP BNSP BNSP BNSP BNSP BNSP BNSP BNSP BNSP BNSP BNSP BNSP BNSP BNSP

Gambar 2. 20 Sertifikat BNSP *Web Developer*.

Pada gambar 2.20 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 *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.

3.1 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 *create, read, update, delete* (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.