

**LAPORAN TUGAS AKHIR**  
**MATA KULIAH PEMROGRAMAN WEB LANJUT**  
**SISTEM INFORMASI MANAJEMEN BPJS KESEHATAN “ORHAINSURANCE”**



Disusun Oleh :

Kelompok 5 Kelas 2020 B

- |                           |             |
|---------------------------|-------------|
| 1. Rizka Nurul Septiani H | 20051397026 |
| 2. Karina Irna Della      | 20051397030 |
| 3. Anggelina Kismasari    | 20051397034 |
| 4. Nisa Amalia            | 20051397038 |

Link Youtube : <https://youtu.be/1o66V2R-VvI>

Link Github : <https://github.com/Anggelina-Kismasari/Tugas-UAS-Pemweb-Lanjut>

Link Web Hosting : <http://orhainsurance.vokasi-unesa.site/>

**PROGRAM STUDI MANAJEMEN INFORMATIKA**

**PROGRAM VOKASI**

**UNIVERSITAS NEGERI SURABAYA**

**2022**

## DAFTAR ISI

DAFTAR ISI .....	i
------------------	---

### BAB I PENDAHULUAN

1.1 Latar Belakang .....	1
1.2 Rumusan Masalah .....	2
1.3 Tujuan .....	2

### BAB II TINJAUAN PUSTAKA

2.1 Sistem Informasi Manajemen .....	3
2.2 BPJS Kesehatan .....	3
2.3 Website .....	3
2.4 PHP .....	4
2.5 MySQL .....	4
2.6 XAMPP .....	4
2.7 LARAVEL .....	4

### BAB III PEMBAHASAN

3.1 Daftar Fitur .....	5
3.2 Daftar Pengguna .....	6
3.3 Grafik Analisa .....	12
3.4 Struktur Kode Script dan Program .....	15
3.5 Tabel Tugas Anggota .....	28

### BAB IV PENUTUP

4.1 Kesimpulan .....	29
4.2 Saran .....	29

# **BAB I**

## **PENDAHULUAN**

### **1.1 Latar Belakang**

Fasilitas Kesehatan merupakan fasilitas pelayanan kesehatan yang digunakan untuk menyelenggarakan upaya pelayanan kesehatan. Fasilitas kesehatan dapat berupa rumah sakit, puskesmas, dan klinik Informasi mengenai fasilitas kesehatan merupakan suatu kebutuhan yang sangat penting, mengingat kebutuhan informasi yang mendesak atau darurat dapat terjadi kapan saja dan dimana saja.

BPJS Kesehatan (Badan Penyelenggara Jaminan Sosial Kesehatan) merupakan suatu Badan Hukum Publik yang bertanggung jawab langsung kepada Presiden dan memiliki tugas untuk menyelenggarakan Jaminan Kesehatan Nasional bagi seluruh rakyat Indonesia. Layanan dari BPJS kesehatan ini adalah bagian dari program pemerintah yaitu Jaminan Kesehatan Nasional (JKN) yang diresmikan pada tanggal 31 Desember 2013 yang sistemnya menggunakan sistem asuransi. Untuk BPJS kesehatan sendiri mulai beroperasi sejak tanggal 1 Januari 2014 dan pengelolaannya sendiri telah didukung oleh sistem informasi/ teknologi informasi yaitu sistem informasi manajemen BPJS kesehatan.

Walaupun telah dua tahun beroperasi, terdapat berbagai masalah diantaranya data peserta di seluruh penyelenggara jaminan kesehatan belum terhimpun dengan baik, masih banyaknya keluhan pelayanan BPJS pada pelayanan tingkat pertama di Puskesmas maupun klinik sampai ke sistem rujukan ke rumah sakit dan masalah akurasi pendataan penerima bantuan iuran untuk warga miskin.

Banyaknya kelemahan dari layanan BPJS kesehatan yang telah didukung SI/TI juga dapat berdampak buruk terhadap citra rumah sakit yang ada di Indonesia. Padahal pada masa kini, pemanfaatan SI/ TI sudah menjadi kebutuhan yang utama pada dunia industri baik di bidang kesehatan maupun non kesehatan, karena teknologi informasi memberikan peluang terjadinya transformasi dan peningkatan produktivitas bisnis menjadi semakin cepat serta dapat meningkatkan daya saing rumah sakit, sehingga pada penelitian ini bertujuan untuk mengukur keefektifan layanan dan menghitung level kematangan pengelolaan SI/TI layanan BPJS kesehatan pada rumah sakit yang melayani BPJS agar level manajemen TI pada kondisi saat ini (as-is) dapat mencapai level manajemen TI yang diharapkan (to-be).

Berdasarkan permasalahan yang telah diuraikan, maka penulis ingin meneliti bagaimana mengembangkan sistem yang dapat mengelola keefektifan layanan dan menghitung level kematangan pengelolaan SI/TI layanan BPJS Kesehatan, agar terorganisasi dan dapat membantu para rumah sakit sehingga dapat meningkatkan layanan kesehatan di Indonesia.

## 1.2 Rumusan Masalah

1. Bagaimana mengembangkan sistem informasi manajemen yang baik dalam mengelola data BPJS Kesehatan?
2. Bagaimana teknik pengolahan data asuransi di dalam sistem informasi manajemen BPJS Kesehatan?

## 1.3 Tujuan

1. Mempermudah dan mempercepat dalam mendapatkan informasi yang dibutuhkan.
2. Dapat mengelola data asuransi sehingga dapat tertata dan saling berhubungan.
3. Membantu pegawai di sektor umum dan SDM dalam memantau pendaftaran dan penjadwalan pembayaran iuran.

## **BAB II**

### **TINJAUAN PUSTAKA**

#### **2.1 Sistem Informasi Manajemen**

Menurut Danu Wira Pangestu (2007) Sistem Informasi Manajemen merupakan kumpulan dari interaksi sistem-sistem informasi yang berwenang dalam mengumpulkan dan mengolah data guna menyediakan informasi yang bermanfaat bagi semua tingkatan manajemen di dalam kegiatan perencanaan dan pengendalian. Sedangkan menurut Joel. D. Aron di dalam buku tulisan E.S Margianti disebutkan bahwa Sistem Informasi Manajemen adalah sebuah sistem yang memberikan informasi yang dibutuhkan oleh manajer dalam membuat keputusan. Sehingga dapat disimpulkan bahwa Sistem Informasi Manajemen dijalankan menggunakan software dan hardware serta terdiri dari kumpulan interaksi yang dapat digunakan sebagai pembuat keputusan.

#### **2.2 BPJS Kesehatan**

BPJS Kesehatan (Badan Penyelenggara Jaminan Sosial Kesehatan) adalah sebuah badan hukum publik yang mengatur jaminan kesehatan bagi seluruh warga Indonesia. Badan hukum ini diresmikan pada tanggal 31 Desember 2013 bersama BPJS Ketenagakerjaan dan mulai beroperasi pada tanggal 1 Januari 2014. Para penerima jaminan kesehatan seperti PNS, penerima pensiunan, veteran, pemilik badan usaha, maupun rakyat sipil wajib untuk daftar BPJS apabila sudah bekerja di Indonesia minimal 6 bulan. Hal ini sesuai dengan pasal 14 UU BPJS. Selain itu pekerja di sektor informal juga wajib menjadi anggota BPJS. Para pekerja harus mendaftarkan diri dan membayar iuran dengan besaran yang ditentukan. Sedangkan untuk masyarakat yang kurang mampu akan mendapatkan bantuan iuran dari pemerintah.

#### **2.3 Website**

Website adalah kumpulan halaman web yang saling terhubung dan seluruh file saling terkait. Web terdiri dari page atau halaman dan kumpulan halaman yang dinamakan *homepage*. Homepage berada pada posisi teratas dengan halaman-halaman terkait di bawahnya. Biasanya, setiap halaman di bawah homepage (child page) berisi *hyperlink* ke halaman lain dalam web (Gregorius, 2000).

## 2.4 PHP

PHP adalah sebuah bahasa pemrograman server side scripting yang bersifat open source. Sebagai sebuah scripting language, PHP menjalankan instruksi pemrograman saat proses runtime. Hasil dari instruksi tentu akan berbeda tergantung data yang diproses. PHP merupakan bahasa pemrograman server-side, maka script dari PHP nantinya akan diproses di server. Jenis server yang sering digunakan bersama dengan PHP antara lain Apache, Nginx, dan LiteSpeed. Selain itu, PHP juga merupakan bahasa pemrograman yang bersifat open source. Pengguna bebas memodifikasi dan mengembangkan sesuai dengan kebutuhan mereka. Kelompok kami menggunakan PHP versi 8.

## 2.5 MySQL

MySQL adalah salah satu jenis database yang bersifat open source. Tentunya, banyak sekali bentuk database selain MySQL sendiri. Dalam pembuatan sebuah aplikasi yang kompleks dan dapat dijalankan secara dinamis, database sangatlah dibutuhkan untuk menyimpan berbagai data dalam bentuk informasi. Kami menggunakan phpMyAdmin untuk menyimpan dan mengedit database yang kelompok kami gunakan.

## 2.6 XAMPP

adalah perangkat lunak bebas, yang mendukung banyak sistem operasi, merupakan kompilasi dari beberapa program. Fungsinya adalah sebagai server yang berdiri sendiri (localhost), yang terdiri atas program Apache HTTP Server, MySQL database, dan penerjemah bahasa yang ditulis dengan bahasa pemrograman PHP dan Perl. Nama XAMPP merupakan singkatan dari X (tempat sistem operasi apapun), Apache, MySQL, PHP dan Perl. Program ini tersedia dalam GNU General Public License dan bebas, merupakan web server yang mudah digunakan yang dapat melayani tampilan halaman web yang dinamis.

## 2.7 LARAVEL

adalah kerangka kerja aplikasi web berbasis PHP yang sumber terbuka, menggunakan konsep Model-View-Controller (MVC). Laravel berada dibawah lisensi MIT, dengan menggunakan GitHub. Kelompok kami menggunakan laravel framework versi 9.15.0

## **BAB III**

### **PEMBAHASAN**

#### **3.1 Daftar Fitur**

**a. Login dan Register**

Pengguna dapat melakukan login untuk masuk ke dalam akun yang telah dibuat sebelumnya. Hal ini bertujuan untuk memberikan akses kepada pengguna dalam penggunaan fitur-fitur di dalam website yang bersifat personal. Apabila pengguna belum memiliki akun, maka diwajibkan untuk membuat akun terlebih dahulu pada bagian registrasi.

**b. Home**

Di halaman ini terdapat ucapan selamat datang untuk pengguna dan pilihan menu website seperti About Us, KCU BPJS, Info, Contact Us, Appointment, Login, dan Register.

**c. About Us**

Berisi deskripsi informasi umum mengenai BPJS Kesehatan.

**d. KCU BPJS**

Halaman ini berisi informasi mengenai letak Kantor Cabang Umum BPJS Surabaya beserta alamat lengkapnya. Dan berisi seputar informasi mengenai fasilitas kesehatan yaitu ketersediaan kamar untuk pengguna BPJS Kesehatan di rumah sakit kota Surabaya. Data yang ditampilkan juga sangat detail mulai dari nama rumah sakit dan ketersediaan berbagai kategori kamar seperti VVIP, VIP, I, II, III, kamar UGD, dan kamar Isolasi.

**e. Info**

Halaman ini berisi informasi berita terkini di dunia kesehatan dan kedokteran serta berisi seputar pertanyaan tentang BPJS Kesehatan.

**f. Contact Us**

Halaman ini berisi form yang dapat digunakan oleh pengguna untuk mengirim pertanyaan atau kritik dan saran. Di bawah form ini juga terdapat pertanyaan serta jawaban seputar BPJS Kesehatan.

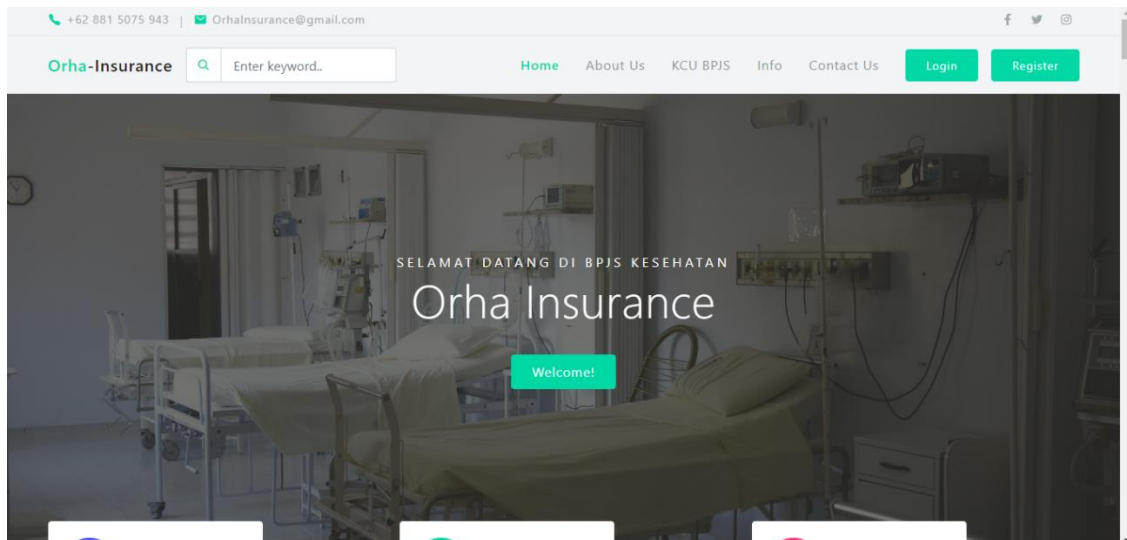
**g. Appointment**

Halaman ini berisi untuk mengecek pesan yang kita kirim sudah diterima oleh Admin atau belum sehingga nanti Admin mengapprove pesan tersebut.

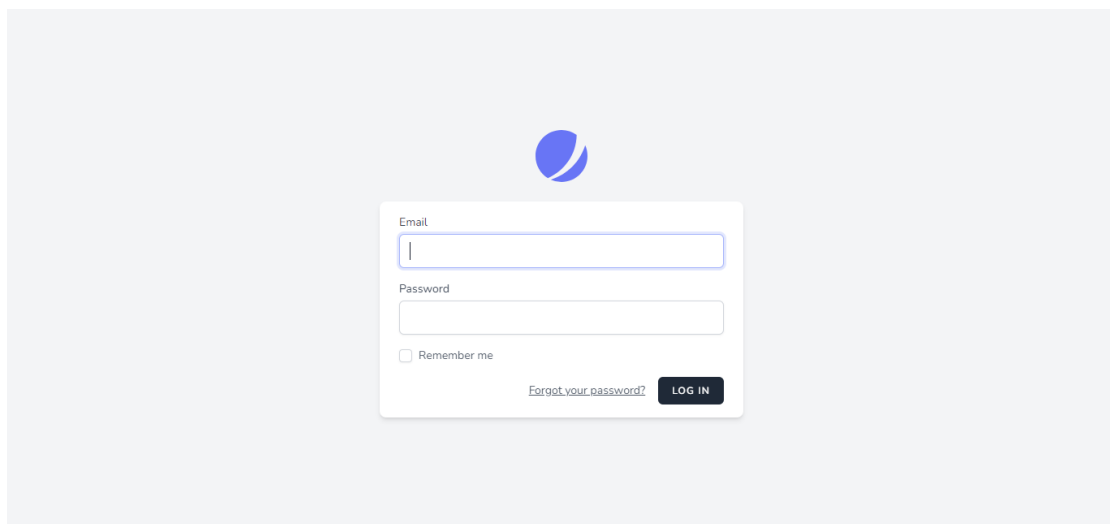
## 3.2 Daftar Pengguna

### a. User

- Tampilan halaman Home user sebelum melakukan Login/Registrasi

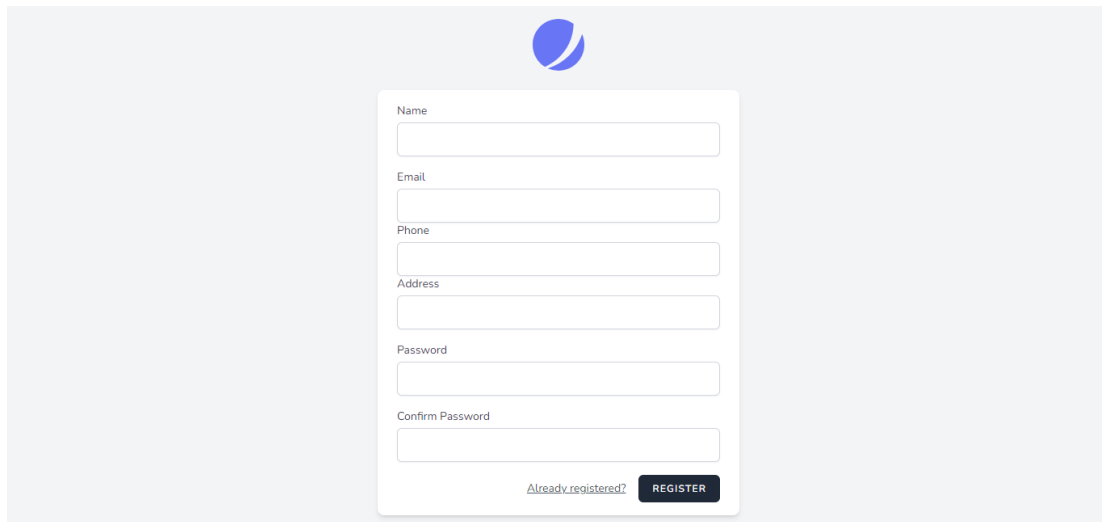


- Tampilan halaman Login user



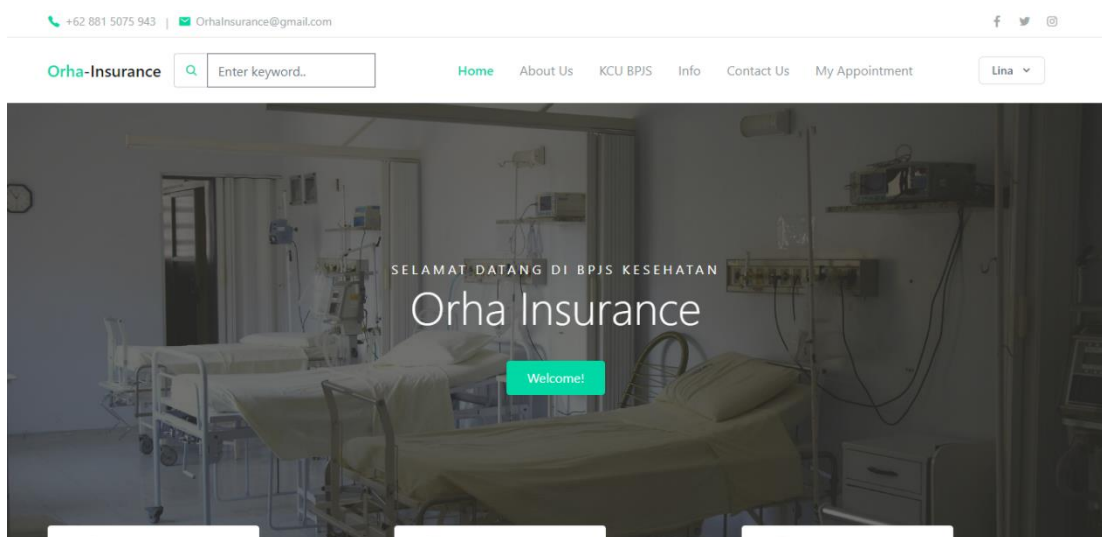


- Tampilan halaman Registrasi user

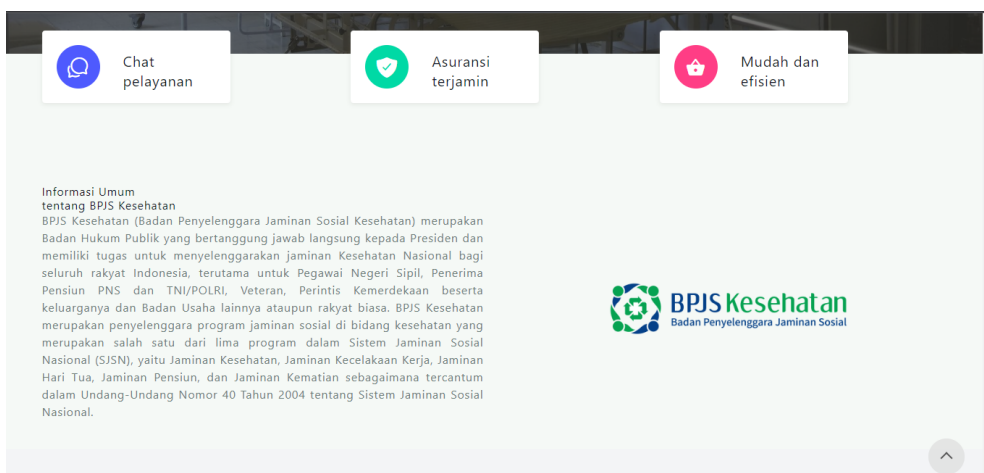


The registration form is centered on a light gray background. At the top is a blue circular logo with a white swoosh. The form itself is a white box with the following fields: Name, Email, Phone, Address, Password, and Confirm Password. Each field has a corresponding input box. Below the fields, there is a link that says "Already registered?" and a dark blue button labeled "REGISTER".

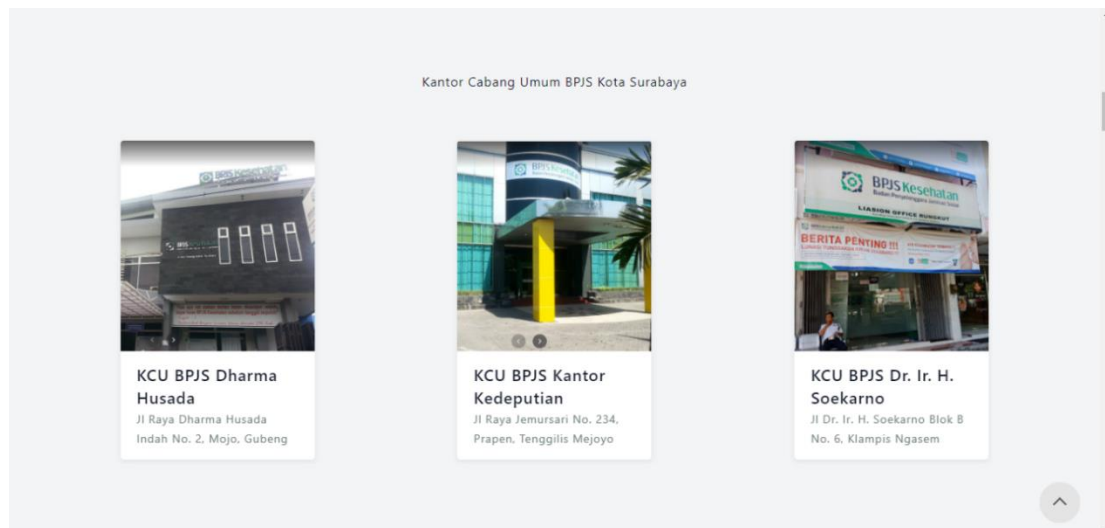
- Tampilan halaman Home user setelah melakukan Login/Registrasi



- Tampilan halaman About Us



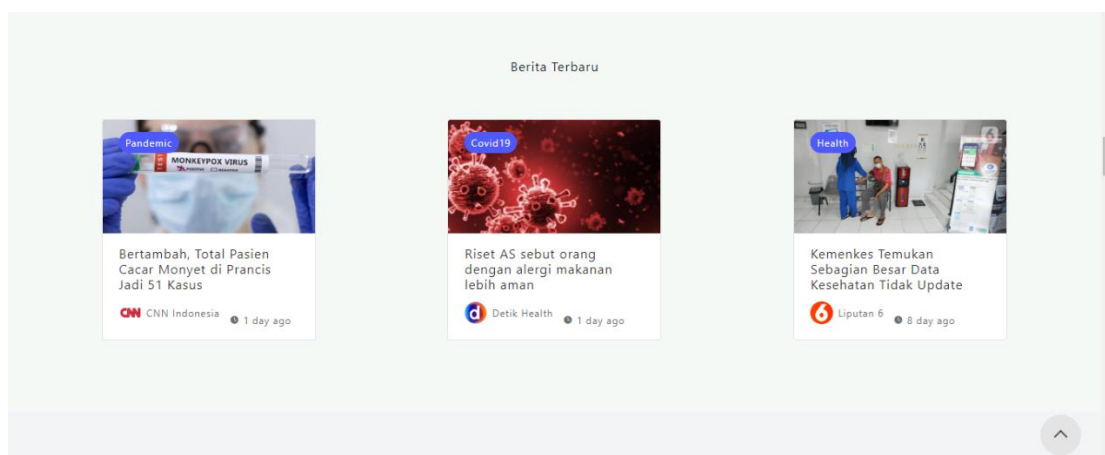
- Tampilan halaman KCU BPJS.



**Fasilitas Kesehatan Ketersediaan Kamar Untuk BPJS Kesehatan Di Rumah Sakit Kota Surabaya**

Nama	VVIP	VIP	I	II	III	UGD	ISOLASI
RSIA BANTUAN 05.08.05 SURABAYA	2	2	1	3	8	0	0
Rumkitalmar Ewa Pangalila	1	0	1	23	17	0	11
Rumkit Soemitro Lanud Mulyono	0	2	8	18	23	0	2
RSUD Bhakti Dharma Husada	0	0	20	18	10	0	42
Rumkital Dr. Oepomo	0	0	5	14	22	0	13
RS BUNDA SURABAYA	1	0	0	1	6	0	0
RUMAH SAKIT MATA UNDAAN	1	1	2	6	15	0	4
RS Universitas Airlangga	0	2	0	0	0	0	79
RS Wijaya	1	3	5	2	12	0	2
RS Royal Surabaya	5	0	52	34	23	0	8
RS BHAKTI RAHAYU	0	3	8	29	11	0	12
RSI DARUS SYIFA	0	12	35	26	83	0	33
RS WIYUNG SEJAHTERA	0	0	3	0	0	0	4
RS WILLIAM BOOTH SURABAYA	1	12	26	34	24	0	3
RS MANYAR MEDICAL CENTRE	1	3	3	5	8	0	27
SILOAM HOSPITALS SURABAYA	7	2	8	5	4	0	0
RSIA PERDANA MEDICA	2	2	5	2	8	0	0

- Tampilan halaman Info fasilitas Kesehatan dan Berita



Seputar **BPJS Kesehatan**

Apakah yang dimaksud dengan BPJS Kesehatan?

Apakah yang dimaksud dengan Iuran Jaminan Kesehatan?

Apa yang dimaksud dengan Pelayanan Kesehatan Tingkat Pertama?

Apa yang dimaksud dengan Rawat Jalan Tingkat Pertama?

- Tampilan halaman Contact Us user

Hubungi kami untuk info seputar BPJS

Nama Lengkap

Alamat Email

mm/dd/yyyy

--Pilih Kantor Cabang Umum--

No Telp

Masukkan Pesan

Submit Request

- Tampilan halaman My Appointment user

+62 881 5075 943 | OrhaInsurance@gmail.com

f t @ @

Orha-Insurance

Enter keyword..

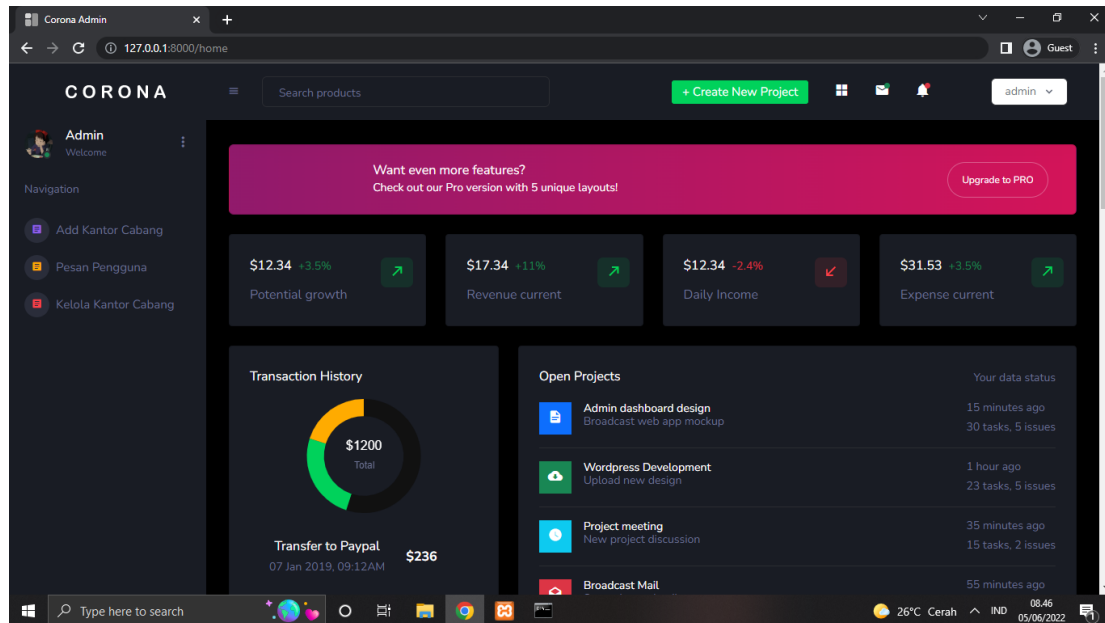
[Home](#)
[About Us](#)
[FasKes](#)
[Info](#)
[Contact Us](#)
[My Appointment](#)

Lina ▾

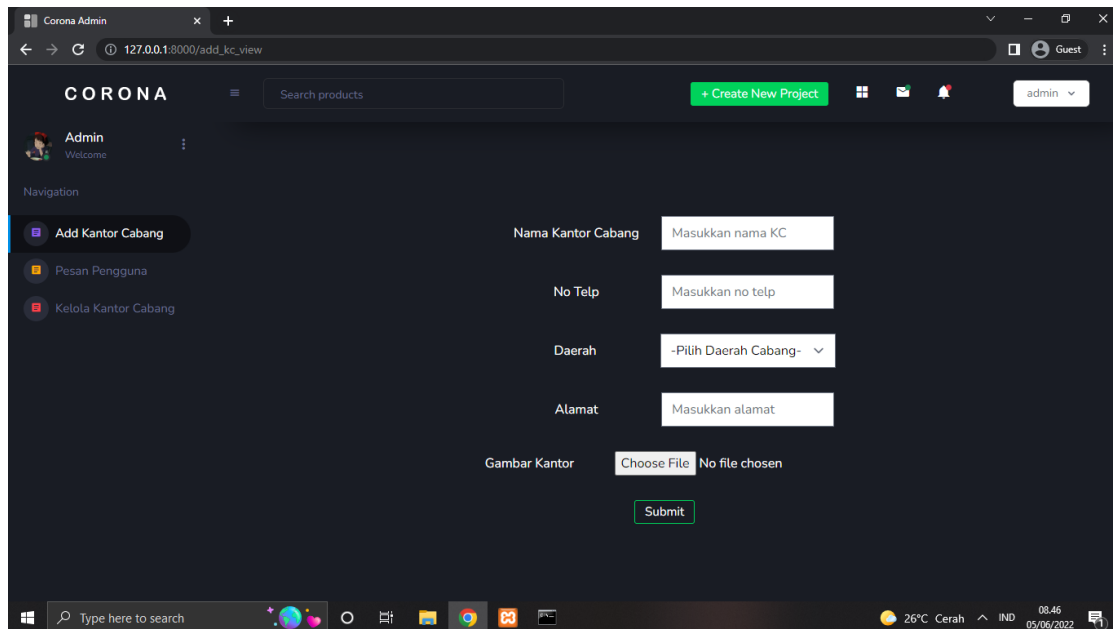
Kantor Cabang	Tanggal	Pesan	Status	Batal
KCU BPJS Dr. Ir. H. Soekarno	2022-06-25	Tes	In Progress	Batal
KCU BPJS Dr. Ir. H. Soekarno	2022-06-12	Hello	In Progress	Batal

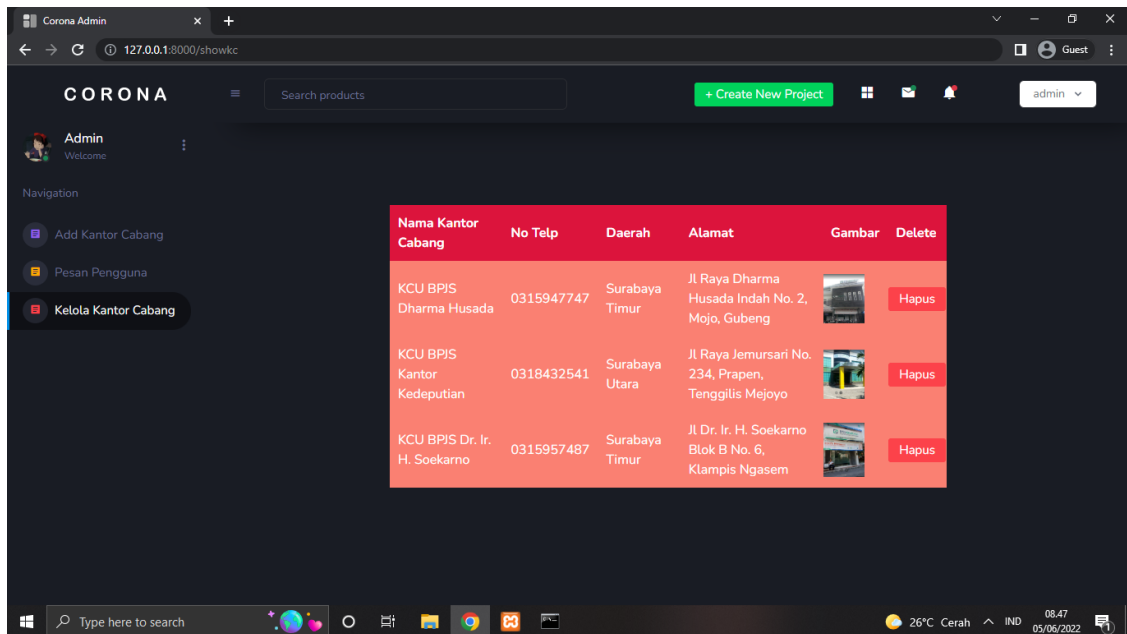
## b. Admin

- Tampilan akses halaman Home Admin

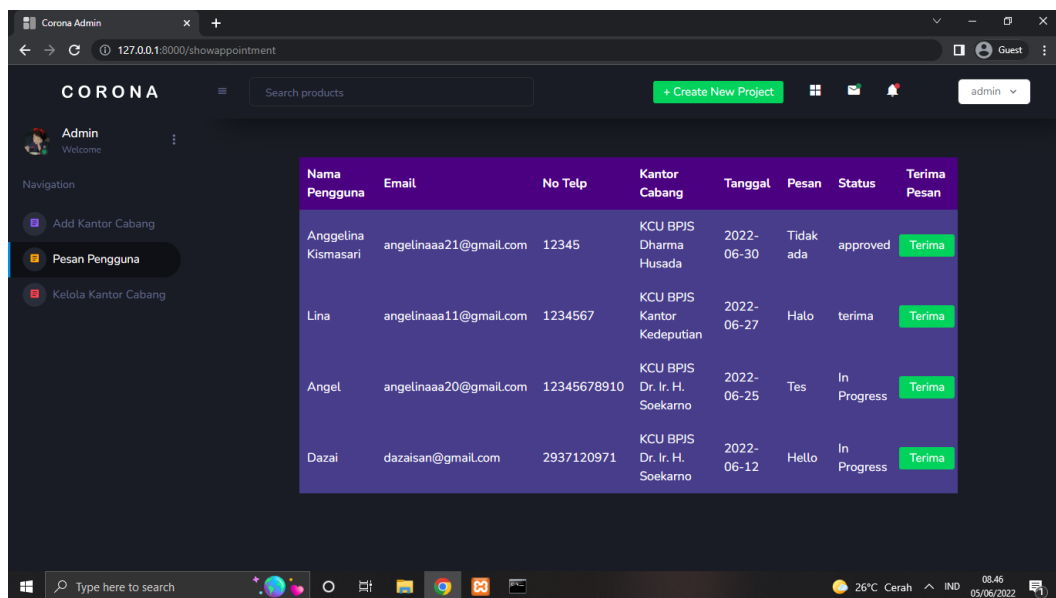


- Tampilan akses halaman admin data Kantor Cabang



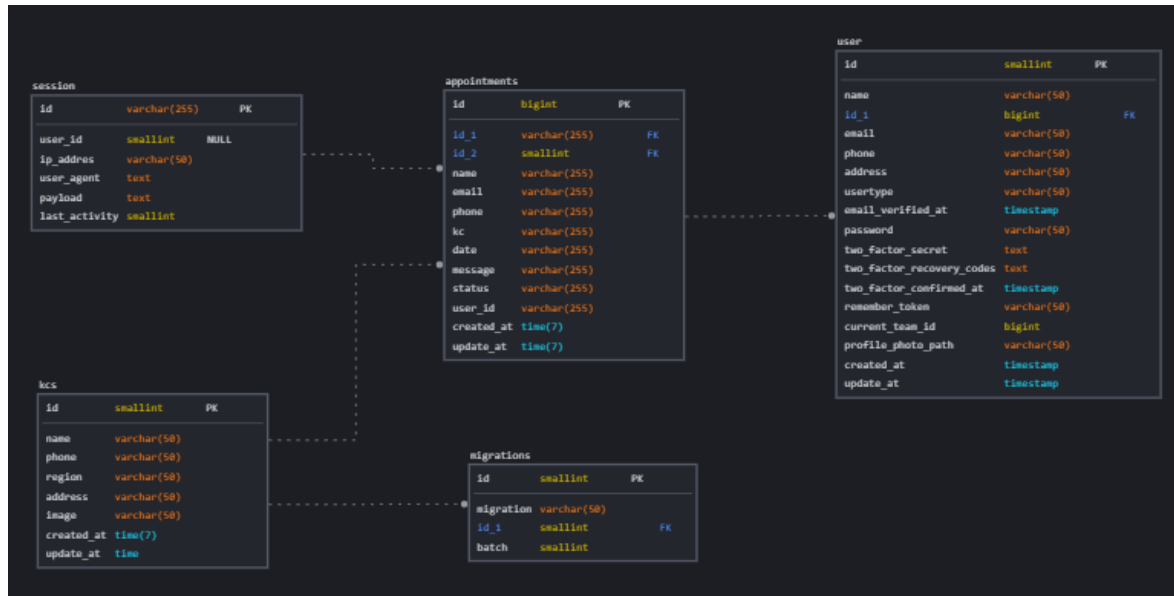


- Tampilan akses halaman admin pesan pengguna atau pengisian form Contact Us di halaman user.

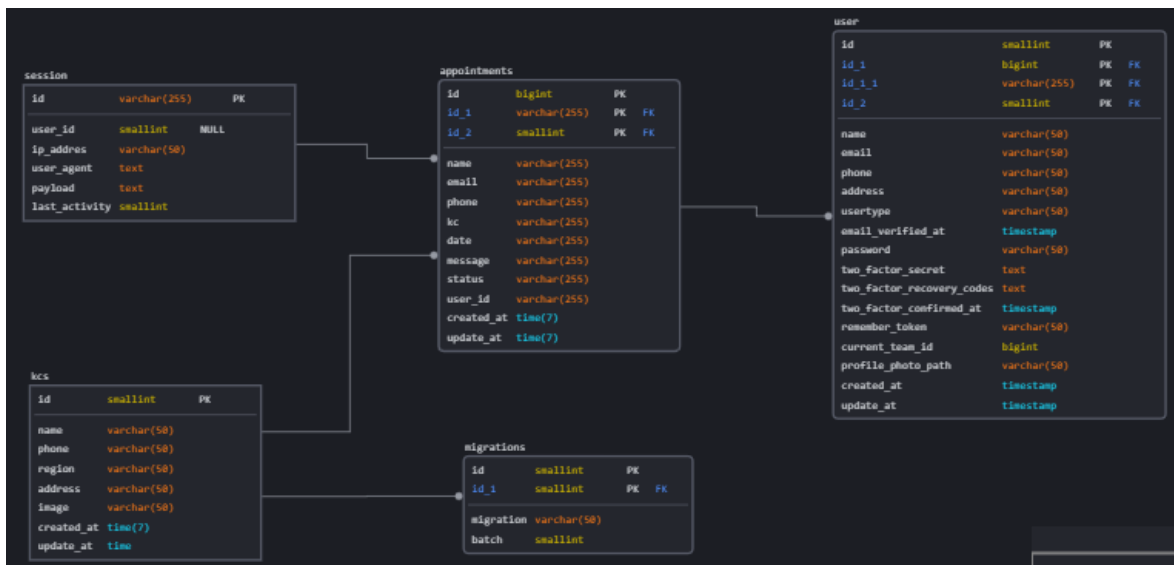


### 3.3 Grafik Analisa

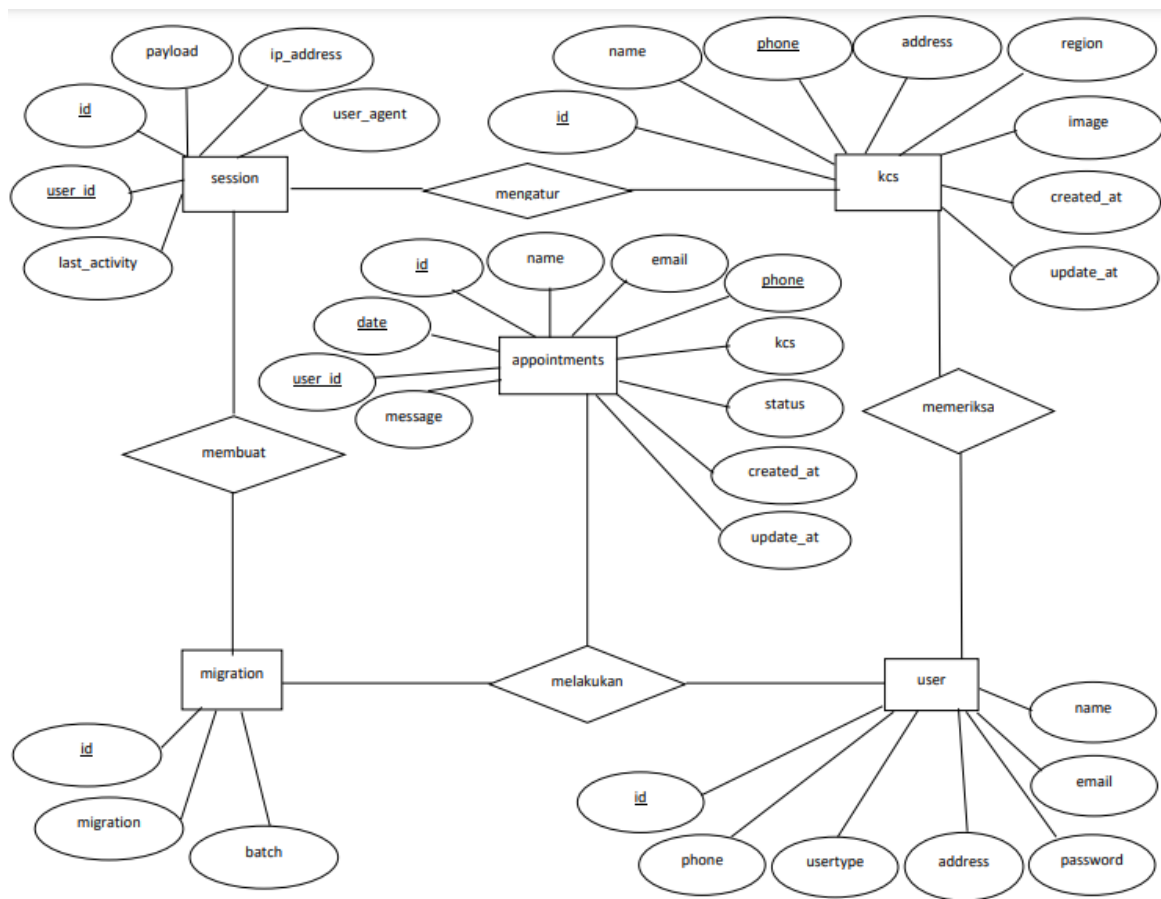
#### a. CDM (Conceptual Data Model)



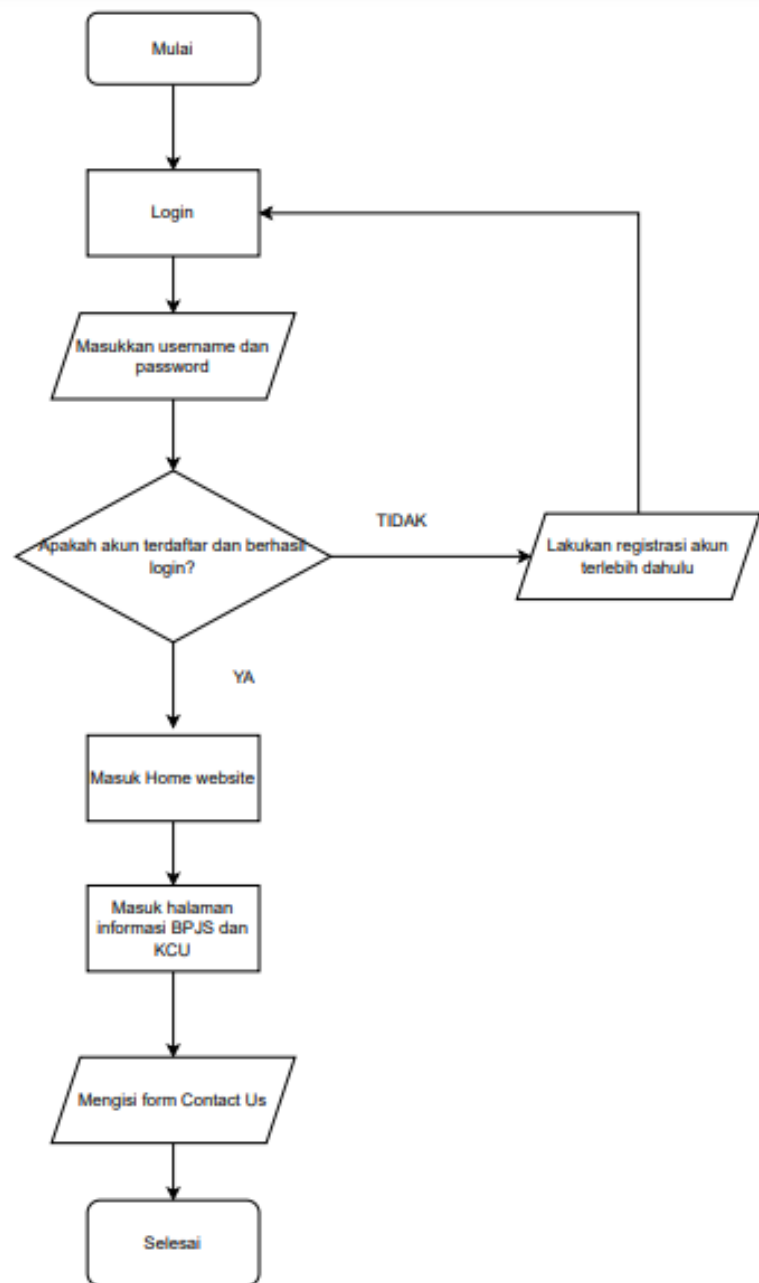
#### b. PDM (Physical Data Model)



### c. ER-Diagram

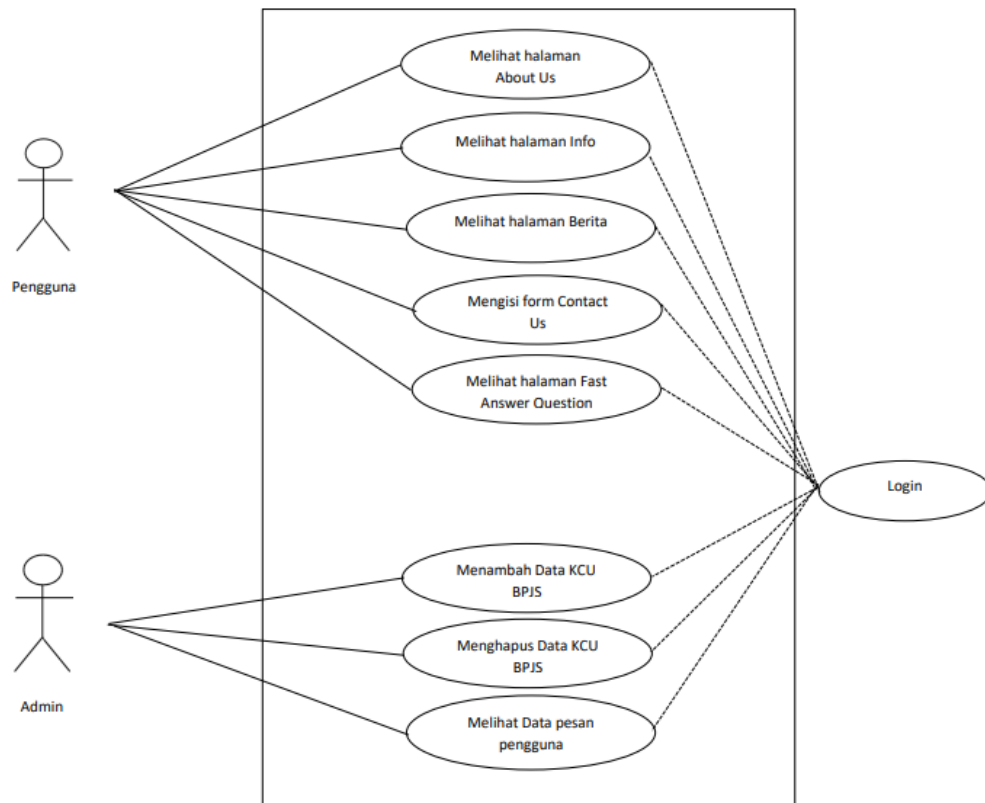


d. Flowchart





### e. Use Case Diagram



## 3.4 Struktur Kode Script dan Program

### a. Routes

```
EXPLORER
└─ ROUTES
  └─ api.php
  └─ channels.php
  └─ console.php
  └─ web.php

api.php
1  <?php
2
3  use Illuminate\Http\Request;
4  use Illuminate\Support\Facades\Route;
5
6  /*
7  |-----
8  | API Routes
9  |-----
10 |
11 | Here is where you can register API routes for your application. These
12 | routes are loaded by the RouteServiceProvider within a group which
13 | is assigned the "api" middleware group. Enjoy building your API!
14 |
15 |*/
16
17 Route::middleware('auth:sanctum')->get('/user', function (Request $request) {
18     return $request->user();
19 });
20
```

```
EXPLORER
...
channels.php X
channels.php
1 <?php
2
3 use Illuminate\Support\Facades\Broadcast;
4
5 /*
6 |-----
7 | Broadcast Channels
8 |-----
9 |
10 | Here you may register all of the event broadcasting channels that your
11 | application supports. The given channel authorization callbacks are
12 | used to check if an authenticated user can listen to the channel.
13 |
14 */
15
16 Broadcast::channel('App.Models.User.{id}', function ($user, $id) {
17     return (int) $user->id === (int) $id;
18 });
19
```

```
EXPLORER
...
console.php X
console.php
1 <?php
2
3 use Illuminate\Foundation\Inspiring;
4 use Illuminate\Support\Facades\Artisan;
5
6 /*
7 |-----
8 | Console Routes
9 |-----
10 |
11 | This file is where you may define all of your Closure based console
12 | commands. Each Closure is bound to a command instance allowing a
13 | simple approach to interacting with each command's IO methods.
14 |
15 */
16
17 Artisan::command('inspire', function () {
18     $this->comment(Inspiring::quote());
19 })->purpose('Display an inspiring quote');
20
```

```
EXPLORER
...
web.php X
web.php
1 <?php
2
3 use Illuminate\Support\Facades\Route;
4
5 use App\Http\Controllers\HomeController;
6
7 use App\Http\Controllers\AdminController;
8
9 /*
10 |-----
11 | Web Routes
12 |-----
13 |
14 | Here is where you can register web routes for your application. These
15 | routes are loaded by the RouteServiceProvider within a group which
16 | contains the "web" middleware group. Now create something great!
17 |
18 */
19
20 Route::get('/login', function ()
21 {
22     return view('auth/login');
23 })->name('login');
24
25 Route::get('/register', function ()
26 {
27     return view('auth/register');
28 })->name('register');
29
```

ROUTES

api.php

channels.php

console.php

web.php

web.php

```

29
30 Route::get('/',[HomeController::class,'index']);
31
32 Route::get('/home',[HomeController::class,'redirect']);
33
34 Route::middleware([
35     'auth:sanctum',
36     config('jetstream.auth_session'),
37     'verified'
38 ])->group(function () {
39     Route::get('/dashboard', function () {
40         return view('dashboard');
41     }->name('dashboard'));
42 });
43
44 Route::get('/add_kc_view',[AdminController::class,'addview']);
45
46 Route::post('/upload_kc',[AdminController::class,'upload']);
47
48 Route::post('/appointment',[HomeController::class,'appointment']);
49
50 Route::get('/myappointment',[HomeController::class,'myappointment']);
51
52 Route::get('/cancel_appoint/{id}',[HomeController::class,'cancel_appoint']);
53
54 Route::get('/showappointment',[AdminController::class,'showappointment']);
55
56 Route::get('/approved/{id}',[AdminController::class,'approved']);
57
58 Route::get('/showkc',[AdminController::class,'showkc']);
59
60 Route::get('/deletekc/{id}',[AdminController::class,'daletekc']);

```

## b. Models

EXPLORER

...

Appointment.php

Kc.php

User.php

Appointment.php

```

1 <?php
2
3 namespace App\Models;
4
5 use Illuminate\Database\Eloquent\Factories\HasFactory;
6 use Illuminate\Database\Eloquent\Model;
7
8 class Appointment extends Model
9 {
10     use HasFactory;
11 }
12

```

EXPLORER

...

Appointment.php

Kc.php

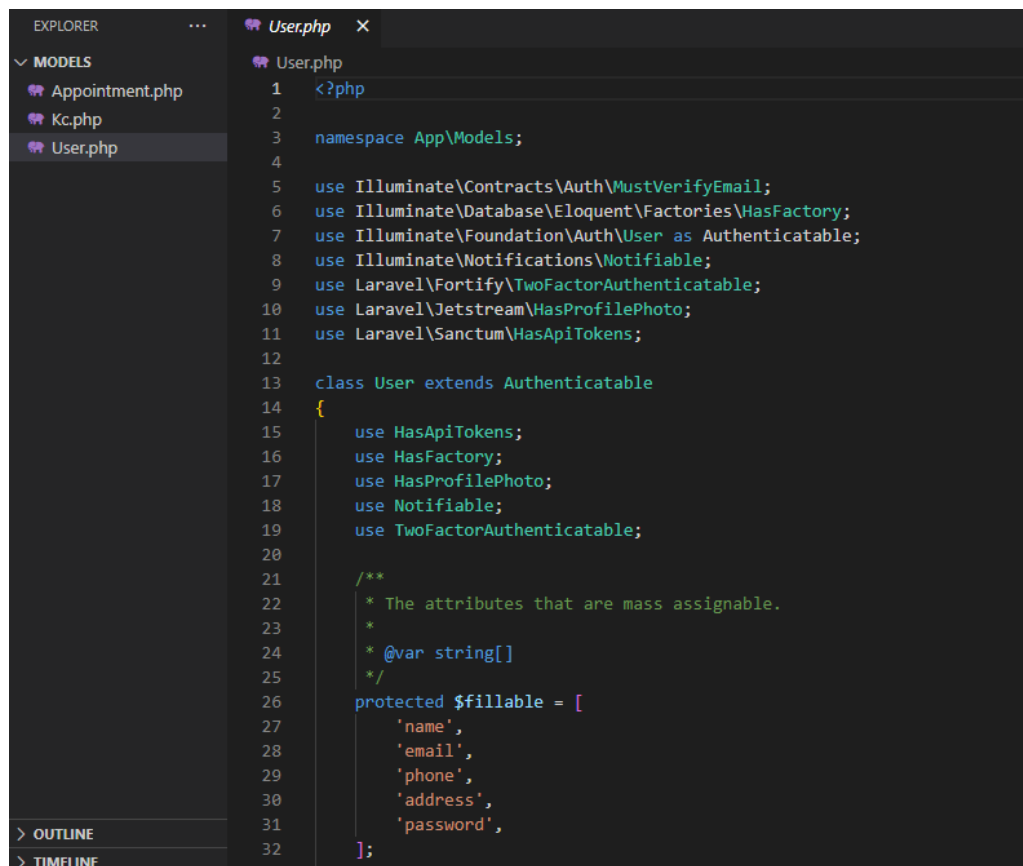
User.php

Kc.php

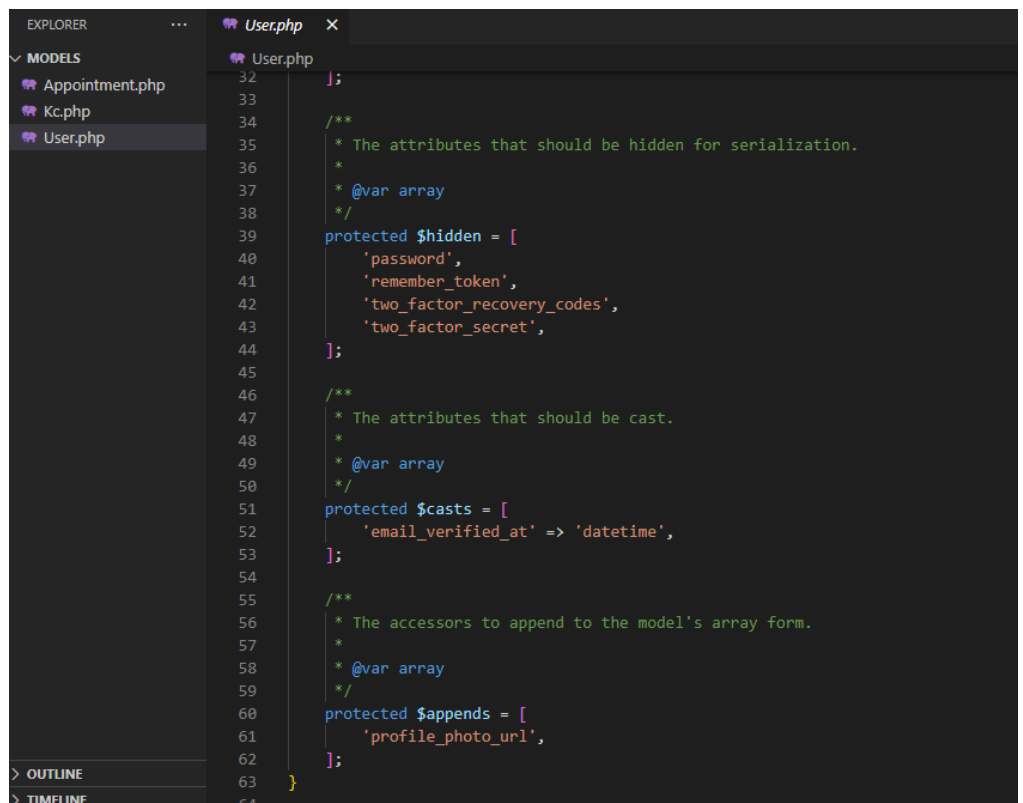
```

1 <?php
2
3 namespace App\Models;
4
5 use Illuminate\Database\Eloquent\Factories\HasFactory;
6 use Illuminate\Database\Eloquent\Model;
7
8 class Kc extends Model
9 {
10     use HasFactory;
11 }
12

```



```
1  <?php
2
3  namespace App\Models;
4
5  use Illuminate\Contracts\Auth\MustVerifyEmail;
6  use Illuminate\Database\Eloquent\Factories\HasFactory;
7  use Illuminate\Foundation\Auth\User as Authenticatable;
8  use Illuminate\Notifications\Notifiable;
9  use Laravel\Fortify\TwoFactorAuthenticatable;
10 use Laravel\Jetstream\HasProfilePhoto;
11 use Laravel\Sanctum\HasApiTokens;
12
13 class User extends Authenticatable
14 {
15     use HasApiTokens;
16     use HasFactory;
17     use HasProfilePhoto;
18     use Notifiable;
19     use TwoFactorAuthenticatable;
20
21     /**
22      * The attributes that are mass assignable.
23      *
24      * @var string[]
25      */
26     protected $fillable = [
27         'name',
28         'email',
29         'phone',
30         'address',
31         'password',
32     ];
```



```
32 ];
33
34 /**
35  * The attributes that should be hidden for serialization.
36  *
37  * @var array
38  */
39 protected $hidden = [
40     'password',
41     'remember_token',
42     'two_factor_recovery_codes',
43     'two_factor_secret',
44 ];
45
46 /**
47  * The attributes that should be cast.
48  *
49  * @var array
50  */
51 protected $casts = [
52     'email_verified_at' => 'datetime',
53 ];
54
55 /**
56  * The accessors to append to the model's array form.
57  *
58  * @var array
59  */
60 protected $appends = [
61     'profile_photo_url',
62 ];
63 }
```

## c. Controller

### Argument Resolver

```
EXPLORER ... ArgumentResolver.php X
CONTROLLER
> ArgumentResolver
ArgumentResolver.php
ArgumentResolverInt...
ArgumentValueResolv...
ContainerControllerR...
ControllerReference.p...
ControllerResolver.php
ControllerResolverInt...
ErrorController.php
TraceableArgumentRe...
TraceableControllerRe...

ArgumentResolver.php
1 <?php
2
3 /*
4  * This file is part of the Symfony package.
5  *
6  * (c) Fabien Potencier <fabien@symfony.com>
7  *
8  * For the full copyright and license information, please view the LICENSE
9  * file that was distributed with this source code.
10 */
11
12 namespace Symfony\Component\HttpKernel\Controller;
13
14 use Symfony\Component\HttpFoundation\Request;
15 use Symfony\Component\HttpKernel\Controller\ArgumentResolver\DefaultValueResolver;
16 use Symfony\Component\HttpKernel\Controller\ArgumentResolver\RequestAttributeValueResolver;
17 use Symfony\Component\HttpKernel\Controller\ArgumentResolver\RequestValueResolver;
18 use Symfony\Component\HttpKernel\Controller\ArgumentResolver\SessionValueResolver;
19 use Symfony\Component\HttpKernel\Controller\ArgumentResolver\VariadicValueResolver;
20 use Symfony\Component\HttpKernel\ControllerMetadata\ArgumentMetadataFactory;
21 use Symfony\Component\HttpKernel\ControllerMetadata\ArgumentMetadataFactoryInterface;
22
```

```
ArgumentResolver.php X
ArgumentResolver.php
23 /**
24  * Responsible for resolving the arguments passed to an action.
25  *
26  * @author Iltar van der Berg <kjarli@gmail.com>
27  */
28 final class ArgumentResolver implements ArgumentResolverInterface
29 {
30     private $argumentMetadataFactory;
31     private iterable $argumentValueResolvers;
32
33     /**
34      * @param iterable<mixed, ArgumentValueResolverInterface> $argumentValueResolvers
35      */
36     public function __construct(ArgumentMetadataFactoryInterface $argumentMetadataFactory = null, iterable $argumentValueResolvers = null)
37     {
38         $this->argumentMetadataFactory = $argumentMetadataFactory ?? new ArgumentMetadataFactory();
39         $this->argumentValueResolvers = $argumentValueResolvers ?: self::getDefaultArgumentValueResolvers();
40     }
41
42     /**
43      * @inheritdoc
44      */
45     public function getArguments(Request $request, callable $controller): array
46     {
47         $arguments = [];
48
49         foreach ($this->argumentMetadataFactory->createArgumentMetadata($controller) as $metadata) {
50             foreach ($this->argumentValueResolvers as $resolver) {
51                 if (!$resolver->supports($request, $metadata)) {
52                     continue;
53                 }
54
55                 $resolved = $resolver->resolve($request, $metadata);
56
57                 $atLeastOne = false;
58                 foreach ($resolved as $append) {
59                     $atLeastOne = true;
60                     $arguments[] = $append;
61                 }
62
63                 if (!$atLeastOne) {
64                     throw new \InvalidArgumentException(sprintf('%s::resolve()' must yield at least one value.', get_debug_backtrace_offset() + 1));
65                 }
66
67                 // continue to the next controller argument
68                 continue 2;
69             }
70
71             $representative = $controller;
72
73             if (\is_array($representative)) {
74                 $representative = sprintf('%s::%s()', \get_class($representative[0]), $representative[1]);
75             } elseif (\is_object($representative)) {
76                 $representative = \get_class($representative);
77             }
78
79             $arguments[] = $representative;
80         }
81
82         return $arguments;
83     }
84
85     /**
86      * Returns the default argument value resolvers.
87      */
88     public static function getDefaultArgumentValueResolvers(): iterable
89     {
90         return [
91             new DefaultValueResolver(),
92             new RequestAttributeValueResolver(),
93             new RequestValueResolver(),
94             new SessionValueResolver(),
95             new VariadicValueResolver(),
96         ];
97     }
98 }
```

```
ArgumentResolver.php X
ArgumentResolver.php
46 {
47     $arguments = [];
48
49     foreach ($this->argumentMetadataFactory->createArgumentMetadata($controller) as $metadata) {
50         foreach ($this->argumentValueResolvers as $resolver) {
51             if (!$resolver->supports($request, $metadata)) {
52                 continue;
53             }
54
55             $resolved = $resolver->resolve($request, $metadata);
56
57             $atLeastOne = false;
58             foreach ($resolved as $append) {
59                 $atLeastOne = true;
60                 $arguments[] = $append;
61             }
62
63             if (!$atLeastOne) {
64                 throw new \InvalidArgumentException(sprintf('%s::resolve()' must yield at least one value.', get_debug_backtrace_offset() + 1));
65             }
66
67             // continue to the next controller argument
68             continue 2;
69         }
70
71         $representative = $controller;
72
73         if (\is_array($representative)) {
74             $representative = sprintf('%s::%s()', \get_class($representative[0]), $representative[1]);
75         } elseif (\is_object($representative)) {
76             $representative = \get_class($representative);
77         }
78
79         $arguments[] = $representative;
80     }
81
82     return $arguments;
83 }
```

```

79         throw new \RuntimeException(sprintf('Controller "%s" requires that you provide a value for the "%s" argument.
80     });
81
82     return $arguments;
83 }
84
85 /**
86  * @return iterable<int, ArgumentValueResolverInterface>
87  */
88 public static function getDefaultArgumentValueResolvers(): iterable
89 {
90     return [
91         new RequestAttributeValueResolver(),
92         new RequestValueResolver(),
93         new SessionValueResolver(),
94         new DefaultValueResolver(),
95         new VariadicValueResolver(),
96     ];
97 }
98
99 }

```

## Argument Resolver Interface

```

ArgumentResolverInterface.php X
ArgumentResolverInterface.php
2
3 /**
4  * This file is part of the Symfony package.
5  *
6  * (c) Fabien Potencier <fabien@symfony.com>
7  *
8  * For the full copyright and license information, please view the LICENSE
9  * file that was distributed with this source code.
10 */
11
12 namespace Symfony\Component\HttpKernel\Controller;
13
14 use Symfony\Component\HttpFoundation\Request;
15
16 /**
17  * An ArgumentResolverInterface instance knows how to determine the
18  * arguments for a specific action.
19  *
20  * @author Fabien Potencier <fabien@symfony.com>
21  */
22 interface ArgumentResolverInterface
23 {
24     /**
25      * Returns the arguments to pass to the controller.
26      *
27      * @throws \RuntimeException When no value could be provided for a required argument
28      */
29     public function getArguments(Request $request, callable $controller): array;
30 }
31

```

## Argument Value Resolver Interface

```

ArgumentValueResolverInterface.php X
ArgumentValueResolverInterface.php
1 <?php
2
3 /**
4  * This file is part of the Symfony package.
5  *
6  * (c) Fabien Potencier <fabien@symfony.com>
7  *
8  * For the full copyright and license information, please view the LICENSE
9  * file that was distributed with this source code.
10 */
11
12 namespace Symfony\Component\HttpKernel\Controller;
13
14 use Symfony\Component\HttpFoundation\Request;
15 use Symfony\Component\HttpKernel\ControllerMetadata\ArgumentMetadata;
16
17 /**
18  * Responsible for resolving the value of an argument based on its metadata.
19  *
20  * @author Iltar van der Berg <kjarli@gmail.com>
21  */
22 interface ArgumentValueResolverInterface
23 {
24     /**
25      * Whether this resolver can resolve the value for the given ArgumentMetadata.
26      */
27     public function supports(Request $request, ArgumentMetadata $argument): bool;
28
29     /**
30      * Returns the possible value(s).
31      */
32     public function resolve(Request $request, ArgumentMetadata $argument): iterable;
33 }

```

## Controller Resolver

```
ControllerResolver.php X
ControllerResolver.php
1 <?php
2
3 /**
4  * This file is part of the Symfony package.
5  *
6  * (c) Fabien Potencier <fabien@symfony.com>
7  *
8  * For the full copyright and license information, please view the LICENSE
9  * file that was distributed with this source code.
10 */
11
12 namespace Symfony\Component\HttpKernel\Controller;
13
14 use Psr\Log\LoggerInterface;
15 use Symfony\Component\HttpFoundation\Request;
16
17 /**
18  * This implementation uses the '_controller' request attribute to determine
19  * the controller to execute.
20  *
21  * @author Fabien Potencier <fabien@symfony.com>
22  * @author Tobias Schultze <http://tobias.de>
23  */
24 class ControllerResolver implements ControllerResolverInterface
25 {
26     private $logger;
27
28     public function __construct(LoggerInterface $logger = null)
29     {
30         $this->logger = $logger;
31     }
32 }
```

```
ControllerResolver.php X
ControllerResolver.php
33 /**
34  * {@inheritdoc}
35  */
36 public function getController(Request $request): callable|false
37 {
38     if (!$controller = $request->attributes->get('_controller')) {
39         if (null !== $this->logger) {
40             $this->logger->warning('Unable to look for the controller as the "_controller" parameter is missing.');
```

```

66     if (\is_object($controller)) {
67         if (!\is_callable($controller)) {
68             throw new \InvalidArgumentException(sprintf('The controller for URI "%s" is not callable: ', $request->getPathInfo()));
69         }
70
71         return $controller;
72     }
73
74     if (\function_exists($controller)) {
75         return $controller;
76     }
77
78     try {
79         $callable = $this->createController($controller);
80     } catch (\InvalidArgumentException $e) {
81         throw new \InvalidArgumentException(sprintf('The controller for URI "%s" is not callable: ', $request->getPathInfo()));
82     }
83
84     if (!\is_callable($callable)) {
85         throw new \InvalidArgumentException(sprintf('The controller for URI "%s" is not callable: ', $request->getPathInfo()));
86     }
87
88     return $callable;
89 }
90

```

ControllerResolver.php X

ControllerResolver.php

```

92     * Returns a callable for the given controller.
93     *
94     * @throws \InvalidArgumentException When the controller cannot be created
95     */
96     protected function createController(string $controller): callable
97     {
98         if (!str_contains($controller, '::')) {
99             $controller = $this->instantiateController($controller);
100
101             if (!\is_callable($controller)) {
102                 throw new \InvalidArgumentException($this->getControllerError($controller));
103             }
104
105             return $controller;
106         }
107
108         [$class, $method] = explode('::', $controller, 2);
109
110         try {
111             $controller = [$this->instantiateController($class), $method];
112         } catch (\Error|\LogicException $e) {
113             try {
114                 if ((new \ReflectionMethod($class, $method))->isStatic()) {
115                     return $class.'::'.$method;
116                 }
117             } catch (\ReflectionException $reflectionException) {
118                 throw $e;
119             }
120
121             throw $e;
122         }
123     }
124

```



```

ControllerResolver.php X
ControllerResolver.php
124     if (!\is_callable($controller)) {
125         throw new \InvalidArgumentException($this->getControllerError($controller));
126     }
127
128     return $controller;
129 }
130
131 /**
132  * Returns an instantiated controller.
133  */
134 protected function instantiateController(string $class): object
135 {
136     return new $class();
137 }
138
139 private function getControllerError(mixed $callable): string
140 {
141     if (\is_string($callable)) {
142         if (str_contains($callable, '::')) {
143             $callable = explode('::', $callable, 2);
144         } else {
145             return sprintf('Function "%s" does not exist.', $callable);
146         }
147     }
148
149     if (\is_object($callable)) {
150         $availableMethods = $this->getClassMethodsWithoutMagicMethods($callable);
151         $alternativeMsg = $availableMethods ? sprintf(' or use one of the available methods: "%s"', implode('', $availableMethods)) : '';
152
153         return sprintf('Controller class "%s" cannot be called without a method name. You need to implement "__invoke"%s', $callable, $alternativeMsg);
154     }
155 }

```

```

ControllerResolver.php X
ControllerResolver.php
156     if (!\is_array($callable)) {
157         return sprintf('Invalid type for controller given, expected string, array or object, got "%s".', get_debug_type($callable));
158     }
159
160     if (isset($callable[0]) || isset($callable[1]) || 2 !== \count($callable)) {
161         return 'Invalid array callable, expected [controller, method].';
162     }
163
164     [$controller, $method] = $callable;
165
166     if (\is_string($controller) && !class_exists($controller)) {
167         return sprintf('Class "%s" does not exist.', $controller);
168     }
169
170     $className = \is_object($controller) ? get_debug_type($controller) : $controller;
171
172     if (method_exists($controller, $method)) {
173         return sprintf('Method "%s" on class "%s" should be public and non-abstract.', $method, $className);
174     }
175
176     $collection = $this->getClassMethodsWithoutMagicMethods($controller);
177
178     $alternatives = [];
179
180     foreach ($collection as $item) {
181         $lev = levenshtein($method, $item);
182
183         if ($lev <= \strlen($method) / 3 || str_contains($item, $method)) {
184             $alternatives[] = $item;
185         }
186     }

```

```

ControllerResolver.php X
ControllerResolver.php
187
188     asort($alternatives);
189
190     $message = sprintf('Expected method "%s" on class "%s"', $method, $className);
191
192     if (\count($alternatives) > 0) {
193         $message .= sprintf(', did you mean "%s"?', implode('"', '"', $alternatives));
194     } else {
195         $message .= sprintf('. Available methods: "%s".', implode('"', '"', $collection));
196     }
197
198     return $message;
199 }
200
201 private function getClassMethodsWithoutMagicMethods($classOrObject): array
202 {
203     $methods = get_class_methods($classOrObject);
204
205     return array_filter($methods, function (string $method) {
206         return 0 !== strpos($method, '_');
207     });
208 }
209 }
210

```

## Controller Reference

```

ControllerReference.php X
ControllerReference.php
12 namespace Symfony\Component\HttpKernel\Controller;
13
14 use Symfony\Component\HttpKernel\Fragment\FragmentRendererInterface;
15
16 /**
17  * Acts as a marker and a data holder for a Controller.
18  *
19  * Some methods in Symfony accept both a URI (as a string) or a controller as
20  * an argument. In the latter case, instead of passing an array representing
21  * the controller, you can use an instance of this class.
22  *
23  * @author Fabien Potencier <fabien@symfony.com>
24  *
25  * @see FragmentRendererInterface
26  */
27 class ControllerReference
28 {
29     public $controller;
30     public $attributes = [];
31     public $query = [];
32
33     /**
34      * @param string $controller The controller name
35      * @param array $attributes An array of parameters to add to the Request attributes
36      * @param array $query An array of parameters to add to the Request query string
37      */
38     public function __construct(string $controller, array $attributes = [], array $query = [])
39     {
40         $this->controller = $controller;
41         $this->attributes = $attributes;
42         $this->query = $query;
43     }

```

## Container Controller Resolver

```
ContainerControllerResolver.php X
ContainerControllerResolver.php
12 namespace Symfony\Component\HttpKernel\Controller;
13
14 use Psr\Container\ContainerInterface;
15 use Psr\Log\LoggerInterface;
16 use Symfony\Component\DependencyInjection\Container;
17
18 /**
19  * A controller resolver searching for a controller in a psr-11 container when using the "service::method" notation.
20  *
21  * @author Fabien Potencier <fabien@symfony.com>
22  * @author Maxime Steinhauser <maxime.steinhauser@gmail.com>
23  */
24 class ContainerControllerResolver extends ControllerResolver
25 {
26     protected $container;
27
28     public function __construct(ContainerInterface $container, LoggerInterface $logger = null)
29     {
30         $this->container = $container;
31
32         parent::__construct($logger);
33     }
34 }
```

```
ContainerControllerResolver.php X
ContainerControllerResolver.php
35 /**
36  * {@inheritdoc}
37  */
38 protected function instantiateController(string $class): object
39 {
40     $class = ltrim($class, '\\');
41
42     if ($this->container->has($class)) {
43         return $this->container->get($class);
44     }
45
46     try {
47         return parent::instantiateController($class);
48     } catch (\Error $e) {
49     }
50
51     $this->throwExceptionIfControllerWasRemoved($class, $e);
52
53     if ($e instanceof \ArgumentCountError) {
54         throw new \InvalidArgumentException(sprintf('Controller "%s" has required constructor arguments and does not exist', $class), $e->getCode(), $e->getMessage());
55     }
56
57     throw new \InvalidArgumentException(sprintf('Controller "%s" does neither exist as service nor as class.', $class), $e->getCode(), $e->getMessage());
58 }
59
60 private function throwExceptionIfControllerWasRemoved(string $controller, \Throwable $previous)
61 {
62     if ($this->container instanceof Container && isset($this->container->getRemovedIds())[$controller]) {
63         throw new \InvalidArgumentException(sprintf('Controller "%s" cannot be fetched from the container because it is removed', $controller), $previous->getCode(), $previous->getMessage());
64     }
65 }
66 }
```

## Controller Resolver Interface

```
ControllerResolverInterface.php X
ControllerResolverInterface.php
10  */
11
12  namespace Symfony\Component\HttpKernel\Controller;
13
14  use Symfony\Component\HttpFoundation\Request;
15
16  /**
17   * A ControllerResolverInterface implementation knows how to determine the
18   * controller to execute based on a Request object.
19   *
20   * A Controller can be any valid PHP callable.
21   *
22   * @author Fabien Potencier <fabien@symfony.com>
23   */
24  interface ControllerResolverInterface
25  {
26      /**
27       * Returns the Controller instance associated with a Request.
28       *
29       * As several resolvers can exist for a single application, a resolver must
30       * return false when it is not able to determine the controller.
31       *
32       * The resolver must only throw an exception when it should be able to load a
33       * controller but cannot because of some errors made by the developer.
34       *
35       * @return callable|false A PHP callable representing the Controller,
36       *                        or false if this resolver is not able to determine the controller
37       *
38       * @throws \LogicException If a controller was found based on the request but it is not callable
39       */
40      public function getController(Request $request): callable|false;
41  }
```

## Error Controller

```
ErrorController.php X
ErrorController.php
12  namespace Symfony\Component\HttpKernel\Controller;
13
14  use Symfony\Component\ErrorHandler\ErrorRenderer\ErrorRendererInterface;
15  use Symfony\Component\HttpFoundation\Request;
16  use Symfony\Component\HttpFoundation\Response;
17  use Symfony\Component\HttpKernel\Exception\HttpException;
18  use Symfony\Component\HttpKernel\HttpKernelInterface;
19
20  /**
21   * Renders error or exception pages from a given FlattenException.
22   *
23   * @author Yonel Ceruto <yonelceruto@gmail.com>
24   * @author Matthias Pigulla <mp@webfactory.de>
25   */
26  class ErrorController
27  {
28      private $kernel;
29      private string|object|array|null $controller;
30      private $errorRenderer;
31
32      public function __construct(HttpKernelInterface $kernel, string|object|array|null $controller, ErrorRendererInterface
33      {
34          $this->kernel = $kernel;
35          $this->controller = $controller;
36          $this->errorRenderer = $errorRenderer;
37      }
38
39      public function __invoke(\Throwable $exception): Response
40      {
41          $exception = $this->errorRenderer->render($exception);
42
43          return new Response($exception->getAsString(), $exception->getStatusCode(), $exception->getHeaders());
44      }
45  }
```

```

ErrorController.php X
ErrorController.php
42
43     return new Response($exception->getAsString(), $exception->getStatusCode(), $exception->getHeaders());
44 }
45
46 public function preview(Request $request, int $code): Response
47 {
48     /**
49      * This Request mimics the parameters set by
50      * \Symfony\Component\HttpKernel\EventListener\ErrorListener::duplicateRequest, with
51      * the additional "showException" flag.
52      */
53     $subRequest = $request->duplicate(null, null, [
54         '_controller' => $this->controller,
55         'exception' => new HttpException($code, 'This is a sample exception.'),
56         'logger' => null,
57         'showException' => false,
58     ]);
59
60     return $this->kernel->handle($subRequest, HttpKernelInterface::SUB_REQUEST);
61 }
62
63

```

## Traceable Argument Resolver

```

TraceableArgumentResolver.php X
TraceableArgumentResolver.php
12 namespace Symfony\Component\HttpKernel\Controller;
13
14 use Symfony\Component\HttpFoundation\Request;
15 use Symfony\Component\Stopwatch\Stopwatch;
16
17 /**
18  * @author Fabien Potencier <fabien@symfony.com>
19  */
20 class TraceableArgumentResolver implements ArgumentResolverInterface
21 {
22     private $resolver;
23     private $stopwatch;
24
25     public function __construct(ArgumentResolverInterface $resolver, Stopwatch $stopwatch)
26     {
27         $this->resolver = $resolver;
28         $this->stopwatch = $stopwatch;
29     }
30
31     /**
32      * {@inheritdoc}
33      */
34     public function getArguments(Request $request, callable $controller): array
35     {
36         $e = $this->stopwatch->start('controller.get_arguments');
37
38         $ret = $this->resolver->getArguments($request, $controller);
39
40         $e->stop();
41
42         return $ret;
43     }

```

## Traceable Controller Resolver

```

TraceableControllerResolver.php
TraceableControllerResolver.php
12 namespace Symfony\Component\HttpKernel\Controller;
13
14 use Symfony\Component\HttpFoundation\Request;
15 use Symfony\Component\Stopwatch\Stopwatch;
16
17 /**
18  * @author Fabien Potencier <fabien@symfony.com>
19  */
20 class TraceableControllerResolver implements ControllerResolverInterface
21 {
22     private $resolver;
23     private $stopwatch;
24
25     public function __construct(ControllerResolverInterface $resolver, Stopwatch $stopwatch)
26     {
27         $this->resolver = $resolver;
28         $this->stopwatch = $stopwatch;
29     }
30
31     /**
32      * {@inheritdoc}
33      */
34     public function getController(Request $request): callable|false
35     {
36         $e = $this->stopwatch->start('controller.get_callable');
37
38         $ret = $this->resolver->getController($request);
39
40         $e->stop();
41
42         return $ret;
43     }

```

### 3.5 Tabel Tugas Anggota

Nama	NIM	Tugas yang dilakukan
Rizka Nurul Septiani Hakim	20051397026	<ol style="list-style-type: none"> <li>1. Mengembangkan halaman Info</li> <li>2. Mengembangkan halaman Appoinment</li> <li>3. Membantu mengedit Laporan</li> </ol>
Karina Irna Della	20051397030	<ol style="list-style-type: none"> <li>1. Mengembang halaman About Us</li> <li>2. Menyusun Laporan</li> <li>3. Membantu mengedit Aplikasi</li> </ol>
Anggelina Kismasari	20051397034	<ol style="list-style-type: none"> <li>1. Mengembangkan halaman Login dan Registrasi, Contac Us</li> <li>2. Mengembangkan database</li> <li>3. Mengembangkan halaman Admin</li> </ol>
Nisa Amalia	20051397038	<ol style="list-style-type: none"> <li>1. Mengembangkan halaman KCU BPJS</li> <li>2. Mengembangkan halaman FasKes</li> <li>3. Membantu mengedit Aplikasi</li> </ol>

## **BAB IV**

### **PENUTUP**

#### **4.1 Kesimpulan**

BPJS Kesehatan (Badan Penyelenggara Jaminan Sosial Kesehatan) adalah sebuah badan hukum publik yang mengatur jaminan kesehatan bagi seluruh warga Indonesia. Sedangkan Sistem Informasi Manajemen adalah kumpulan dari interaksi sistem-sistem informasi yang berwenang dalam mengumpulkan dan mengolah data guna menyediakan informasi yang bermanfaat bagi semua tingkatan manajemen di dalam kegiatan perencanaan dan pengendalian.

Pembuatan sistem ini menggunakan framework Laravel 8. Dengan menggunakan Laravel dapat lebih memudahkan dalam pembuatan aplikasi web. Selain lebih rapi dalam penyusunan source code di dalam framework Laravel telah banyak fitur-fitur yang memudahkan developer untuk membangun sebuah aplikasi web.

Oleh karena itu, dengan adanya sistem informasi manajemen asuransi BPJS Kesehatan dapat membantu pegawai di sektor umum dan SDM dalam memantau informasi atau berita mengenai BPJS Kesehatan.

#### **4.2 Saran**

1. Pengembangan aplikasi website lebih lanjut agar pengguna bisa melakukan pendaftaran dan pembayaran iuran BPJS Kesehatan.
2. Pengembangan fitur verifikasi email agar pengguna dapat memastikan data yang dipakai benar miliknya dan merasa lebih aman dalam menggunakan website.