BAB 3
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

Selama pelaksanaan kerja magang di PT Hexa Global Teknologi, posisi yang dijalani adalah sebagai *IT Developer Intern* dengan tanggung jawab utama sebagai *App Developer*. Dalam pelaksanaannya, kegiatan magang diawasi oleh Felix Carry selaku *Co-Founder* dan *IT Engineer* di perusahaan tersebut. Proses kerja diawali dengan pemberian tugas oleh *supervisor* yang meliputi penjelasan mengenai aplikasi yang akan dikembangkan, pemberian *application requirements*, serta rancangan antarmuka (*UI design*). Seluruh proses pengembangan aplikasi dilakukan menggunakan platform *GitHub* sebagai media kolaborasi dan pengelolaan versi proyek.

Setelah beberapa fitur atau halaman aplikasi selesai dikerjakan, hasil pengembangan akan di-*push* ke *GitHub* untuk diperiksa oleh *supervisor*. Tahap selanjutnya adalah pelaksanaan *meeting* untuk mempresentasikan hasil pekerjaan yang telah dibuat. Dalam sesi tersebut, *supervisor* memberikan umpan balik dan evaluasi terkait hasil pengembangan. Berdasarkan masukan tersebut, dilakukan revisi dan penyempurnaan agar aplikasi sesuai dengan standar dan kebutuhan perusahaan.

3.2 Tugas yang Dilakukan

Tugas utama selama pelaksanaan magang adalah mengembangkan *frontend* aplikasi *GameSynce* menggunakan data *dummy* sebagai simulasi. *GameSynce* merupakan aplikasi berbasis *mobile* yang berfungsi untuk membantu pengguna menemukan dan membentuk grup bermain *game* melalui sistem pencocokan tim berdasarkan preferensi peran, ketersediaan waktu, serta skor reputasi (*Kudos*). Aplikasi ini bertujuan untuk mempermudah proses pencarian rekan bermain dengan antarmuka yang intuitif dan berbasis *swipe*, sekaligus menyajikan informasi preferensi tim seperti urutan posisi dan *friend codes* secara terstruktur. Seluruh proses pengembangan *frontend* aplikasi ini dilakukan sendiri tanpa tim.

3.3 Uraian Pelaksanaan Magang

Berikut ini adalah uraian kerja magang di PT Hexa Global Teknologi yang dibagi per minggu bisa dilihat pada Tabel 3.1:

Tabel 3.1. Pekerjaan yang dilakukan tiap minggu selama kerja magang

Minggu Ke-	Uraian Kegiatan
1	Perkenalan dengan perusahaan, pemberian tugas magang, serta mempelajari dan berlatih *Flutter*.
2	Pemberian *application requirements*, desain tampilan *UI*, pembelajaran dari *supervisor*, pemberian akses *GitHub*, serta mulai mengerjakan halaman *Login*.
3	Mengerjakan halaman *First-Time Setup* yang terdiri dari empat *tab*: *Basic Info*, *Availability*, *Game Preference*, dan *Game Description*.
4	Membuat halaman *Discover* yang berisi *dropdown* pemilihan game dan *card* berisi detail *room*.
5	Melanjutkan halaman *Discover* dengan menambahkan data *dummy* untuk ditampilkan pada *card*, serta menambahkan fitur *swipe card*.
6	Membuat struktur utama untuk perpindahan halaman menggunakan *Bottom Navigator*, membuat halaman *Rating Player* yang berisi *card* detail pemain serta fitur *swipe*, dan membuat halaman *Room* yang terdiri dari tiga *tab*: *Recruit*, *Active*, dan *Closed*.
7	Mengimplementasikan perpindahan dari halaman *Room* ke halaman *Detail Room* berdasarkan statusnya. Mengerjakan halaman detail *Recruit Room* dan *Active Room*.
8	Melanjutkan pengerjaan halaman *Active Room* dengan menambahkan fitur *chat*, *player voting*, dan *room management*, serta melakukan *meeting* dengan *supervisor*.
	Lanjut pada halaman berikutnya

Tabel 3.1 Pekerjaan yang dilakukan tiap minggu selama kerja magang (lanjutan)

Minggu Ke-	Uraian Kegiatan
9	Melakukan perbaikan dan perapihan kode pada seluruh halaman aplikasi berdasarkan hasil evaluasi dan masukan dari *supervisor*, termasuk penyesuaian struktur kode agar lebih konsisten dan mudah dipahami.
10	Melakukan perbaikan kode lanjutan, pengecekan ulang fungsionalitas aplikasi, serta memastikan tidak terdapat kesalahan sebelum melakukan *push* ke repositori GitHub.
11	Membuat halaman *Add Room* yang digunakan untuk menambahkan data *room* baru, serta melakukan pemisahan komponen antarmuka ke dalam beberapa *widget* terpisah.
12	Membuat halaman *Settings* yang berisi pengaturan aplikasi, dan memisahkan setiap bagian antarmuka ke dalam *widget* terpisah agar struktur kode lebih rapi.
13	Melakukan perubahan *encapsulation* pada seluruh kode di bagian *view model*, membuat *widget empty state card*, mengubah beberapa *widget* agar bersifat *reusable*, dan melakukan *push* hasil pengerjaan ke repositori GitHub.
14	Melakukan sinkronisasi state pada global simulated data agar konsisten di semua *tab*, mengatur navigasi agar memuat ulang data terbaru, serta meeting terakhir bersama *supervisor*.

3.3.1 Implementasi dan Hasil Kerja

Proses implementasi aplikasi dilakukan menggunakan *framework Flutter* yang dikembangkan oleh *Google* untuk pengembangan aplikasi *mobile* lintas platform. *Flutter* menggunakan pendekatan deklaratif berbasis *widget* sebagai representasi antarmuka pengguna yang ringan dan bersifat *immutable*. Perubahan data atau status akan memicu pembangunan ulang *widget* sehingga pembaruan tampilan dapat dilakukan secara efisien dengan performa mendekati aplikasi *native*. Selain itu, *Flutter* menyediakan pustaka antarmuka seperti Material *Components* dan *Cupertino* untuk mendukung tampilan yang adaptif di berbagai ukuran dan resolusi layar [5].

Bahasa pemrograman yang digunakan adalah *Dart*, yaitu bahasa yang dioptimalkan untuk pengembangan aplikasi cepat di berbagai platform dan menjadi fondasi utama *Flutter*. *Dart* dirancang untuk meningkatkan produktivitas pengembang sekaligus menghasilkan kualitas aplikasi yang baik pada target *web, mobile,* dan *desktop*. Sebagai bahasa inti *Flutter*, *Dart* menyediakan *runtime* serta dukungan untuk pengembangan, pengujian, dan analisis kode. Selain itu, *Dart* memiliki sistem tipe yang aman dengan dukungan *sound null safety* untuk meminimalkan kesalahan nilai *null* [6].

Arsitektur aplikasi menerapkan pola *MVVM* yang berfokus pada pemisahan antara logika bisnis dan tampilan antarmuka aplikasi. Pola ini terdiri dari tiga lapisan utama, yaitu *Model* sebagai representasi data untuk kebutuhan logika bisnis, *View* sebagai komponen antarmuka pengguna, serta *ViewModel* sebagai penghubung antara keduanya. *Model* berperan dalam merepresentasikan data yang digunakan dalam logika aplikasi, sedangkan *View* bertugas mengatur bagaimana informasi tersebut ditampilkan kepada pengguna. *ViewModel* berfungsi untuk berinteraksi dengan *Model* dan meneruskan data ke *View*, dengan pengelolaan *state* aplikasi dibantu oleh *Provider* agar alur data tetap terstruktur [7].

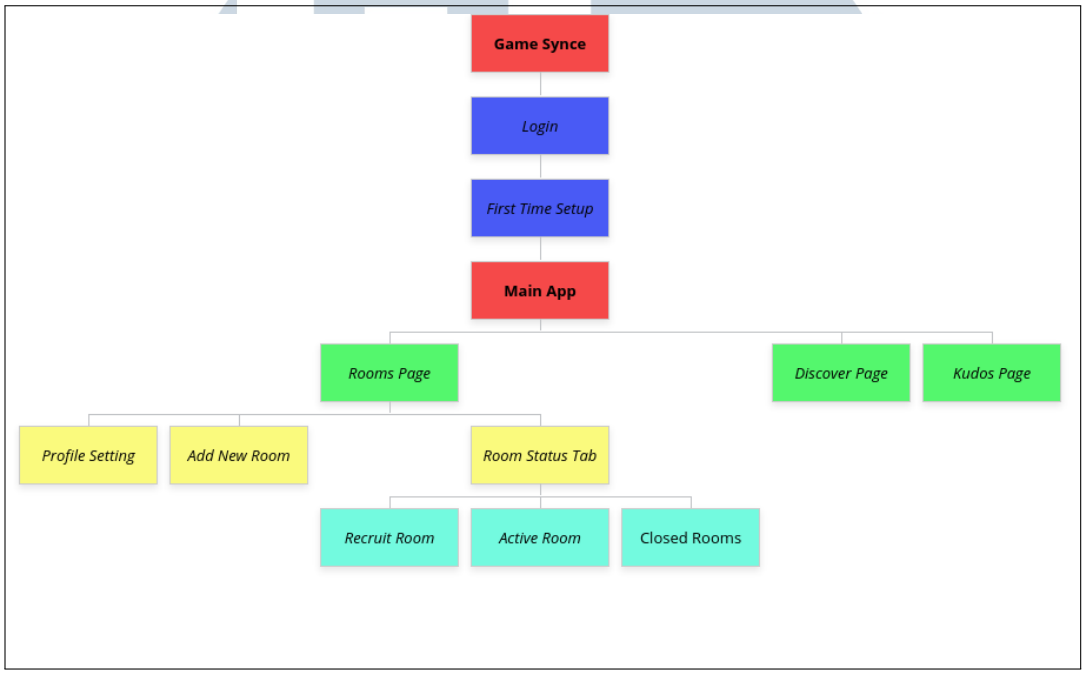
Seluruh tampilan antarmuka aplikasi dikembangkan secara responsif dengan bantuan *ScreenUtil* untuk menyesuaikan ukuran layar dan *font* pada berbagai perangkat *mobile* [8]. Pada tahap pengembangan, data yang digunakan masih berupa data *dummy* sebagai simulasi sebelum terhubung dengan data sebenarnya. Setiap halaman dan komponen dirancang dengan memperhatikan konsistensi tampilan serta interaksi pengguna. Hasil implementasi menunjukkan aplikasi berjalan dengan baik, memiliki antarmuka yang konsisten, dan mudah digunakan.

A Perancangan

Pada tahap perancangan aplikasi, seluruh acuan pengembangan diperoleh langsung dari pihak kantor, yaitu PT Hexa Global Teknologi. Perancangan tersebut diberikan dalam bentuk rancangan antarmuka pengguna *(UI design)* yang menjadi pedoman utama dalam pengembangan tampilan aplikasi. Selain itu, perusahaan juga menyediakan *application requirements* yang berisi kebutuhan fungsional dan alur penggunaan aplikasi secara keseluruhan. Dengan adanya perancangan yang telah ditetapkan sejak awal, proses pengembangan aplikasi dapat dilakukan secara lebih terarah dan sesuai dengan standar yang ditentukan perusahaan.

B Sitemap Aplikasi

Gambar 3.1 menunjukkan *sitemap* dari aplikasi yang dikembangkan, menggambarkan alur navigasi antar halaman dari proses *login* hingga halaman utama aplikasi.

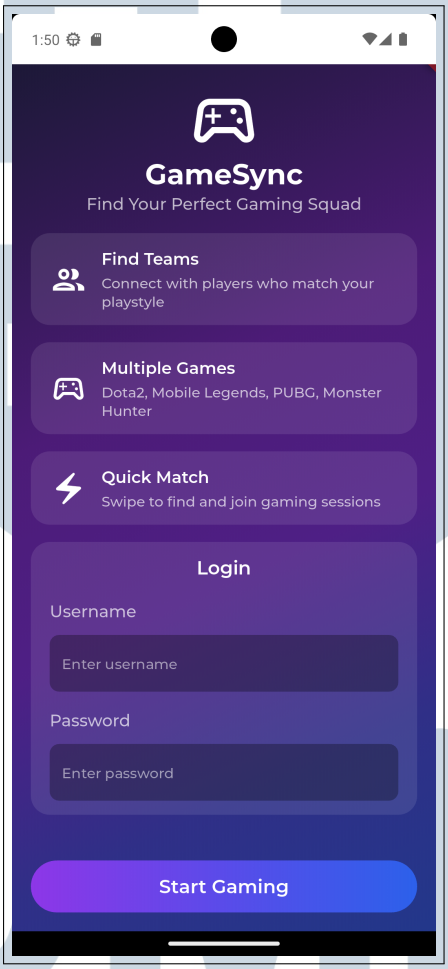


Gambar 3.1. *Sitemap gamesynce*

Sitemap ini memperlihatkan struktur halaman secara hierarkis, dimulai dari halaman *Login* sebagai pintu masuk utama aplikasi, kemudian menuju *First-Time Setup* untuk pengaturan awal sebelum pengguna dapat mengakses halaman utama. Halaman utama aplikasi dibagi menjadi tiga menu utama, yaitu *Kudos Page* yang berfungsi untuk melakukan *review* terhadap pemain lain, *Discover Page* yang berfungsi untuk mencari *room game* yang tersedia, dan *Rooms Page*. Pada *Rooms Page*, terdapat tiga *tab* berdasarkan status room, yaitu *Recruit Room* untuk room yang sedang mencari pemain baru, *Active Room* untuk room yang sedang berlangsung, dan *Closed Room* untuk room yang sudah selesai. Selain itu, pada *Rooms Page* terdapat fitur untuk menambahkan room baru melalui *Add New Room* serta melakukan pengaturan profil pengguna pada *Profile Setting*. Dengan hierarki yang jelas ini, *sitemap* membantu menunjukkan hubungan antar halaman, memudahkan navigasi pengguna, dan memberikan gambaran utuh mengenai struktur aplikasi yang dikembangkan.

C Login

Tampilan halaman *login* dapat dilihat pada Gambar 3.2. Gambar tersebut menunjukkan susunan elemen antarmuka yang dirancang untuk memberikan pengalaman awal yang jelas, informatif, dan mudah dipahami oleh pengguna.

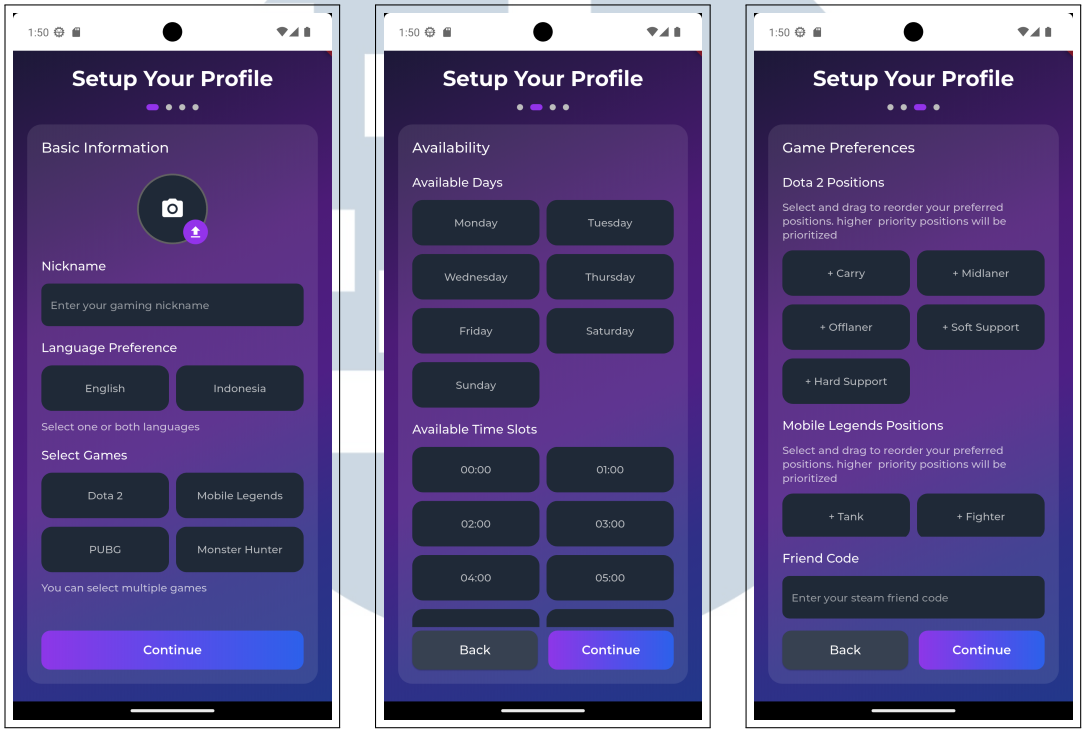


Gambar 3.2. Halaman *login*

Halaman *login* ini menampilkan logo dan nama aplikasi pada bagian atas sebagai identitas utama yang membantu pengguna mengenali aplikasi sejak pertama kali dibuka. Di bawah area tersebut, ditampilkan tiga fitur utama aplikasi, yaitu *Find Teams*, *Multiple Games*, dan *Quick Match*, yang berfungsi sebagai gambaran singkat mengenai manfaat dan kapabilitas aplikasi bagi pengguna. Setelah itu, tersedia *form login* yang terdiri dari kolom input email dan kata sandi sebagai sarana autentikasi pengguna sebelum masuk ke dalam sistem. Pada bagian paling bawah, terdapat tombol aksi yang digunakan untuk melanjutkan proses *login*.

D First-Time Setup

Keempat tampilan pada proses *First-Time Setup* ditunjukkan pada Gambar 3.3. Setiap gambar merepresentasikan fungsi dari masing-masing *tab*.



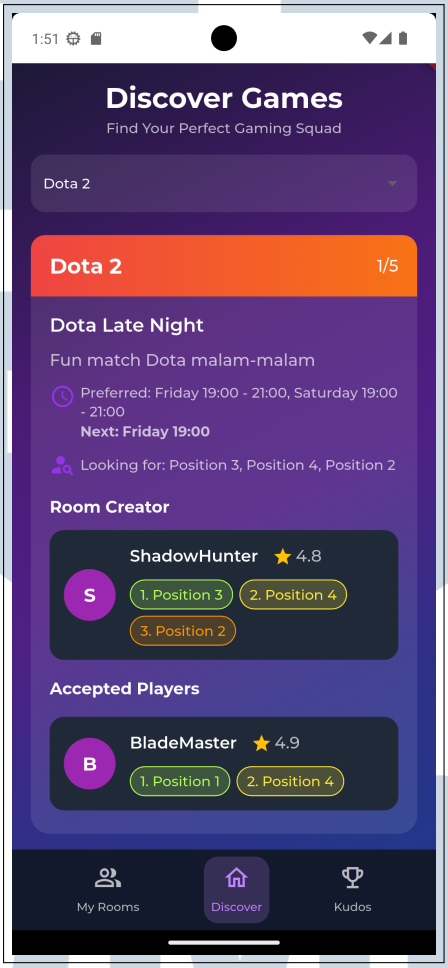
(a) setup_1 (b) setup_2 (c) setup_3

Gambar 3.3. *First-time setup*

Halaman *setup* merupakan halaman pertama yang muncul untuk pengguna baru dan berfungsi untuk mengumpulkan informasi dasar sebelum pengguna mulai menggunakan aplikasi. Pada halaman ini terdapat empat *tab* utama yang masing-masing memiliki peran berbeda. *Tab* pertama adalah *Basic Information* yang berisi kolom untuk mengisi *nickname*, memilih bahasa, dan menentukan jenis permainan yang akan dimainkan. *Tab* berikutnya adalah *Availability* yang memungkinkan pengguna memilih hari dan waktu bermain *game* sesuai jadwal mereka. Setelah itu terdapat *tab Game Preference* yang berisi pilihan preferensi permainan seperti posisi atau senjata yang disukai, dan preferensi tersebut dapat diurutkan berdasarkan prioritas. *Tab* terakhir adalah *Game Description* yang memungkinkan pengguna menuliskan deskripsi mengenai cara bermain mereka untuk membantu proses pencarian tim.

E Discover

Gambar 3.4 menampilkan antarmuka halaman *discover* yang menunjukkan struktur kartu *room* dan interaksi *swipe*.

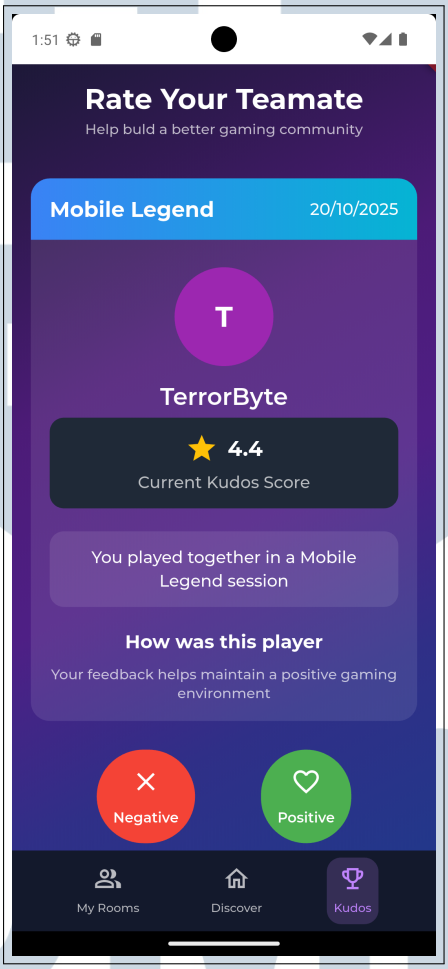


Gambar 3.4. Halaman *discover*

Halaman *discover* digunakan sebagai sarana utama bagi pengguna untuk mencari dan menemukan *room* permainan yang sesuai dengan preferensi mereka. Pada bagian atas halaman tersedia *dropdown* pemilihan *game* yang memungkinkan pengguna menentukan jenis permainan yang ingin diikuti, sehingga daftar *room* yang ditampilkan akan secara otomatis menyesuaikan dengan pilihan tersebut. Di bawahnya terdapat sebuah kartu yang menampilkan judul *room*, deskripsi, nama pembuat, serta pemain yang sudah bergabung. Interaksi pada kartu dilakukan dengan cara *swipe*, yaitu ke kanan untuk bergabung dan ke kiri untuk menolak *room* tersebut, serta terdapat tombol aksi dengan fungsi yang sama.

F Kudos (Rating)

Tampilan halaman *kudos* yang berisi daftar pemain ditampilkan pada Gambar 3.5. Tampilan tersebut menggambarkan bagaimana kartu pemain digunakan dalam proses pemberian *rating*.

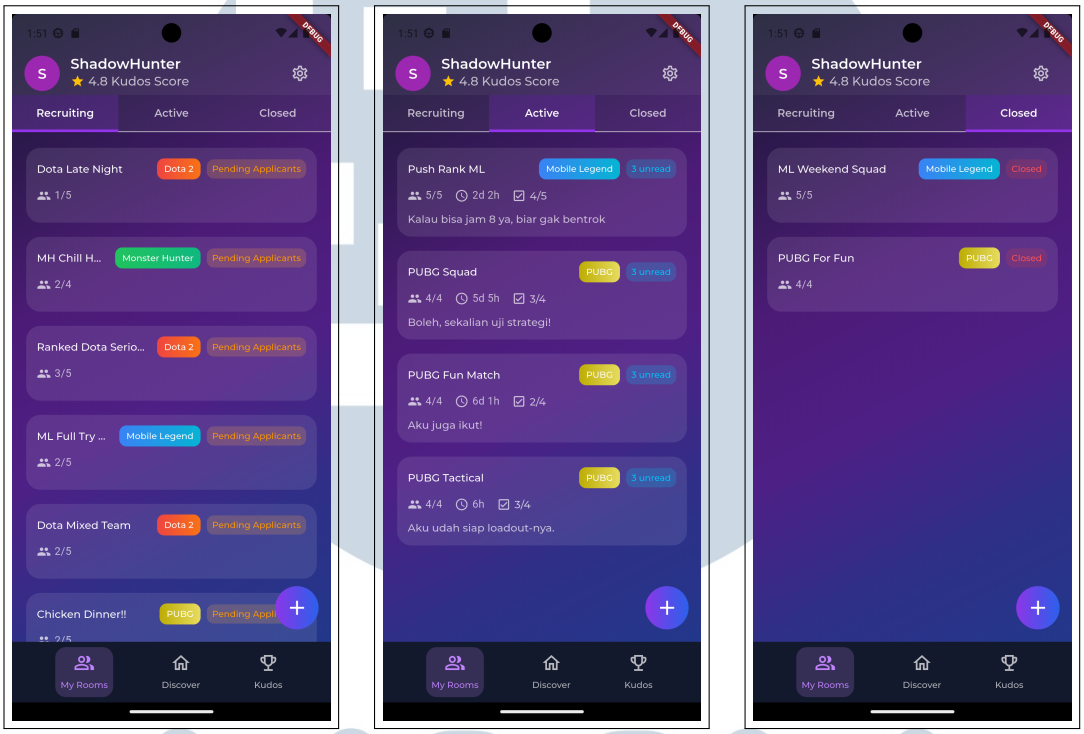


Gambar 3.5. Halaman *kudos*

Halaman *kudos* berfungsi untuk memberikan penilaian atau *rating* terhadap pemain lain yang sebelumnya pernah bermain bersama dalam satu *room*. Setiap pemain ditampilkan melalui sebuah kartu yang berisi nama, *rating*, dan nama *game* yang sebelumnya dimainkan bersama. Selain itu, kartu juga dilengkapi dengan keterangan singkat mengenai konteks pertemuan permainan, sehingga pengguna dapat mengingat kembali pemain tersebut. Interaksi penilaian dilakukan dengan cara *swipe* kartu ke kanan untuk memberikan *rating* positif dan kiri untuk *rating* negatif. Selain itu, terdapat tombol aksi dengan fungsi yang sama.

G Rooms

Visualisasi ketiga *tab Recruiting*, *Active*, dan *Closed* dapat dilihat pada Gambar 3.6. Gambar tersebut memperlihatkan perbedaan susunan informasi pada tiap status *room*.



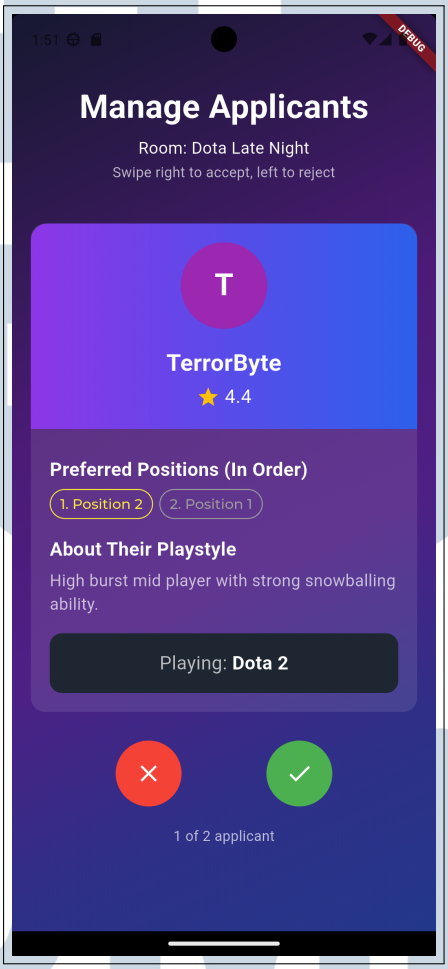
(a) rooms_1 (b) rooms_2 (c) rooms_3

Gambar 3.6. Room dengan 3 tab sesuai status

Halaman *rooms* berfungsi untuk menampilkan daftar ruangan berdasarkan status dari setiap *room* yang dimiliki atau diikuti oleh pengguna. Pada bagian atas halaman terdapat nama pengguna beserta tombol untuk masuk ke halaman pengaturan akun. Di bawahnya terdapat *tab bar* yang terdiri dari tiga kategori status *room*, yaitu *Recruiting*, *Active*, dan *Closed*, yang digunakan untuk memfilter daftar *room*. Daftar *room* yang ditampilkan akan secara otomatis menyesuaikan dengan status yang dipilih pada *tab bar*, sehingga memudahkan pengguna dalam memantau setiap tahapan *room*. Setiap item *room* menampilkan informasi ringkas seperti nama *room*, jenis *game*, jumlah pemain, serta informasi pendukung lainnya yang relevan. Pengguna dapat memilih salah satu *room* untuk melihat detail lebih lanjut sesuai dengan status ruangan tersebut. Selain itu, pada bagian bawah halaman terdapat sebuah *floating button* yang berfungsi untuk membuat *room* baru.

H Recruit Room

Tampilan lengkap halaman *recruit room* ditunjukkan pada Gambar 3.7. Gambar tersebut memperlihatkan posisi kartu pemain yang dapat di-*swipe* sesuai fungsinya.

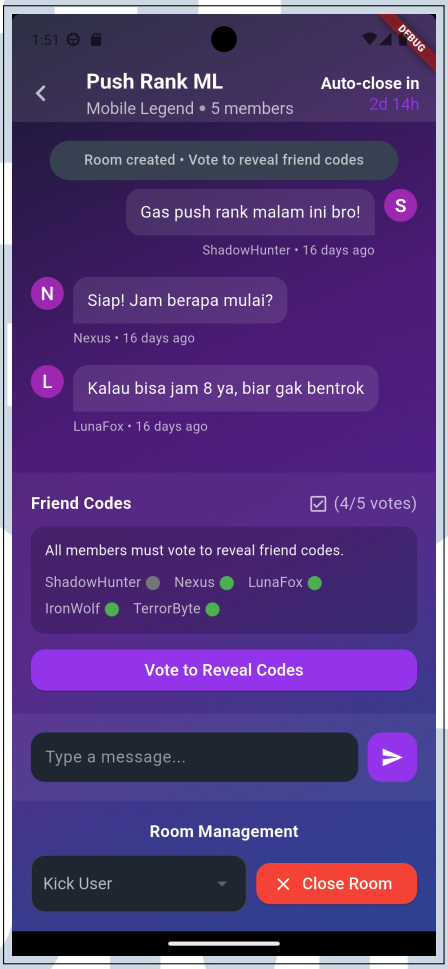


Gambar 3.7. Halaman *recruit*

Halaman *recruit room* merupakan detail dari *room* berstatus *Recruiting* dan berfungsi untuk meninjau pemain yang ingin bergabung. Pada bagian atas halaman menampilkan nama *room* beserta informasi singkat mengenai ruangan tersebut. Selanjutnya, pada bagian bawah terdapat sebuah *card* yang berisi detail pemain yang mengajukan permintaan bergabung untuk membantu pemilik *room* menentukan kecocokan pemain tersebut. *Card* ini dapat di-*swipe* ke kanan untuk menerima atau ke kiri untuk menolak, serta tersedia tombol di bagian bawah dengan fungsi yang sama.

I Active Room

Tampilan halaman *active room* dapat dilihat pada Gambar 3.8. Gambar tersebut memperlihatkan area *chat*, informasi pemain, serta menu manajemen ruangan.

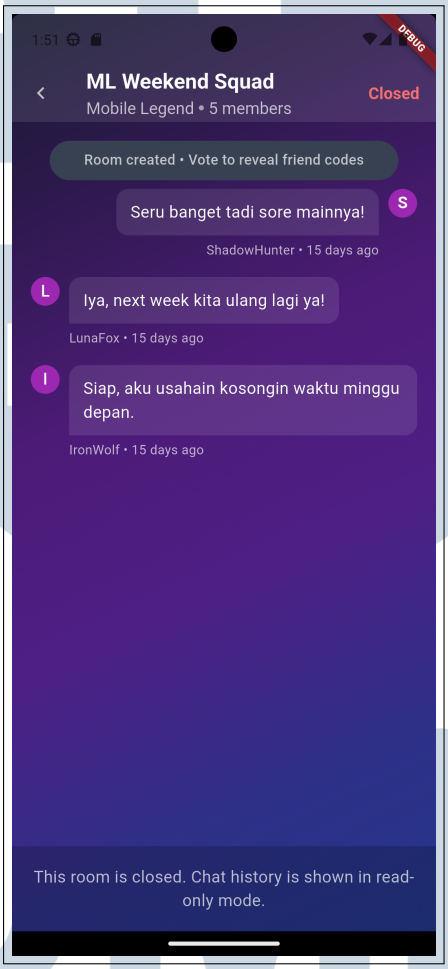


Gambar 3.8. Halaman *active*

Halaman *active room* merupakan halaman detail yang menampilkan informasi dari *room* yang sedang berstatus aktif dan digunakan dalam proses permainan. Pada bagian atas berisi nama *room*, nama *game*, jumlah pemain, serta hitungan mundur menuju penutupan otomatis. Di bawahnya terdapat fitur *chat* yang memungkinkan pemain berkomunikasi, serta daftar *friend codes* yang muncul setelah seluruh pemain melakukan *voting*. Bagi pengguna yang berperan sebagai *room creator*, tersedia menu *Room Management* yang memungkinkan mereka mengeluarkan pemain atau menutup *room*.

J Closed Room

Gambar 3.9 menunjukkan tampilan halaman *closed room* beserta riwayat percakapannya. Tampilan ini menggambarkan fungsi halaman yang bersifat informatif tanpa interaksi tambahan.

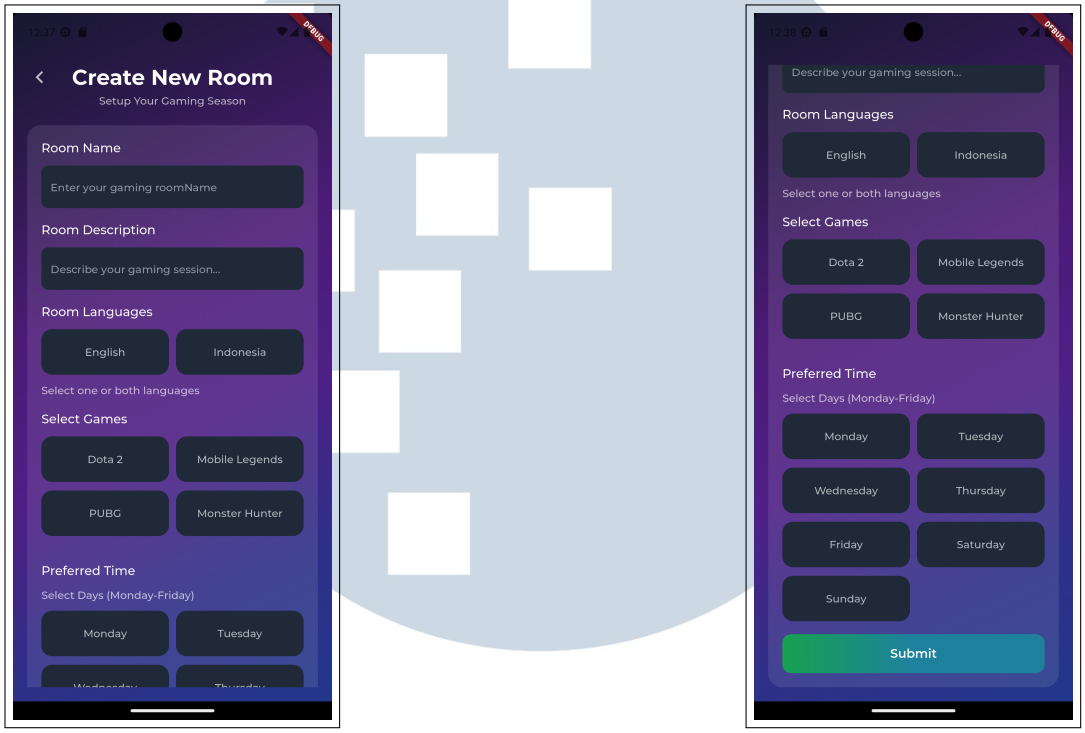


Gambar 3.9. Halaman *closed*

Halaman *closed room* merupakan halaman detail yang menampilkan informasi dari *room* yang telah berstatus *Closed* dan tidak lagi dapat digunakan untuk aktivitas maupun interaksi. Pada bagian atas halaman ditampilkan nama *room*, jenis *game*, jumlah pemain, serta label *Closed* sebagai penanda status. Halaman ini juga menampilkan riwayat percakapan dari fitur *chat* yang terjadi saat *room* masih aktif sebagai bentuk arsip interaksi antar pemain. Seluruh konten ditampilkan dalam mode (*read-only*) tanpa menyediakan fitur interaksi, karena halaman ini berfungsi sebagai dokumentasi setelah sesi permainan berakhir.

K Add New Room

Gambar 3.10 memperlihatkan tampilan halaman *add new room* yang digunakan untuk membuat ruangan permainan baru.



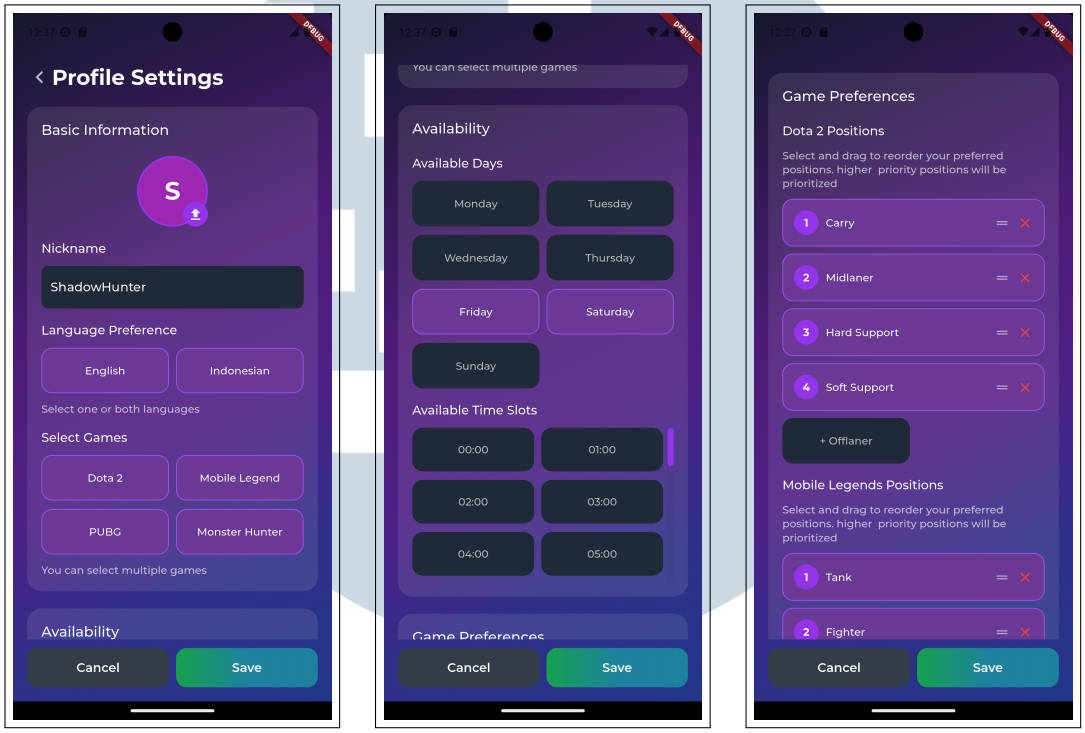
(a) *add_room_1* (b) *add_room_2*

Gambar 3.10. *Add room*

Halaman *Add Room* merupakan halaman yang digunakan oleh pengguna untuk membuat *room* baru sesuai dengan kebutuhan dan preferensi bermain. Pada bagian atas halaman terdapat tombol kembali yang berfungsi untuk membatalkan proses pembuatan *room* dan mengarahkan pengguna kembali ke halaman sebelumnya. Di bawahnya disediakan berbagai *input field* yang mencakup nama *room*, deskripsi singkat, bahasa yang digunakan, pilihan *game*, serta pemilihan *role* dan waktu bermain yang diinginkan, sehingga pengguna dapat menentukan karakteristik *room* secara lebih rinci. Selain itu, tersedia pengaturan preferensi jadwal bermain yang memungkinkan pengguna memilih hari dan jam bermain tertentu agar calon anggota *room* memiliki kesesuaian waktu. Pada bagian paling bawah halaman terdapat tombol *submit* yang berfungsi untuk menyimpan seluruh data yang telah diisi dan membuat *room* baru sesuai dengan pengaturan yang ditentukan.

L Profile Settings

Gambar 3.11 menampilkan tampilan halaman *profile settings* yang berisi berbagai bagian pengaturan informasi pengguna.



(a) *Profile_settings_1* (b) *Profile_settings_2* (c) *Profile_settings_3*

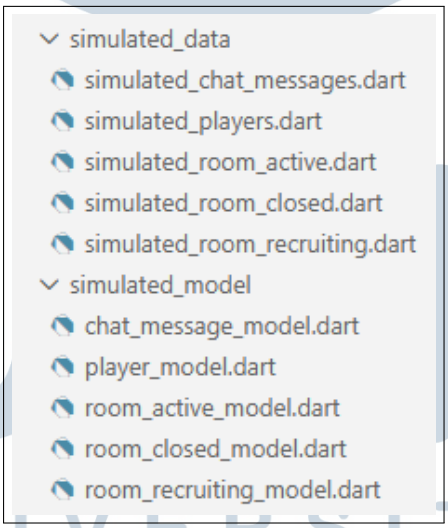
Gambar 3.11. *Profile settings*

Halaman *profile settings* terdiri dari lima bagian utama, yaitu *basic information*, *availability*, *game preferences*, *friend codes*, dan *game description*, yang digunakan untuk menyesuaikan profil dan preferensi pengguna secara menyeluruh. Pada bagian *basic information*, pengguna dapat mengelola informasi dasar seperti foto profil, bahasa aplikasi, serta daftar *game* yang dimainkan. Bagian *availability* memungkinkan pengguna mengatur jadwal dan preferensi waktu bermain agar sesuai dengan ketersediaan mereka. Selanjutnya, pada *game preferences*, pengguna dapat mengatur urutan preferensi bermain untuk setiap *game*. Bagian *friend codes* digunakan untuk memasukkan atau memperbarui *invite code* masing-masing *game*, sedangkan pada *game description* pengguna dapat menuliskan deskripsi singkat mengenai gaya bermain. Di bagian bawah halaman tersedia dua tombol *sticky*, yaitu *cancel* untuk membatalkan dan *save* untuk menyimpan perubahan yang telah dilakukan.

M Data Dummy

Data *dummy* digunakan pada aplikasi ini untuk mensimulasikan data yang seharusnya berasal dari *backend* atau *database*. Dengan adanya data *dummy*, proses pengembangan antarmuka (*UI*) dapat dilakukan tanpa harus menuliskan data secara manual di setiap komponen *frontend*, khususnya pada tampilan yang memiliki struktur data cukup kompleks. Data *dummy* cukup dipanggil dan dipetakan ke *UI* sehingga tampilan aplikasi tetap menyerupai kondisi nyata. Pendekatan ini memungkinkan proses pengembangan dan pengujian fitur berjalan lebih efisien meskipun integrasi *backend* belum tersedia.

Pada aplikasi ini terdapat lima jenis data *dummy* yang digunakan, yaitu data *dummy player*, *room recruiting*, *room active*, *room closed*, dan *chat message*. Setiap data *dummy* terdiri dari dua buah *file*, yaitu *file model* yang mendefinisikan struktur data dan *file* yang berisi kumpulan data *dummy* itu sendiri. Struktur direktori data *dummy* pada aplikasi ini dapat dilihat pada Gambar 3.12, yang menunjukkan pembagian *file model* dan *file* data untuk setiap jenis data *dummy* sehingga memudahkan pengelolaan selama proses pengembangan.



Gambar 3.12. Direktori *data dummy*

File model berfungsi untuk mendefinisikan struktur data, *field*, serta tipe data yang digunakan, sehingga menyerupai struktur data asli dari *backend*. Sementara itu, *file* data *dummy* berisi kumpulan data statis yang dibuat berdasarkan *model* tersebut dan digunakan langsung untuk ditampilkan pada antarmuka pengguna. Pemisahan ini bertujuan agar struktur data tetap konsisten dan mudah dikembangkan ketika nantinya terhubung dengan *database* sebenarnya.

M.1 Data Dummy Player

Data *dummy player* direpresentasikan melalui kelas *Player* dan digunakan sebagai simulasi data pengguna yang akan ditampilkan pada berbagai fitur aplikasi dapat dilihat pada Kode 3.1.

```
Player(
    id: '1',
    nickname: 'ShadowHunter',
    steamFriendcode: 'SHDW-4592',
    moleFriendcode: 'MOL-9021',
    mhwFriendcode: 'MHW-1234',
    pubgFriendcode: 'PUBG-5678',
    languages: ['English', 'Indonesian'],
    availability: [
      {
        'day': 'Friday',
        'times': ['19:00', '20:00', '21:00']
      },
    ],
    gameSelections: [
      GameConfig(
        gameName: 'Dota 2',
        preferences: ['Carry', 'Midlaner', 'Hard Support', 'Soft
   Support'],
        playstyle: 'Experienced carry with map awareness'
      ),
    ],
    kudos: 4.8,
  )
```

Kode 3.1: Data *dummy player*

Kelas *Player* berfungsi sebagai *model* data pemain yang menyimpan informasi profil pengguna, di mana *field* id digunakan sebagai identitas unik dan *nickname* sebagai nama yang ditampilkan pada aplikasi. *Field languages* menyimpan daftar bahasa yang dipilih pemain, sedangkan beberapa *field* seperti *steamFriendcode, pubgFriendcode, mhwFriendcode*, dan *moleFriendcode* digunakan untuk menyimpan kode pertemanan sesuai dengan *game*. *Field availability* berisi data ketersediaan waktu bermain dalam bentuk hari dan jam, sementara *gameSelections* menyimpan konfigurasi *game* berupa nama *game*, preferensi *role* atau senjata, serta gaya bermain pemain. Terakhir, *field kudos* digunakan sebagai nilai penilaian pemain dalam bentuk *rating*.

M.2 Data Dummy Room Recruiting

Data *dummy room recruiting* digunakan untuk mensimulasikan kondisi *room* yang sedang membuka perekrutan pemain. Potongan kode pada Kode 3.2 menunjukkan kelas *RoomRecruitingModel* yang berisi data *room* dengan status *recruiting*.

```
final List<RoomRecruitingModel> simulatedRoomRecruiting = [
  RoomRecruitingModel(
    id: '1',
    creatorId: '1',
    roomName: 'Dota Late Night',
    description: 'Fun match Dota malam-malam',
    roomLanguages: ['Indonesian', 'English'],
    game: 'Dota 2',
    availableSlots: ['Position 3', 'Position 4', 'Position 2'],
    preferredSchedules: [
      {'day': 'Friday', 'timeRange': '19:00 - 21:00'},
      {'day': 'Saturday', 'timeRange': '19:00 - 21:00'},
    ],
    maxPlayer: 5,
    acceptedPlayers: ['3'],
    recruitPlayers: ['6', '8'],
    status: 0,
  )
```

Kode 3.2: Data *dummy room recruit*

Kelas *RoomRecruitingModel* menyimpan informasi dasar sebuah *room*, seperti id sebagai identitas *room* dan *creatorId* sebagai penanda pemain pembuat *room*, sedangkan *field roomName* dan *description* digunakan untuk menampilkan nama serta deskripsi singkat *room* pada kartu *UI*. *Field roomLanguages* menunjukkan bahasa yang digunakan dalam *room*, sementara *game* menyimpan jenis *game* yang dimainkan. *Field availableSlots* digunakan untuk menampilkan posisi atau *role* yang masih dibutuhkan, dan *preferredSchedules* berisi jadwal bermain yang diinginkan oleh pembuat *room*. Selain itu, *field maxPlayer* menentukan jumlah maksimal pemain, acceptedPlayers menyimpan daftar pemain yang sudah diterima, *recruitPlayers* berisi pemain yang sedang direkrut, serta status digunakan untuk menandai bahwa *room* berada pada kondisi *recruiting*.

M.3 Data Dummy Room Active

Data *dummy room active* digunakan untuk mensimulasikan *room* yang sedang berjalan atau aktif dimainkan. Potongan kode pada Kode 3.3 menunjukkan kelas *RoomActiveModel* beserta data *dummy room* aktif yang digunakan dalam aplikasi.

```
final List<RoomActiveModel> simulatedRoomActive = [
  RoomActiveModel(
    id: '2',
    creatorId: '1',
    roomName: 'Push Rank ML',
    description: 'Push bareng sampe Mythic!',
    roomLanguages: ['Indonesian'],
    game: 'Mobile Legends',
    availableSlots: ['Mid Lane', 'Gold Lane'],
    preferredSchedules: [
      {'day': 'Tuesday', 'timeRange': '16:00 - 19:00'},
      {'day': 'Thursday', 'timeRange': '16:00 - 19:00'},
    ],
    maxPlayer: 5,
    acceptedPlayers: [
      PlayerConfig(playerId: 1, playerVote: false),
      PlayerConfig(playerId: 4, playerVote: true),
      PlayerConfig(playerId: 5, playerVote: true),
      PlayerConfig(playerId: 8, playerVote: true),
      PlayerConfig(playerId: 6, playerVote: true),
    ],
    status: 1,
  )
```

Kode 3.3: Data *dummy room active*

Kelas *RoomActiveModel* memiliki struktur yang mirip dengan *room recruiting*, namun perbedaannya terletak pada *field acceptedPlayers* yang menggunakan objek *PlayerConfig* untuk menyimpan informasi pemain beserta status *voting* (*playerVote*) sebagai simulasi fitur kesiapan pemain. *Field* status digunakan untuk menandai bahwa *room* berada pada kondisi aktif dan sedang berjalan. Selain itu, terdapat properti tambahan seperti *acceptedPlayerCount* dan *remainingSlots* yang berfungsi untuk menghitung jumlah pemain yang sudah bergabung serta sisa *slot* yang tersedia secara otomatis. Data *dummy* ini digunakan untuk menampilkan antarmuka *room* aktif beserta status masing-masing pemain di dalamnya.

M.4 Data Dummy Room Closed

Data *dummy room closed* digunakan untuk mensimulasikan *room* yang telah selesai atau ditutup. Potongan kode pada Kode 3.4 menunjukkan kelas *simulatedRoomClosed* yang berisi data *room* dengan status *closed*.

```
final List<RoomClosedModel> simulatedRoomClosed = [
  RoomClosedModel(
    id: '10',
    creatorId: '5',
    roomName: 'ML Weekend Squad',
    description: 'Santai tapi menang',
    roomLanguages: ['Indonesian'],
    game: 'Mobile Legends',
    availableSlots: ['Jungle', 'Gold Lane'],
    preferredSchedules: [{'day': 'Sunday', 'timeRange': '15:00 - 18:00'}],
    maxPlayer: 5,
    acceptedPlayers: ['1', '6', '5', '8', '9'],
    status: 2,
  ),
  RoomClosedModel(
    id: '11',
    creatorId: '1',
    roomName: 'PUBG For Fun',
    description: 'Sampai Chicken Dinner',
    roomLanguages: ['Indonesian'],
    game: 'PUBG',
    availableSlots: [],
    preferredSchedules: [{'day': 'Monday', 'timeRange': '19:00 - 21:00'}],
    maxPlayer: 4,
    acceptedPlayers: ['1', '4', '2', '7'],
    status: 2,
  )
```

Kode 3.4: Data *dummy room closed*

Kelas *RoomClosedModel* digunakan untuk menyimpan informasi *room* yang sudah tidak aktif. *Field* seperti *roomName, description*, dan *game* digunakan untuk menampilkan informasi umum *room* sebagai riwayat aktivitas pengguna. *Field* acceptedPlayers berisi daftar pemain yang telah bergabung dan mengikuti *room* tersebut hingga selesai, sedangkan *field* status digunakan sebagai penanda bahwa *room* berada pada kondisi *closed* dan hanya ditampilkan sebagai *history*.

M.5 Data Dummy Chat Message

Data *dummy chat message* digunakan untuk mensimulasikan percakapan antar pemain di dalam *room*. Potongan kode pada Kode 3.5 menunjukkan kelas *ChatMessageModel* yang berisi pesan-pesan *chat* berdasarkan *room*.

```
final List<ChatMessageModel> simulatedChatMessages = [
  ChatMessageModel(
    id: '1',
    roomId: '2',
    senderId: '1',
    message: 'Gas push rank malam ini bro!',
    messageDate: DateTime(2025, 10, 30, 21, 10),
  ),
  ChatMessageModel(
    id: '2',
    roomId: '2',
    senderId: '4',
    message: 'Siap! Jam berapa mulai?',
    messageDate: DateTime(2025, 10, 30, 21, 11),
  ),
  ChatMessageModel(
    id: '3',
    roomId: '2',
    senderId: '5',
    message: 'Kalau bisa jam 8 ya, biar gak bentrok',
    messageDate: DateTime(2025, 10, 30, 21, 13),
  )
```

Kode 3.5: Data *dummy chat message*

Kelas *ChatMessageModel* digunakan untuk menyimpan dan merepresentasikan data pesan *chat* yang dikirim oleh pemain di dalam sebuah *room*. *Field* id berfungsi sebagai identitas unik setiap pesan, sedangkan roomId digunakan untuk mengelompokkan pesan berdasarkan *room* tempat *chat* berlangsung. *Field senderId* menyimpan identitas pengirim pesan. *Field message* berisi isi pesan yang ditampilkan pada antarmuka *chat*, sementara *messageDate* menyimpan informasi waktu pengiriman pesan agar urutan percakapan dapat ditampilkan secara kronologis. Data *dummy chat* ini memungkinkan fitur *chat* pada aplikasi tetap dapat dikembangkan, diuji, dan ditampilkan secara visual meskipun sistem *realtime* dan integrasi *backend chat* belum tersedia.

N Logika Matchmaking

matchmaking merupakan proses pencocokan pemain dengan *room* berdasarkan preferensi dan kebutuhan yang dimiliki. Proses ini menerapkan penyaringan *room*, dengan hanya memperbolehkan *room* yang berstatus *recruiting* dengan *game* yang sesuai, kemudian dilanjutkan dengan aturan penilaian yang mencakup kesesuaian bahasa, jadwal bermain, serta kecocokan slot atau *role* dengan bobot skor yang berbeda. Pada tahap pengembangan, data *dummy* digunakan untuk menguji logika *matchmaking* sebelum sistem terhubung dengan *backend*.

Kelas *MatchmakingService*, seperti ditunjukkan pada Kode 3.6, berfungsi sebagai layanan yang menangani proses *matchmaking* antara pemain dan *room*. Kelas ini menyediakan fungsi yang menerima dua parameter utama, yaitu data pemain (*Player*) dan daftar *room* (*RoomRecruitingModel*). Hasil dari fungsi tersebut berupa daftar *room* yang telah melalui proses penilaian dan diberikan skor kecocokan berdasarkan kriteria yang telah ditentukan.

```
class MatchmakingService {
  static List<Map<String, dynamic>> findBestRoomsForPlayer(
    Player player,
    List<RoomRecruitingModel> rooms,
  )
```

Kode 3.6: *Matchmaking*

Pada tahap awal proses *matchmaking*, sebagaimana diperlihatkan pada Kode 3.7, sistem melakukan penyaringan awal dengan hanya memproses *room* yang berstatus *recruiting* (status = 0). Selain itu, sistem juga mengecek kecocokan *game* antara pemain dan *room*, di mana sebuah *room* hanya akan dipertimbangkan apabila *game* yang dipilih oleh pemain sesuai dengan *game* yang tersedia pada *room*. Tahap penyaringan ini bertujuan untuk memastikan bahwa hanya *room* yang relevan dan sesuai kebutuhan pemain yang dilanjutkan ke proses penilaian berikutnya.

```
if (room.status != 0) continue;
final bool isGameMatched = player.gameSelections.any(
  (game) => game.gameName.toLowerCase() == room.game.toLowerCase()
    ,
);
if (!isGameMatched) {
  continue;
}
```

Kode 3.7: *Game match*

Aturan penilaian yang ditunjukkan pada Kode 3.8 digunakan untuk menentukan tingkat kecocokan antara pemain dan *room* berdasarkan beberapa kriteria utama. Kriteria tersebut meliputi kesesuaian bahasa, kecocokan jadwal bermain, serta kecocokan slot atau *role*. Setiap kriteria yang terpenuhi akan menambah skor *room* dengan bobot yang berbeda. Kesesuaian bahasa bernilai 2 poin, kecocokan jadwal bernilai 3 poin, dan kecocokan slot atau *role* memiliki bobot tertinggi sebesar 5 poin.

```
if (player.languages.any(
  (lang) => room.roomLanguages
    .map((e) => e.toLowerCase())
    .contains(lang.toLowerCase()),
)) {
  score += 2;
}
if (_isScheduleOverlap(player.availability, room.
    preferredSchedules)) {
  score += 3;
}
if (_isSlotMatch(player.gameSelections, room)) {
  score += 5;
}
```

Kode 3.8: *Scoring match criteria*

Setelah seluruh *room* diproses dan diberi skor, daftar *room* akan diurutkan secara menurun berdasarkan nilai skor, sebagaimana ditunjukkan pada Kode 3.9. Pengurutan ini menempatkan *room* dengan tingkat kecocokan tertinggi pada urutan teratas. Dengan demikian, *room* tersebut direkomendasikan terlebih dahulu kepada pemain.

```
matchedRooms.sort((a, b) => b['score'].compareTo(a['score']));
return matchedRooms;
```

Kode 3.9: *Final match*

Secara keseluruhan, logika *matchmaking* yang diterapkan terdiri dari tahapan penyaringan awal dan pemberian skor kecocokan untuk menghasilkan rekomendasi *room* yang sesuai dengan preferensi pemain. Proses ini memastikan bahwa *room* yang direkomendasikan telah disesuaikan dengan kebutuhan dan preferensi pemain. Pendekatan tersebut menjadikan proses *matchmaking* lebih terstruktur, relevan, serta mudah dikembangkan pada tahap integrasi dengan *backend* di masa mendatang.

O Pengujian

Pengujian pada aplikasi ini menggunakan metode *Black Box Testing* yang berfokus pada pengamatan hasil *input* dan *output* tanpa memperhatikan struktur kode program. Metode ini dilakukan pada tahap akhir pengembangan untuk memastikan aplikasi dapat berfungsi sesuai dengan kebutuhan yang telah ditentukan. Pengujian dilakukan melalui antarmuka aplikasi dengan mensimulasikan aksi pengguna dan mengevaluasi respons sistem yang dihasilkan. Pendekatan ini sesuai untuk aplikasi yang menitikberatkan pada fungsionalitas *frontend* dan interaksi pengguna [9].

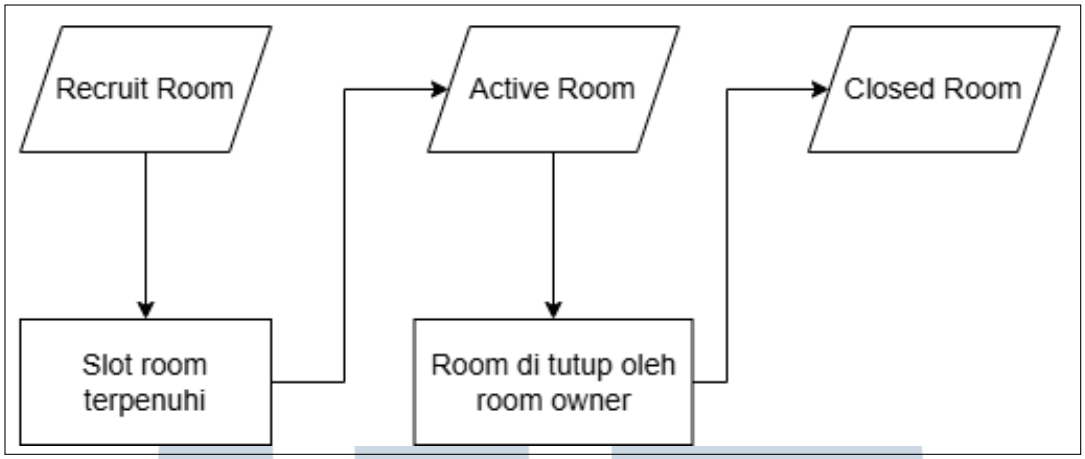
Black Box Testing tidak menuntut penguji untuk memiliki kemampuan dalam menulis atau memahami kode program. Dengan karakteristik tersebut, metode ini dapat dilakukan oleh berbagai pihak yang terlibat dalam proses evaluasi aplikasi. Hal ini memungkinkan pengujian dilakukan dari sudut pandang pengguna secara langsung. Oleh karena itu, *Black Box Testing* dinilai efektif untuk memvalidasi fungsi aplikasi secara umum [9].

Salah satu teknik *Black Box Testing* yang digunakan adalah *State Transition Testing* [10]. Teknik ini diterapkan pada fitur *Rooms* karena memiliki perubahan status yang jelas dan berurutan. Perubahan status tersebut meliputi kondisi *Recruiting, Active,* dan *Closed* yang memengaruhi fungsi serta tampilan aplikasi. Dengan demikian, *State Transition Testing* membantu memastikan perpindahan antar *state* berjalan sesuai alur yang diharapkan.

Sementara itu, fitur lain seperti *Login, First-Time Setup, Discover, Add Room,* dan *Profile Settings* diuji menggunakan *Black Box Testing* berbasis skenario pengguna. Fitur-fitur tersebut tidak memiliki perubahan *state* yang kompleks, melainkan berfokus pada *input*, navigasi, dan validasi data. Oleh karena itu, pengujian dilakukan tanpa menggunakan diagram *State Transition* agar tetap efisien. Pendekatan ini memastikan seluruh fungsi aplikasi tetap teruji secara menyeluruh sesuai karakteristik masing-masing fitur.

O.1 Diagram State Transition Rooms

Pengujian menggunakan teknik *State Transition Testing* pada fitur *Rooms* divisualisasikan melalui diagram *State Transition* yang ditunjukkan pada Gambar 3.13. Diagram tersebut digunakan sebagai acuan untuk menguji perpindahan status *room* berdasarkan aksi pengguna selama penggunaan aplikasi.



Gambar 3.13. Diagram *state transition rooms*

Diagram *State Transition* pada fitur *Rooms* menggambarkan alur perubahan status *room* dari awal hingga akhir. Pada kondisi awal, *room* berada pada status *Recruiting* saat masih mencari pemain. Setelah seluruh *slot* terpenuhi dan permainan dimulai, status *room* berubah menjadi *Active*. Selanjutnya, ketika permainan selesai atau *room* ditutup oleh pengguna, status *room* akan berpindah ke *Closed*. Setiap transisi status tersebut digunakan sebagai dasar dalam melakukan pengujian untuk memastikan sistem menampilkan antarmuka dan fungsi yang sesuai.

O.2 Hasil Pengujian

Hasil pengujian menunjukkan bahwa seluruh fitur aplikasi telah berfungsi sesuai dengan kebutuhan yang ditetapkan. Pengujian pada fitur *Rooms* memastikan bahwa setiap transisi status dari *Recruiting* ke *Active*, serta dari *Active* ke *Closed*, berjalan dengan benar dan menampilkan antarmuka yang sesuai. Sementara itu, pengujian pada halaman lain menunjukkan bahwa proses input, navigasi, dan validasi data dapat dilakukan tanpa kendala.

Berdasarkan hasil tersebut, dapat disimpulkan bahwa aplikasi telah memenuhi aspek fungsionalitas *frontend* dan siap digunakan sesuai dengan tujuan pengembangannya. Dengan menggunakan metode *Black Box Testing* serta teknik *State Transition Testing* pada fitur kritis, proses pengujian dinilai telah dilakukan secara menyeluruh dan efektif.

3.4 Kendala dan Solusi yang Ditemukan

Pelaksanaan kerja magang dalam pengembangan aplikasi *mobile* tidak terlepas dari berbagai kendala yang muncul selama proses pengerjaan. Kendala tersebut berkaitan dengan aspek teknis pengembangan *frontend*, pengelolaan struktur kode, serta ketersediaan data pendukung aplikasi. Setiap kendala yang ditemukan perlu dianalisis agar solusi yang tepat dapat diterapkan secara sistematis. Oleh karena itu, pada bagian ini dijabarkan kendala-kendala yang dihadapi beserta solusi yang diterapkan untuk mengatasi permasalahan tersebut.

3.4.1 Kendala

Selama pelaksanaan kerja magang di PT Hexa Global Teknologi, beberapa kendala ditemukan dalam proses pengembangan *frontend* aplikasi *GameSynce*. Kendala utama yang terjadi pada tahap awal adalah penggunaan bahasa pemrograman dan *framework* yang belum sepenuhnya dikuasai, yaitu Dart dan Flutter. Kondisi tersebut menyebabkan proses pengembangan membutuhkan waktu tambahan, khususnya dalam memahami konsep dasar Flutter, struktur *widget*, serta alur pengembangan aplikasi *mobile*. Selain itu, proses adaptasi terhadap pola pengembangan yang diterapkan pada proyek juga memerlukan pemahaman yang bertahap.

Pengembangan aplikasi yang difokuskan pada sisi *frontend* juga menimbulkan kendala terkait ketersediaan data. Pada tahap pengembangan, data dari *backend* belum tersedia sehingga menyulitkan proses penampilan data secara dinamis pada antarmuka pengguna. Tanpa adanya data yang terstruktur, pembuatan tampilan berisiko dilakukan secara manual dan berulang. Kondisi ini dapat menyebabkan inkonsistensi data serta menyulitkan proses pengujian tampilan aplikasi.

Selain itu, kendala lain yang dihadapi selama pelaksanaan kerja magang adalah penerapan sistem kerja *Work From Home (WFH)* secara penuh. Kondisi ini menyebabkan proses komunikasi dan koordinasi dengan *supervisor* tidak selalu dapat dilakukan secara langsung. Beberapa diskusi teknis terkait implementasi fitur dan alur aplikasi harus dilakukan melalui media komunikasi daring, sehingga terkadang membutuhkan waktu lebih lama untuk mencapai pemahaman yang sama. Selain itu, keterbatasan interaksi secara tatap muka juga menuntut untuk lebih mandiri dalam memahami kebutuhan pengembangan aplikasi.

3.4.2 Solusi

Untuk mengatasi kendala pada tahap awal terkait penggunaan bahasa pemrograman dan *framework* baru, dilakukan proses pembelajaran secara bertahap melalui dokumentasi resmi *Flutter*. Selain itu, contoh implementasi yang tersedia pada proyek dipelajari untuk memahami pola penggunaan *widget* dan struktur aplikasi. Arahan yang diberikan oleh *supervisor* juga dijadikan acuan dalam memahami alur pengembangan, struktur proyek, serta penerapan arsitektur yang digunakan. Dengan pendekatan tersebut, proses adaptasi terhadap teknologi yang digunakan dapat berjalan dengan lebih terarah.

Untuk mengatasi kendala terkait ketersediaan data *backend*, digunakan data *dummy* yang disusun menyerupai struktur data yang direncanakan. Penggunaan data *dummy* memungkinkan penampilan data pada antarmuka dilakukan secara dinamis tanpa perlu menuliskan data secara manual pada setiap komponen *UI*. Dengan pendekatan ini, konsistensi data dapat terjaga dan proses pengujian tampilan aplikasi menjadi lebih efektif. Selain itu, penggunaan data *dummy* juga mempermudah proses integrasi dengan *backend* pada tahap pengembangan selanjutnya.

Untuk mengatasi kendala komunikasi akibat penerapan sistem kerja *Work From Home (WFH)*, dilakukan pemanfaatan media komunikasi daring secara optimal. Diskusi dan koordinasi dengan *supervisor* serta anggota tim dilakukan melalui pertemuan *online* dan komunikasi tertulis menggunakan platform yang telah ditentukan. Selain itu, setiap arahan dan hasil diskusi dicatat dalam bentuk dokumentasi agar dapat dijadikan acuan dalam proses pengembangan aplikasi. Pendekatan ini membantu meminimalkan kesalahpahaman, meningkatkan kejelasan tugas, serta menjaga kelancaran proses pengembangan meskipun tidak dilakukan secara tatap muka langsung.