



POLITEKNIK NEGERI BANYUWANGI

Laporan Final Project: CinemaPro

Mobile Application Development
Semester Ganjil 2024/2025

Disusun Oleh Kelompok 07:

Cheryl Aurellya Bangun Jaya	362458302037	Backend Engineer & Data Seeder
Dino Febiyan	362458302043	Frontend Engineer (Home Module)
Rusydi Jabir Al Awfa	362458302044	Frontend Engineer (Seat Matrix Module)
Mohammad Faisal	NIM004	Logic Controller
Semua		QA Lead, Auth & Profile

Tahun 2025

BAB 1

Pendahuluan & Pembagian Kerja

1.1 Deskripsi Aplikasi

CinemaPro adalah aplikasi pemesanan tiket bioskop berbasis Flutter dengan integrasi Firebase, mendukung multi-platform, dan memiliki sistem autentikasi yang aman.

Tabel berikut menunjukkan pembagian tugas tim:

Anggota	Peran	Tanggung Jawab
Anggota 1	Backend Engineer & Data Seeder	Cheryl Aurellya Bangun Jaya
Anggota 2	Frontend Engineer (Home Module)	Dino Febiyan
Anggota 3	Frontend Engineer (Seat Matrix Module)	Rusydi Jabir Al Awfa
Anggota 4	Logic Controller (The Brain)	Mohammad Faisal
Anggota 5	QA Lead, Auth & Profile	Semua anggota

BAB 2

Bukti Keaslian Kode (Strict Mode)

Sesuai instruksi ujian untuk mencegah penggunaan code generator otomatis

2.1 Watermark Code

Berikut adalah bukti penggunaan suffix inisial pada variabel Widget dan Fungsi:

```
String _formatPrice_dino(int price) {  
    final priceStr = price.toString();  
    final buffer = StringBuffer('Rp ');  
    for (int i = 0; i < priceStr.length; i++) {  
        if (i > 0 && (priceStr.length - i) % 3 == 0) {  
            buffer.write('.');  
        }  
        buffer.write(priceStr[i]);  
    }  
    return buffer.toString();  
}
```

Penjelasan: Fungsi `_formatPrice_dino` menunjukkan kode dibuat oleh Dino (UI Engineer)

2.2 Logic Trap

Implementasi logika bisnis “The ”Long Title” Tax”

```
// Contoh snippet kode (Logic Trap)  
void hitungDiskon_dani(String nim) {  
    int lastDigit = int.parse(nim.characters.last);  
    if (lastDigit % 2 != 0) {  
        // Logika Ganjil  
        print("Diskon 5%");  
    } else {  
        // Logika Genap  
        print("Gratis Ongkir");  
    }  
}
```

TEMPEL KODEMU NDEK KENE @ISAL

Implementasi logika bisnis unik “Odd/Even Seat Rule:”

```
// Contoh snippet kode (Logic Trap)  
void hitungDiskon_dani(String nim) {  
    int lastDigit = int.parse(nim.characters.last);  
    if (lastDigit % 2 != 0) {  
        // Logika Ganjil  
        print("Diskon 5%");  
    } else {
```

```
// Logika Genap  
print("Gratis Ongkir");  
}  
}
```

TEMPEL KODEMU NDEK KENE @ISAL

BAB 3

Arsitektur Backend

Dikerjakan oleh Backend Engineer & Data Seeder.

3.1 Struktur Database (Firestore)

Menggunakan 3 Collection utama: Users, Movies, dan Bookings. Berikut Struktur field sebagai berikut:

A. Collection: Users

Menyimpan data profil pengguna

- uid: String (Primary key)
- email: String (Email pengguna, Validasi domain student.univ.ac.id)
- username: String (Nama tampilan pengguna)
- balance: Number(int) (Saldo awal, Default: 0)
- password: String (Password pengguna)
- created_at: Timestamp (Waktu registrasi akun)

B. Collection: Movies

Menyimpan data film

- movie_id: String (Primary key)
- title: String (Judul film)
- poster_url: String (Gambar poster film)
- base_price: Number(int) (Harga dasar tiket)
- rating: Number(Double) (Rating film, skala 1.0-5.0)
- duration: Number(int) (Durasi film dalam menit)

C. Collection: Bookings

Menyimpan riwayat transaksi pembelian tiket

- booking_id: String (Primary key)
- user_id: String (Foreign key ke collection Users)
- movie_title: String (Judul film yang di pesan)
- seats: Array(string) (Daftar kursi yang dipilih, contoh: A1, A2)
- total_price: Number(int) (Harga final setelah perhitungan pajak dan diskon)
- booking_date: Timestamp (Waktu transaksi dilakukan)

Screenshot Collection Users:

Screenshot Collection Movies:

Screenshot Collection Bookings:

3.2 Data Seeding

Bukti seeding minimal 10 produk dummy ke Firebase. Untuk pengujian Logic Trap, maka ada seeding data sebanyak 10 film:

1. Strategi Judul Pendek (kurang dari atau sama dengan 10 Karakter):

Film: Up, Coco, Frozen, Venom, Dilan 1990. Ini untuk menguji bahwa harga tiket tidak dikenakan pajak tambahan.

2. Strategi Judul Panjang (lebih dari 10 Karakter):

Film: Spider-Man: No Way Home, Avengers: Endgame, Pengabdi Setan 2: Communion, Demon Slayer: Kimetsu no Yaiba - The Movie: Infinity Castle, KKN di Desa Penari. Ini untuk logika yang menambahkan harga sebesar Rp. 2.500.

Berikut kode untuk otomatis seeding data ke firebase (file seedMovies function):

The screenshot shows the Google Cloud Platform Firestore console. At the top, there's a header with 'CinemPro' and 'Cloud Firestore >'. Below it, a sub-header says 'Database' with 'Add Database' and 'Ask Gemini about the core concepts to use Firestore'. There are tabs for 'Data', 'Rules', 'Indexes', 'Disaster Recovery', 'Usage', and 'Extensions'. A note 'Protect your Cloud Firestore resources from abuse, such as billing fraud or phishing' is present, along with 'Configure App Check' and 'Panel view' and 'Query builder' buttons.

The main area shows a hierarchical tree structure of collections and documents. On the left, under 'movies', there are 10 documents labeled 'mv01' through 'mv10'. Each document has a detailed view on the right, showing fields like 'base_price: 30000', 'duration: 96', 'movie_id: "mv01"', 'poster_url: "https://upload.wikimedia.org/wikipedia/en/0/06/Up_%282009%29.jpg"', 'rating: 4.8', and 'title: "Up"'. The interface includes a 'More in Google Cloud' button at the top right of the document details.

Proses seeding berhasil memasukkan 10 data film ke dalam Firebase Firestore dengan struktur collection Movies yang ditentukan.

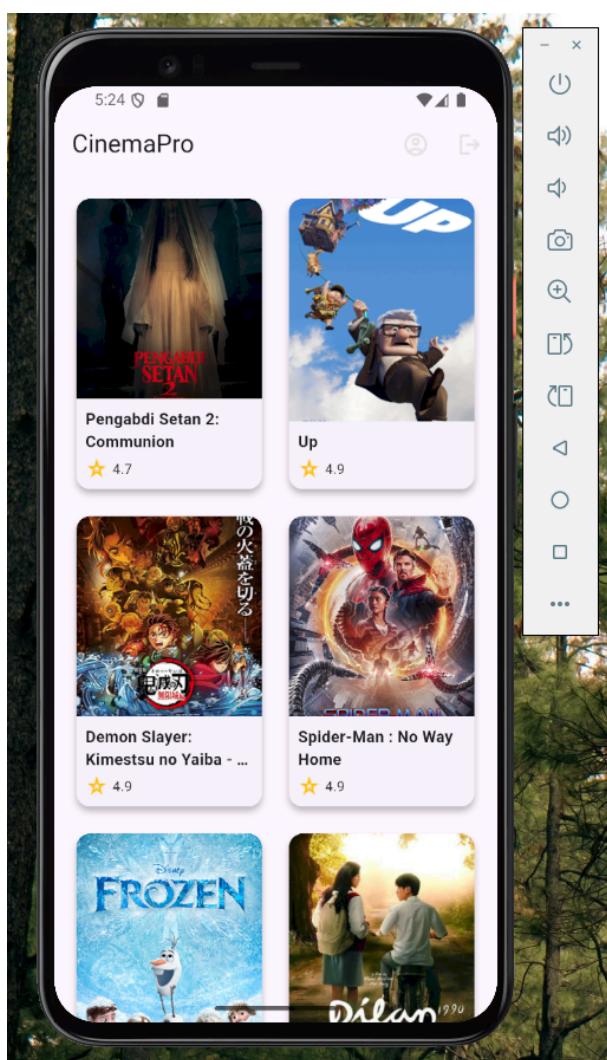
BAB 4

Implementasi UI Home & Detail

Dikerjakan oleh Frontend Engineer (Home Module)

4.1 UI Home

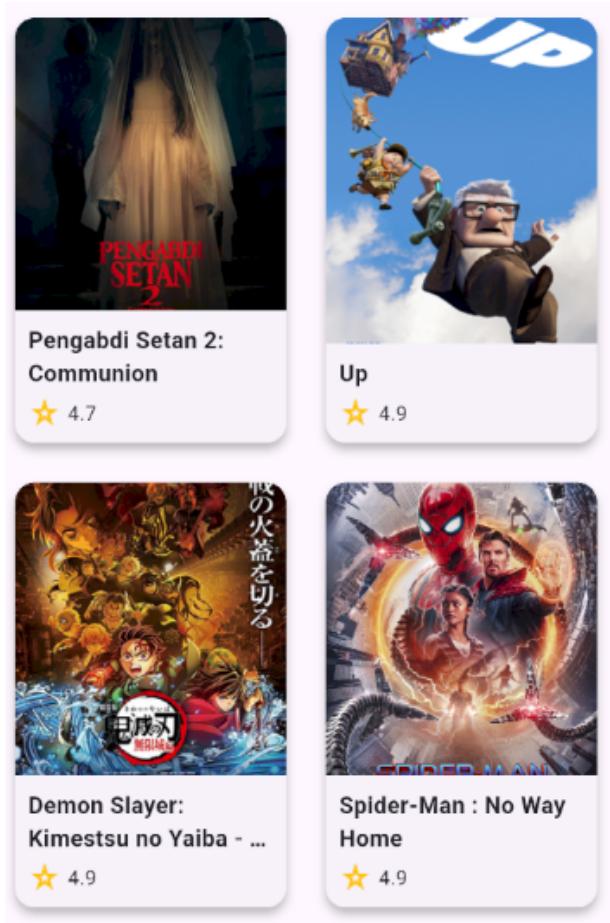
Halaman UI Home ini dirancang untuk menampilkan koleksi film dalam format grid beserta dengan judul nya. Dengan halaman ini, diharapkan pengguna akan bisa melihat berbagai film secara bersamaan dengan susunan yang rapi dan mudah untuk diakses. Ketika sebuah poster dipilih, aplikasi akan menggunakan konsep Hero widget agar muncul animasi transisi ke detail page, sehingga perpindahan menuju halaman detail film diharapkan akan lebih bagus dan menarik.



4.1.1 Membuat tampilan Grid Film (SliverGridDelegate).

Pada bagian ini saya membuat tampilan daftar film dalam bentuk grid dengan memanfaatkan `GridView.builder`. Untuk mengatur susunan grid, saya menggunakan `SliverGridDelegateWithMaxCrossAxisExtent`, di mana saya menentukan lebar maksimal setiap item sebesar 200 pixel. Dengan cara ini, tampilan home akan menjadi lebih bagus

karena jumlah kolom akan menyesuaikan ukuran layar. Selain itu, saya juga menambahkan jarak antar item menggunakan crossAxisSpacing dan mainAxisSpacing, serta mengatur rasio tinggi-lebar dengan childAspectRatio agar poster film lebih rapi. Data film saya ambil dari Firebase melalui snapshot.data!.docs kemudian saya taruh ke model MovieModelCheryl. Setiap item grid saya bungkus dengan GestureDetector sehingga ketika pengguna menekan sebuah poster, aplikasi akan diarahkan ke halaman detail film (DetailPage_dino).



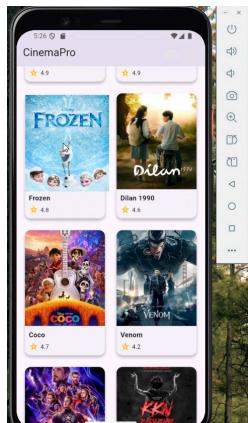
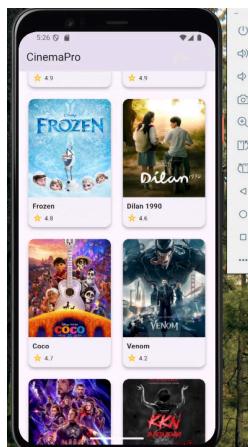
```
return GridView.builder(  
    padding: const EdgeInsets.all(16.0),  
    gridDelegate: SliverGridDelegateWithMaxCrossAxisExtent(  
        maxCrossAxisExtent: 200,  
        crossAxisSpacing: 16.0,  
        mainAxisSpacing: 16.0,  
        childAspectRatio: 0.65,  
    ),  
    itemCount: snapshot.data!.docs.length,  
    itemBuilder: (context, index) {  
        final doc = snapshot.data!.docs[index];  
        final movieData = doc.data() as Map<String, dynamic>;  
        final movie = MovieModelCheryl.fromMap_Cheryl(movieData, doc.id);  
  
        return GestureDetector(  
            onTap: () {  
                Navigator.push(  
                    context,  
                    MaterialPageRoute(builder: (context) =>  
                        DetailPage_dino(  
                            movie: movie,  
                            id: doc.id,  
                        ),  
                );  
            },  
            child: Container(  
                width: 200,  
                height: 300,  
                decoration: BoxDecoration(  
                    image: DecorationImage(  
                        fit: BoxFit.cover,  
                        image: NetworkImage(movie.posterUrl),  
                    ),  
                ),  
            ),  
        );  
    },  
);
```

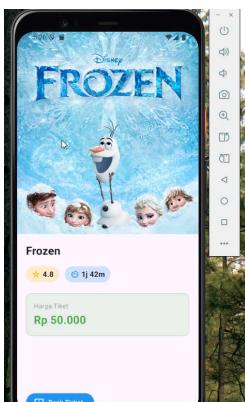
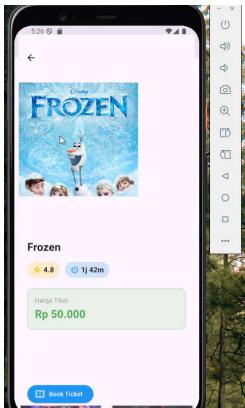
```

        context,
        MaterialPageRoute(
            builder: (context) => DetailPage_dino(movie: movie),
        ),
    );
},
);
},
);
);

```

4.1.2 Menggunakan widget Hero pada gambar poster agar ada animasi saat pindah ke detail. Untuk menampilkan poster film, saya menggunakan Image.network agar gambar bisa langsung diambil dari link URL. Poster tersebut saya bungkus dengan widget Hero yang memiliki tag unik berdasarkan ID film. Dengan cara ini, ketika pengguna menekan sebuah poster, Flutter otomatis menampilkan animasi transisi dari poster di grid menuju poster di halaman detail. Efek animasi ini akan membuat perpindahan menjadi lebih bagus. Selain itu, saya menambahkan ClipRRect agar memberikan efek sudut melengkung pada gambar, serta errorBuilder dan loadingBuilder ketika ada kondisi gambar rusak atau masih dalam proses loading.





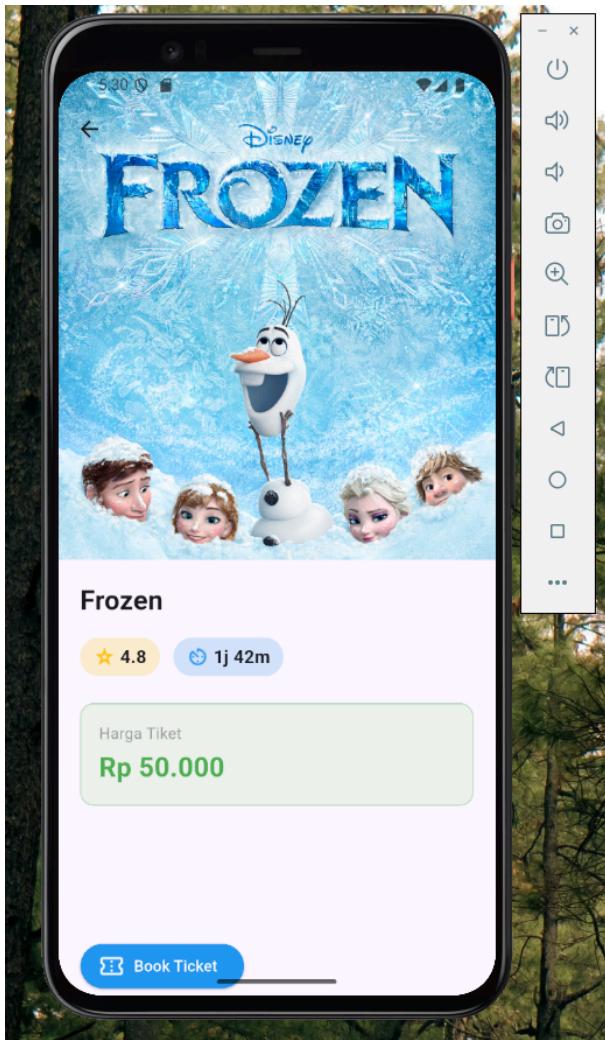
```
child: ClipRRect(
  borderRadius: const BorderRadius.vertical(
    top: Radius.circular(12),
  ),
  child: Hero(
    tag: movie.movieID,
    child: Image.network(
      movie.posterUrl,
      fit: BoxFit.cover,
      errorBuilder: (context, error, stackTrace) {
        return Container(
          color: Colors.grey[300],
          child: Center(
            child: Image.asset(
              'assets/icons/gambarRusak.png',

```

```
        width: 50,
        height: 50,
        fit: BoxFit.contain,
        color: Colors.grey,
      ),
    ),
  );
},
loadingBuilder: (context, child, loadingProgress) {
  if (loadingProgress == null) return child;
  return Center(
    child: CircularProgressIndicator(),
  );
},
),
),
);
};
```

4.2 UI Detail

Pada bagian UI Detail ini, saya membuat tampilan khusus yang berfungsi untuk menampilkan informasi lengkap mengenai film yang dipilih oleh pengguna. Halaman ini menjadi lanjutan dari UI Home, sehingga setelah pengguna menekan salah satu poster film, mereka akan diarahkan ke halaman detail. Tujuan utama dari UI Detail adalah memberikan informasi dari film, di mana pengguna bisa melihat detail film sekaligus melakukan tindakan pemesanan tiket dengan mudah.



4.2.1 Menampilkan info film.

Untuk menampilkan informasi film, saya memanfaatkan data yang sudah diambil dari Firebase dan kemudian ditampilkan dalam bentuk teks maupun gambar. Informasi yang saya tampilkan yaitu judul film, poster, serta harga tiket yang diformat menggunakan fungsi `formatPrice_dino`. Fungsi ini saya buat agar harga tiket ditampilkan dalam format rupiah dengan penulisan yang rapi, misalnya “Rp 25.000”. Poster film ditampilkan menggunakan `Image.network` sehingga gambar bisa langsung diambil dari URL, dan saya tambahkan `ClipRRect` untuk memberikan efek sudut melengkung. Selain itu, saya juga menambahkan `errorBuilder` dan `loadingBuilder` untuk menangani kondisi ketika gambar rusak atau masih dalam proses loading.



Frozen

★ 4.8

⌚ 1j 42m

Harga Tiket

Rp 50.000

```
String _formatPrice_dino(int price) {
    final priceStr = price.toString();
    final buffer = StringBuffer('Rp ');
    for (int i = 0; i < priceStr.length; i++) {
        if (i > 0 && (priceStr.length - i) % 3 == 0) {
            buffer.write('.');
        }
        buffer.write(priceStr[i]);
    }
    return buffer.toString();
}

Widget buildMovieInfo_dino(MovieModelCheryl movie) {
    return SingleChildScrollView(
        child: Column(
            mainAxisAlignment: MainAxisAlignment.start,
            children: [
                ClipRRect(
                    borderRadius: const BorderRadius.vertical(top: Radius.circular(16)),
                    child: Image.network(
```

```

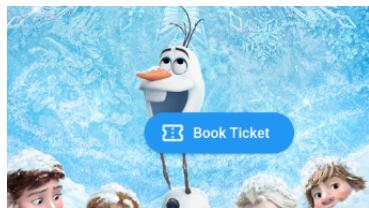
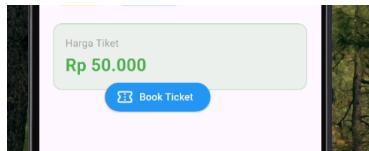
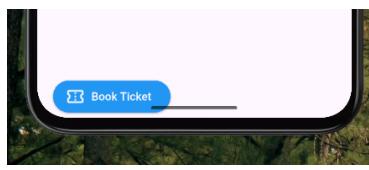
        movie.posterUrl,
        fit: BoxFit.cover,
        width: double.infinity,
        errorBuilder: (context, error, stackTrace) {
            return Container(
                height: 220,
                color: Colors.grey[300],
                child: Center(
                    child: Image.asset(
                        'assets/icons/gambarRusak.png',
                        width: 50,
                        height: 50,
                        color: Colors.grey,
                    ),
                ),
            );
        },
        loadingBuilder: (context, child, loadingProgress) {
            if (loadingProgress == null) return child;
            return const SizedBox(
                height: 220,
                child: Center(child: CircularProgressIndicator()),
            );
        },
    ),
),
const SizedBox(height: 16),
Text(
    movie.title,
    style: const TextStyle(
        fontSize: 22,
        fontWeight: FontWeight.bold,
    ),
),
const SizedBox(height: 8),
Text(
    _formatPrice_dino(movie.basePrice),
    style: const TextStyle(
        fontSize: 18,
        color: Colors.blueAccent,
        fontWeight: FontWeight.w600,
    ),
),
const SizedBox(height: 12),
if (movie.overview != null && movie.overview!.isNotEmpty)
Text(
    movie.overview!,
    style: const TextStyle(fontSize: 14, color: Colors.black87),
),
],
),

```

```
);  
}
```

4.2.2 Tombol "Book Ticket" melayang (Floating Action Button) di bawah.

Di bagian bawah halaman, saya menambahkan tombol Book Ticket yang melayang menggunakan Positioned dan ElevatedButton.icon. Tombol ini saya bungkus dengan GestureDetector sehingga posisinya bisa digeser oleh pengguna, sehingga bisa digeser sesuai kebutuhan. Saat tombol ditekan, sistem akan memeriksa apakah pengguna sudah login melalui Firebase Authentication. Jika belum login, akan muncul pesan peringatan menggunakan Snackbar. Namun jika sudah login, aplikasi akan menampilkan pesan "Sedang memproses pemesanan" dan langsung mengarahkan pengguna ke halaman pemilihan kursi (SeatMatrixJabir). Dengan adanya tombol ini, proses pemesanan tiket menjadi lebih praktis karena pengguna bisa langsung melakukan booking dari halaman detail film.



```
double posX = 16;  
double posY = 0;  
  
@override  
void initState() {  
    super.initState();  
    WidgetsBinding.instance.addPostFrameCallback((_) {  
        final screenHeight = MediaQuery.of(context).size.height;  
        setState(() {  
            posY = screenHeight - 50;  
        });  
    });  
}  
  
Positioned(  
    left: posX,  
    top: posY,  
    child: GestureDetector(  
        onPanUpdate: (details) {
```

```

        setState(() {
            posX += details.delta.dx;
            posY += details.delta.dy;
        });
    },
    child: ElevatedButton.icon(
        onPressed: () async {
            final user = FirebaseAuth.instance.currentUser;
            if (user == null) {
                ScaffoldMessenger.of(context).showSnackBar(
                    const SnackBar(content: Text('Harap Login Terlebih Dahulu!')),
                );
                return;
            }
            ScaffoldMessenger.of(context).showSnackBar(
                const SnackBar(content: Text('Sedang memproses pemesanan')),
            );
            try {
                Navigator.push(
                    context,
                    MaterialPageRoute(
                        builder: (context) => SeatMatrixJabir(
                            movieTitle: movie.title,
                            userId: user.uid,
                            totalPrice: movie.basePrice,
                        ),
                    ),
                );
            } catch (e) {
                if (context.mounted) {
                    ScaffoldMessenger.of(context).hideCurrentSnackBar();
                    ScaffoldMessenger.of(context).showSnackBar(
                        SnackBar(
                            content: Text('Gagal: $e'),
                            backgroundColor: Colors.red,
                        ),
                    );
                }
            }
        },
        icon: Image.asset(
            'assets/icons/ticket.png',
            width: 24,
            height: 24,
            color: Colors.white,
        ),
        label: const Text(
            'Book Ticket',
            style: TextStyle(color: Colors.white),
        ),
        style: ElevatedButton.styleFrom(
            backgroundColor: Colors.blue,

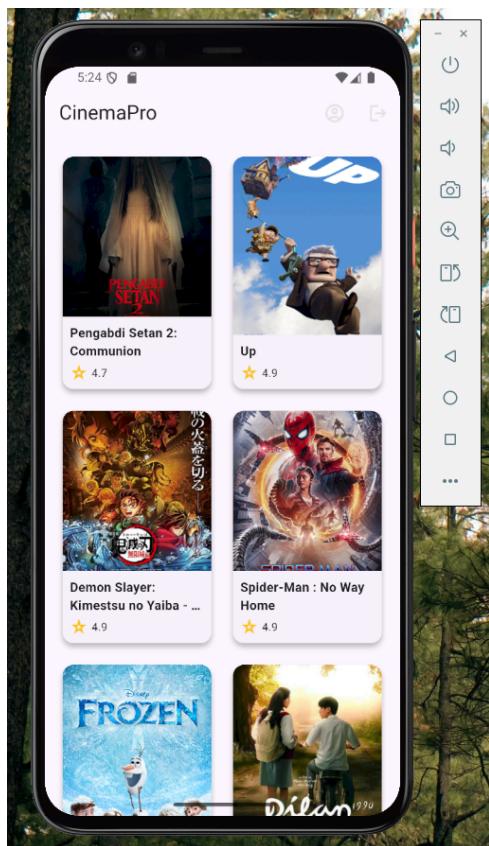
```

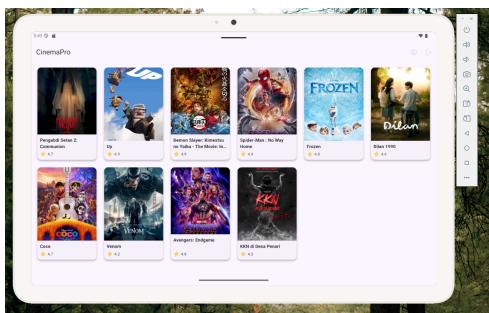
```
        elevation: 6,
        padding: const EdgeInsets.symmetric(horizontal: 16, vertical: 12),
        shape: RoundedRectangleBorder(
            borderRadius: BorderRadius.circular(12),
        ),
    ),
),
),
),
),
),
),
)
```

4.3 Constraint

4.3.1 Layout harus responsif di HP kecil maupun besar.

Pada bagian constraint ini, saya akan memberikan bukti bahwa tampilan aplikasi benar benar responsif, artinya bisa menyesuaikan dengan berbagai ukuran layar ponsel. Sebelumnya saya menggunakan SliverGridDelegateWithMaxCrossAxisExtent untuk menampilkan daftar film dalam bentuk grid. Dengan cara ini, setiap item grid akan memiliki lebar maksimal tertentu, sehingga jumlah kolom akan otomatis menyesuaikan ukuran layar.





Ketika saya mencoba aplikasi di emulator dengan layar kecil, grid hanya menampilkan dua kolom agar poster tetap terlihat jelas dan tidak terlalu sempit. Sebaliknya, saat saya jalankan di layar yang lebih besar, jumlah kolom bertambah, sehingga ruang kosong bisa dimanfaatkan dengan baik. Hal ini menunjukkan bahwa penggunaan sliver membuat layout lebih fleksibel dan tetap rapi di berbagai ukuran layar.

Dengan cara ini, constraint “layout harus responsif” terbukti sudah terpenuhi, karena tampilan aplikasi tidak pecah atau berantakan meskipun dicoba di perangkat dengan resolusi berbeda. Bukti implementasi dapat dilihat langsung saat aplikasi dijalankan di emulator atau ponsel dengan ukuran layar berbeda, grid film tetap rapi serta poster tidak terpotong.

BAB 5

State Management & Logic

Dikerjakan oleh Logic Controller dan Frontend Engineer (Seat Matrix Module)[cite: 37].

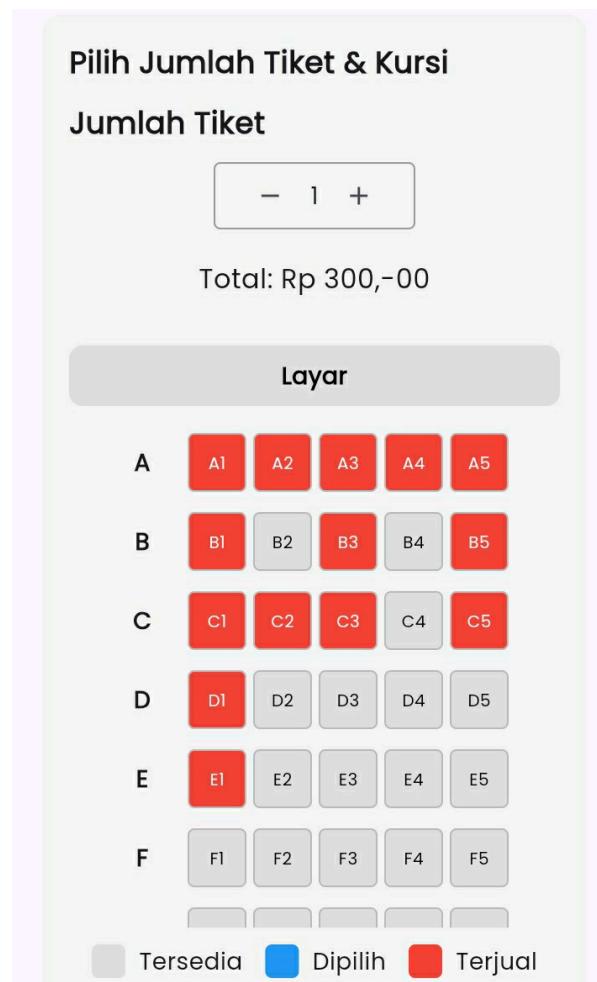
5.1 Seat Matrix Implementation

Bagian seat matrix dikerjakan oleh Rusydi Jabir Al Awfa (Frontend Engineer - Seat Matrix Module) untuk menampilkan dan mengelola pemilihan kursi dalam bioskop. Sistem ini memungkinkan pengguna untuk melihat status kursi (tersedia, dipilih, atau sudah dipesan) dan memilih kursi yang diinginkan.

5.1.1 Struktur Seat Matrix

Sistem seat matrix terdiri dari beberapa komponen utama:

- **SeatItemJabir**: Widget yang merepresentasikan satu kursi dengan status tertentu (available, selected, booked)
- **SeatMatrixJabir**: StatefulWidget utama yang menangani logika pemilihan kursi dan interaksi dengan pengguna
- **SeatStatus**: Enum yang mendefinisikan status kursi (available, selected, booked)



5.1.2 Implementasi Seat Item

Berikut adalah implementasi dari SeatItemJabir widget:

```
// SeatItem widget representing a single seat
class SeatItemJabir extends StatelessWidget {
    final String seatNumber;
    final SeatStatus status;
    final VoidCallback? onTap;
    final bool isSelected;

    const SeatItemJabir({
        Key? key,
        required this.seatNumber,
        required this.status,
        this.onTap,
        this.isSelected = false,
    }) : super(key: key);

    @override
    Widget build(BuildContext context) {
        Color seatColor;
        bool isInteractive = true;

        switch (status) {
            case SeatStatus.available:
                seatColor = isSelected ? Colors.blue : Colors.grey[300]!;
                break;
            case SeatStatus.selected:
                seatColor = Colors.blue;
                break;
            case SeatStatus.booked:
                seatColor = Colors.red;
                isInteractive = false;
                break;
            default:
                seatColor = Colors.grey[300]!;
        }

        return GestureDetector(
            onTap: isInteractive ? onTap : null,
            child: Container(
                width: 35,
                height: 35,
                margin: EdgeInsets.all(2),
                decoration: BoxDecoration(
                    color: seatColor,
                    borderRadius: BorderRadius.circular(4),
                    border: Border.all(
                        color: Colors.grey[400]!,
                        width: isSelected ? 2 : 1,
                    ),
                ),
            ),
        );
    }
}
```

```

        child: Center(
            child: Text(
                seatNumber,
                style: TextStyle(
                    fontSize: 10,
                    fontWeight: isSelected ? FontWeight.bold : FontWeight.normal,
                    color: status == SeatStatus.booked ? Colors.white :
Colors.black,
                ),
            ),
        ),
    ),
);
}
}

```

5.1.3 State Management untuk Pemilihan Kursi

Sistem ini menggunakan state management untuk melacak status kursi dan kursi yang dipilih oleh pengguna:

```

class _SeatMatrixJabirState extends State<SeatMatrixJabir> {
    Map<String, SeatStatus> seatStatuses = {};
    List<String> selectedSeats = [];
    bool _isLoading = true;

    @override
    void initState() {
        super.initState();
        _loadBookedSeats(); // Fetch booked seats from Firestore
    }

    // Fungsi untuk memuat status kursi yang sudah dipesan dari Firestore
    Future<void> _loadBookedSeats() async {
        try {
            // Fetch all bookings for this movie
            final querySnapshot = await FirebaseFirestore.instance
                .collection('bookings')
                .where('movie_title', isEqualTo: widget.movieTitle)
                .get();

            // Extract all booked seats from the query result
            Set<String> bookedSeatsSet = {};
            for (var doc in querySnapshot.docs) {
                final seats = doc.data()['seats'] as List;
                for (var seat in seats) {
                    if (seat is String) {
                        bookedSeatsSet.add(seat);
                    }
                }
            }
        }

        // Initialize seats based on booked status
    }
}

```

```

        _initializeSeats(bookedSeatsSet.toList());
    } catch (e) {
        print('Error loading booked seats: $e');
        // Initialize seats as available if there's an error
        _initializeSeats([]);
    } finally {
        setState(() {
            _isLoading = false;
        });
    }
}

// Fungsi untuk menginisialisasi status kursi
void _initializeSeats(List<String> bookedSeats) {
    // Initialize all seats as available or booked based on the fetched data
    for (int row = 0; row < 10; row++) {
        String rowLabel = String.fromCharCode(65 + row); // A, B, C, etc.
        for (int col = 1; col <= 5; col++) {
            String seatNumber = '${rowLabel}${col}';
            seatStatuses[seatNumber] = bookedSeats.contains(seatNumber)
                ? SeatStatus.booked
                : SeatStatus.available;
        }
    }
}

// Fungsi untuk mengganti status kursi (toggle seat selection)
void _toggleSeat(String seatNumber) {
    if (seatStatuses[seatNumber] == SeatStatus.booked) {
        return; // Can't select booked seats
    }

    setState(() {
        if (seatStatuses[seatNumber] == SeatStatus.selected) {
            seatStatuses[seatNumber] = SeatStatus.available;
            selectedSeats.remove(seatNumber);
        } else {
            seatStatuses[seatNumber] = SeatStatus.selected;
            selectedSeats.add(seatNumber);
        }
    });
}
}

```

5.2 Integrasi dengan Booking Service

Setelah pengguna memilih kursi, sistem akan mengintegrasikan dengan booking service untuk menyimpan data pemesanan. Fungsi konfirmasi booking:

```

void _confirmBooking() {
    final booking = BookingServiceIsal();
    booking.bookingMovie_Isal(
        movieTitle: widget.movieTitle,

```

```
basePrice: widget.totalPrice,
selectedSeats: selectedSeats,
);
if (context.mounted) {
ScaffoldMessenger.of(context).hideCurrentSnackBar();
ScaffoldMessenger.of(context).showSnackBar(
SnackBar(
content: Text('Berhasil booking ${widget.movieTitle}!'),
backgroundColor: Colors.green,
),
);
Navigator.pop(context);
}
}
```

5.3 Perhitungan Checkout

Total harga dihitung berdasarkan logika diskon NIM dan stok di Firebase berkurang otomatis (Transaction Write)[cite: 41].

BAB 6

Auth & Profile

Dikerjakan oleh Backend Engineer & Frontend Engineer.

6.1 Main

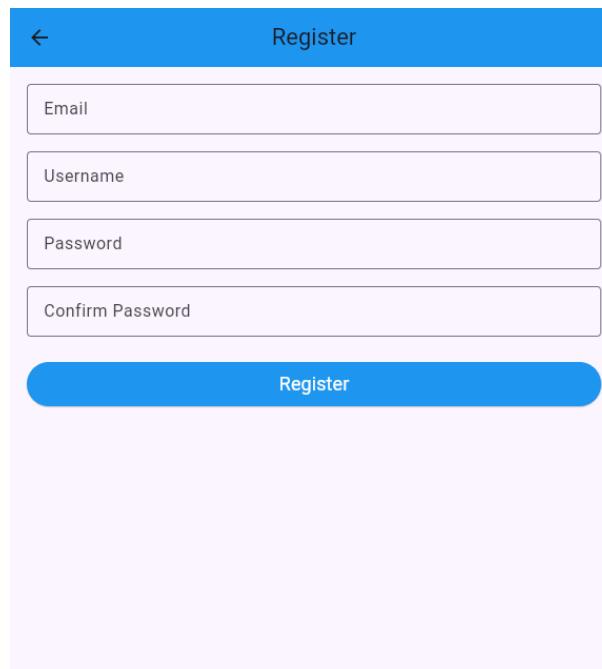
Pada bagian main ini, pertama tama yang saya lakukan adalah mengaktifkan Firebase agar aplikasi bisa terhubung dengan layanan login dan database. Setelah itu saya menjalankannya dengan memanggil myapp.

Di dalam MyApp, saya pakai MaterialApp untuk mengatur tema aplikasi, seperti warna biru sebagai warna utama, font Poppins, dan menghilangkan tulisan debug di pojok. Halaman awalnya saya arahkan ke initialScreen.

Nah, di initialScreen saya membuat logika untuk menentukan halaman pertama yang muncul. Disini saya menggunakan sharedpreferences yang menyimpan user id. Kalau masih loading, tampil indikator bulat. Kalau pengguna ada di sharedpreferences, maka langsung diarahkan ke HomePage. Sedangkan kalau user id di sharedpreferences tidak ada, otomatis diarahkan ke LoginPage.

6.2 Auth Register

Pada bagian ini saya membuat sebuah halaman yang memungkinkan pengguna bisa mendaftarkan diri di aplikasi CinemaPro. Yang mana nanti yang akan diinputkan adalah email, username, password, dan confirm password untuk konfirmasi apakah password yang diinputkan sama. Disini saya juga menambahkan sistem validasi, seperti pada email, saya menambahkan aturan bahwa alamat harus menggunakan domain kampus student.univ.ac.id, sehingga hanya mahasiswa yang bisa mendaftar dan masuk.



6.2.1 Validasi Email Mahasiswa

Bagian validasi email dikerjakan oleh Rusydi Jabir Al Awfa (Frontend Engineer - Seat Matrix Module). Sistem validasi email menggunakan Regular Expression untuk memastikan bahwa hanya alamat email dengan domain @student.univ.ac.id yang dapat digunakan untuk registrasi. Berikut adalah implementasi dari EmailValidator:

```
/// Utility class for email validation
class EmailValidator {
    /// Validates if an email is in the format required: \@student.univ.ac.id
    static bool isValidStudentEmail(String email) {
        // Regex pattern to match emails ending with \@student.univ.ac.id
        RegExp emailRegex = RegExp(r'^[a-zA-Z0-9._%+-]+@student\.univ\.ac\.id$');
        return emailRegex.hasMatch(email);
    }
}
```

6.2.2 Hashing Password

Untuk keamanan data pengguna, password di-hash sebelum disimpan ke database. Implementasi hashing password juga dikerjakan oleh Rusydi Jabir Al Awfa menggunakan algoritma SHA256 dengan salt dari hash key di file .env:

```
import 'dart:typed_data';
import 'dart:convert';
import 'package:crypto/crypto.dart';
import 'package:flutter_dotenv/flutter_dotenv.dart';

class PasswordHashUtil {
    /// Hash a password using SHA256 algorithm with the project's secret key as
    salt
    static Future<String> hashPassword(String password) async {
        // Load the hash key from .env file
```

```

    await dotenv.load(fileName: ".env");
    String hashKey = dotenv.env['HASH_KEY'];

    // Create salted password
    String saltedPassword = password + hashKey;

    // Hash using SHA256
    var bytes = utf8.encode(saltedPassword);
    var digest = sha256.convert(bytes);

    return digest.toString(); // Returns hex representation of the hash
}
}

```

6.2.3 User Service

Bagian UserService yang bertugas untuk menyimpan data pengguna ke Firestore juga dikerjakan oleh Rusydi Jabir Al Awfa:

```

class UserService {
  final FirebaseFirestore _firebase = FirebaseFirestore.instance;

  /// Create a new user with hashed password
  Future<bool> createUserWithEmailAndPassword({
    required String email,
    required String username,
    required String password,
  }) async {
    try {
      // Hash the password
      String hashedPassword = await PasswordHashUtil.hashPassword(password);

      // Generate a new document ID for the user
      String newUid = _firebase.collection('users').doc().id;

      // Create user profile in Firestore using the existing user model
      UserModelCheryl newUser = UserModelCheryl(
        uid: newUid, // Primary Key (auto-generated by Firestore)
        email: email.trim(), // Validated with \@student.univ.ac.id regex
        username: username.trim(), // Display name
        balance: 200000, // Default balance (200000), as specified
        password: hashedPassword, // Store hashed password
      );

      // Create user profile in Firestore with all required fields
      await _firebase.collection('users').doc(newUid).set({
        ...newUser.toMapCheryl(), // Use the model's toMap method
        'createdAt': FieldValue.serverTimestamp(), // Server timestamp for accuracy
      });

      return true;
    } catch (e) {

```

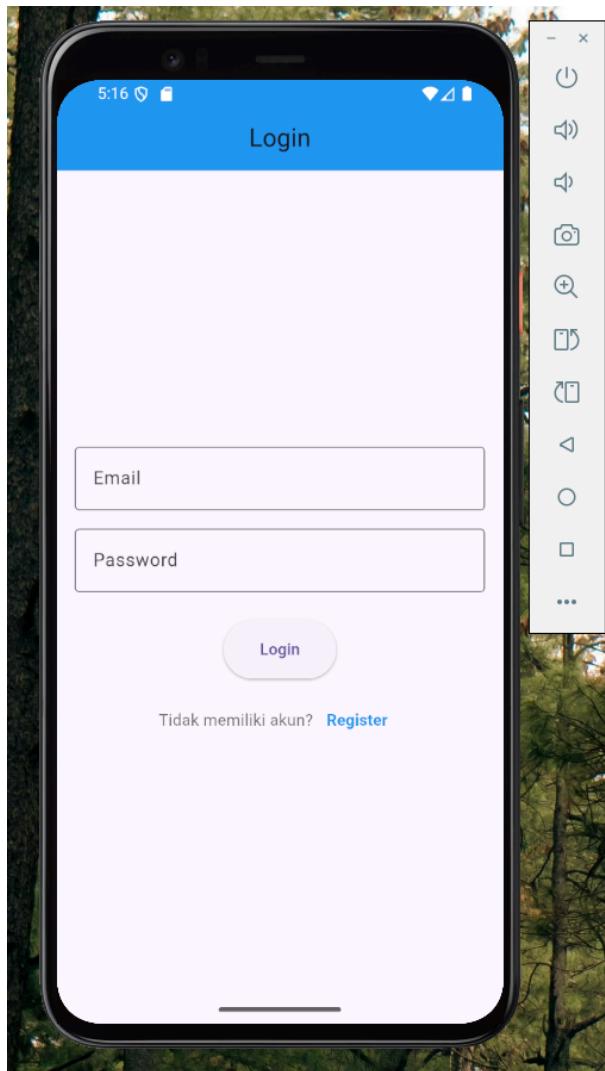
```

        print('Error creating user: $e');
        return false;
    }
}
}

```

6.3 Auth Login

Pada bagian login ini saya membuat sebuah halaman yang memungkinkan pengguna masuk ke aplikasi menggunakan akun yang sudah terdaftar di Firebase. Tampilan login ini terdiri dari dua input utama, yaitu email dan password, yang masing-masing dilengkapi dengan validasi. Untuk email, saya menambahkan aturan bahwa alamat harus menggunakan domain kampus student.univ.ac.id, sehingga hanya mahasiswa yang bisa mendaftar dan masuk.



Berikut kode untuk form login dengan validasi. Form ini menggunakan FormKey untuk validasi dan TextEditingController untuk mengambil input dari pengguna.

Selain itu, saya menambahkan logika agar ketika tombol Login ditekan, aplikasi terlebih dahulu memvalidasi form. Jika valid, maka saya akan memanggil fungsi finduser yang sebelumnya sudah dibuat oleh jabir. Selama proses ini berlangsung, tombol login diganti dengan indikator loading agar pengguna tahu bahwa proses sedang berjalan. Jika login

berhasil, muncul pesan Login berhasil dengan warna hijau, lalu pengguna diarahkan ke halaman utama HomePage dan data login pengguna tersebut juga saya simpan di sharedpreferences yaitu uid nya. Sebaliknya, jika login gagal, sistem menampilkan pesan error berwarna merah.

Di bagian bawah halaman, saya juga menyediakan opsi bagi pengguna yang belum memiliki akun. Terdapat teks Tidak memiliki akun dengan tombol Register yang akan mengarahkan pengguna ke halaman pendaftaran. Dengan cara ini, halaman login tidak hanya berfungsi untuk login saja, tetapi juga menjadi navigator bagi pengguna baru untuk membuat akun.

Tidak memiliki akun? [Register](#)

6.4 Profile

Halaman profile untuk menampilkan identitas pengguna dan menampilkan riwayat transaksi. Pertama, untuk bagian header profil, saya menggunakan FutureBuilder untuk mengambil data detail pengguna (Username, Email dan saldo) dari collection users berdasarkan UID pengguna yang sedang login, kemudian data diambil dan dikonversi menggunakan UserModelCheryl.

Kedua, untuk bagian Riwayat Tiket, saya menggunakan StreamBuilder. Penggunaan Stream dipilih agar daftar tiket dapat diperbarui secara real-time tanpa perlu me-refresh halaman jika ada transaksi baru. Saya menerapkan query filtering menggunakan where untuk memastikan pengguna hanya melihat tiket miliknya sendiri.

Selain itu, sesuai spesifikasi teknis, saya mengimplementasikan library qr_flutter untuk generate QR CODE secara otomatis berdasarkan booking_id dari setiap transaksi.



BAB 7

Pengujian & Penutup

7.1 Handling Data

Bukti fitur `ListView.builder` tidak error saat data kosong dan implementasi `LoadingIndicator`[cite: 44, 45].

7.2 Link Video Demo

Berikut adalah link video demo aplikasi yang diunggah ke Google Drive/YouTube:

- Link: [Masukkan Link Di Sini]