BAB 3
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

Kedudukan atau posisi yang diberikan pada pelaksanaan kerja magang di PT Jaya Santoso Teknologi adalah sebagai *Back End Developer Intern* pada divisi *IT Development* yang berada di bawah koordinasi langsung *Chief Technology Officer* (CTO). Divisi *IT Development* memiliki tanggung jawab utama dalam membangun dan mengembangkan sistem *backend* dari produk digital perusahaan, termasuk mengimplementasikan berbagai fitur dan layanan berbasis API yang dibutuhkan oleh tim *Product Development* serta tim *Front End Developer*. Divisi ini bekerja erat dengan divisi lain seperti *Product Owner* dan *UI/UX Designer* untuk memastikan bahwa setiap pengembangan sistem dilakukan secara tepat dan sesuai kebutuhan bisnis.

Selama masa magang, komunikasi dilakukan baik secara langsung maupun tidak langsung menggunakan *WhatsApp* untuk diskusi harian secara cepat, serta *Discord* yang digunakan sebagai ruang koordinasi bersama untuk menyampaikan laporan harian dan pembagian tugas. Selain menggunakan *WhatsApp*, koordinasi teknis dilakukan menggunakan *platform GitHub* sebagai sistem kendali versi kode sumber proyek. Untuk menjaga kestabilan sistem dan proses kolaborasi yang rapi, tim menggunakan metode *branching strategy*, di mana setiap fitur atau perbaikan dibuat dalam *branch* terpisah yang kemudian akan digabungkan (*merge*) ke *branch* utama setelah di-*review*. *Branch* utama dibagi menjadi dua yaitu `development` dan `production` agar memisahkan antara proses pengembangan dan versi akhir yang siap digunakan oleh pengguna.

Pengawasan dan evaluasi kinerja magang dilakukan secara berkala oleh supervisi setiap dua minggu sekali. Dalam sesi ini, supervisor akan memantau progres proyek, memberikan *feedback*, dan memastikan bahwa tugas yang diberikan berjalan sesuai dengan ekspektasi perusahaan serta standar teknis yang berlaku. Melalui sistem komunikasi dan koordinasi yang terstruktur ini, proses pengembangan dapat berjalan lebih efektif meskipun pelaksanaan magang dilakukan secara *Work From Home (WFH)*.

3.2 Tugas yang Dilakukan

Selama pelaksanaan kerja magang di perusahaan PT Jaya Santoso Teknologi, tugas utama yang diberikan adalah merancang *backend* berbasis *API* untuk sistem undangan digital, khususnya pada *templating service*. Aplikasi ini dikembangkan menggunakan *Framework Spring Boot* dan menggunakan bahasa pemrograman Java. *Templating service* ini ditujukan untuk merender halaman undangan secara dinamis berdasarkan konten yang diambil dari *content service*. Selain itu, *service* ini juga mengimplementasikan *Redis Cache* agar mengoptimalkan proses *rendering* dan menghindari *overload request* ke *content service*. Setelah sistem berjalan dengan stabil, dilakukan juga *deployment* ke *Heroku* sebagai bagian dari proses pengembangan dan optimasi sistem. Berikut adalah garis besar tugas yang dilakukan, antara lain:

1. Mempelajari pengembangan aplikasi menggunakan *Framework Spring Boot*
2. Perancangan dan Integrasi *Templating Service*
3. Implementasi *Caching* dengan *Redis Cache*
4. Penanganan dan *Debugging Error Redis*
5. *Bug Testing* dan *Deployment* ke *Heroku*

3.3 Uraian Pelaksanaan Magang

Penjelasan mengenai pekerjaan yang dilakukan selama masa kerja magang pada PT Jaya Santoso Teknologi dapat dilihat pada Tabel 3.1.

Tabel 3.1. Pekerjaan yang dilakukan tiap minggu selama pelaksanaan kerja magang

Minggu Ke -	Pekerjaan yang dilakukan
1	Pengenalan lingkungan kerja, tools yang digunakan *(IntelliJ, Git, Spring Boot)*, serta memahami alur *development*.
2	Pembuatan struktur awal proyek *templating-service* dan memahami alur integrasi dengan *content-service*.
3	Mendesain dan mulai membuat *utility converter* untuk mengubah *link Google Drive* agar bisa direnderi di halaman undangan.

Minggu Ke -	Pekerjaan yang dilakukan
4	Pengembangan fungsi dasar *converter*, seperti mengekstrak ID dari *URL* dan mengembalikan format *embed*.
5	Integrasi *converter* dengan *templating-service*, serta pengujian awal untuk memastikan video tampil dengan benar di HTML.
6	Mulai perencanaan implementasi *caching* untuk meningkatkan performa *render* template HTML.
7	Diskusi dan riset strategi *caching (Redis, Caffeine, Ehcache)*, serta memutuskan untuk menggunakan *Redis*.
8	Implementasi awal konfigurasi *Redis (cache manager, TTL, serializer)* dalam *proyek templating*.
9	Implementasi konfigurasi *Redis (host, port, auth)* serta *custom cache manager* dengan *TTL dan serializer*.
10	Pengujian *end-to-end* dengan *Redis*: validasi *cache hit/miss, TTL, dan fallback ke content service*.
11	Penambahan *endpoint* untuk menghapus *cache* berdasarkan *template ID*, serta pengujian *cache invalidation*.
12	Pengujian dan *debugging* terhadap fitur *caching*. Memastikan *template* yang sama tidak di-*render* ulang jika sudah ada di *cache*.
13	Pembuatan *endpoint* baru untuk *direct render* dengan *driveLink* (tanpa *template ID* untuk menguji *Redis cache* dan fitur *rendering*).
14	*Debugging* dan perbaikan *bug* saat mengakses *endpoint* baru. Menyesuaikan struktur respon dan validasi parameter.
15	*Deployment* awal *templating service* ke *staging Heroku*.
16	Menangani error saat *deployment*, khususnya error pada *Redis* (koneksi, *serializer, dan TTL*).
17	Menonaktifkan verifikasi *peer certificate*, khusus untuk *testing/dev* untuk mengecek apakah *SSL* memang aktif, supaya kode *disablePeerVerification()* tidak dipanggil sembarangan.
18	*Testing* setelah *deployment*. Verifikasi semua *endpoint* berjalan normal dan *cache* berfungsi dengan benar.
19	Evaluasi struktur proyek dan pengujian *end-to-end* integrasi *templating service* dengan *service* lain.

Minggu Ke -	Pekerjaan yang dilakukan
20	Pengujian integrasi *templating service* dengan *service* lain secara menyeluruh, serta evaluasi performa *caching*.
21	Penambahan dokumentasi teknis akhir dan perapihan struktur kode untuk keperluan *handover* dan review tim.
22	Pembuatan laporan akhir magang serta review hasil kerja dan kontribusi pada proyek templating service.

Masa magang dimulai dengan mempelajari struktur dasar pengembangan aplikasi menggunakan *framework Spring Boot*. Fokus utama adalah memahami arsitektur berlapis, yaitu pembagian tugas antara *Controller*, *Service*, dan *Repository*. Selain itu, cara membuat REST API menggunakan anotasi seperti `@RestController`, `@GetMapping`, dan `@PostMapping` juga dipelajari. Konfigurasi *dependency* juga dilakukan melalui *file* `pom.xml` dengan *Maven*, serta penggunaan `application.properties` untuk pengaturan *database, port, dan environment*. Konsep *dependency injection* menggunakan anotasi `@Autowired`, serta penerapan *logging* dan *error handling* dengan `LoggerFactory` dan `@ControllerAdvice` ditelusuri lebih dalam.

Setelah memahami dasar *Spring-Boot*, proyek *internal dashboard* mulai dikerjakan, yaitu aplikasi *backend* yang digunakan untuk memantau dan mengelola data undangan. Salah satu fitur utama yang dikerjakan dari *dashboard* ini adalah *templating service*, sebuah layanan yang berfungsi untuk me-*render* halaman HTML berdasarkan data dari *content service* dan *ID template* tertentu. Layanan ini dikembangkan dari awal, termasuk pembuatan endpoint utama `/template/templateId` yang mengembalikan hasil *render* ke web utama dan juga disimpan ke *Redis cache* untuk efisiensi performa. Dengan adanya fitur ini, tampilan undangan dapat dihasilkan secara dinamis dan terintegrasi dengan sistem utama.

Pada minggu ke-3 dan 4 pelaksanaan kerja magang, mulai dibuat *utility converter*. *Converter* ini didesain khusus untuk mengubah *link Google Drive* yang menyimpan HTML agar bisa direnderer ke halaman undangan. Hal ini dibutuhkan karena *link Google Drive* yang *default* tidak bisa langsung di-*render*. Ketika *link Google Drive* diakses, *Google* akan menampilkan *preview viewer*, bukan isi *file* secara langsung. Sehingga, *browser* tidak menganggap ini sebagai *file* HTML yang siap di-*render*, melainkan tampilan *Google Drive*.

Utility converter yang telah dikembangkan sebelumnya diintegrasikan

langsung ke dalam modul *templating service* pada minggu ke-5. Tujuan utama integrasi ini adalah agar sistem dapat secara otomatis mengonversi *link Google Drive* menjadi *raw file* yang dapat diakses dan di-*render* oleh *browser* atau server. *Converter* ini secara otomatis mengambil *file* ID dari *link Google Drive* dan membentuk *URL* dengan format khusus, sehingga konten HTML dapat diambil dan ditampilkan tanpa melalui tampilan *preview Google Drive*.

Setelah integrasi berhasil, dilakukan pengujian awal untuk memastikan bahwa isi *file* HTML dapat di-*render* dengan benar. Fokus pengujian adalah pada konten multimedia, khususnya video, karena banyak template undangan yang menggunakan video sebagai elemen pembuka. Pengujian ini mencakup:

- Validasi apakah video ditampilkan dengan benar dalam tag `<video>` atau `<iframe>`.
- Memastikan *link* sumber video (misalnya dari *Google Drive* atau *YouTube*) tetap dapat diakses.
- Mengecek apakah terdapat eror *CORS*, *file not found*, atau *layout* yang rusak.

Pada minggu ke-6 dan 7, performa *render* HTML ditingkatkan dengan menerapkan *caching*. *Render* HTML adalah proses yang cukup berat karena melibatkan pengambilan data dari beberapa sumber, lalu diproses dan dikembalikan sebagai *output* HTML. Untuk mengurangi waktu respon dan beban sistem, dirancang strategi *caching* agar hasil *render* dapat disimpan sementara dan digunakan kembali saat permintaan serupa masuk.

Riset dan diskusi bersama tim pengembang *backend* dilakukan untuk memilih solusi *caching* yang paling sesuai dengan kebutuhan sistem *templating service*. Tujuan utama dari penerapan *caching* ini adalah untuk meningkatkan efisiensi dan kecepatan dalam proses *rendering* HTML, serta mengurangi beban permintaan berulang ke *Content Service*. Dalam proses pemilihan teknologi *cache*, terdapat tiga kandidat utama yang dipertimbangkan, yaitu:

1. *Redis*, sebuah *in-memory data store* yang terkenal karena kecepatan dan performanya yang tinggi. *Redis* mendukung operasi baca dan tulis yang sangat cepat, serta dapat digunakan dalam skenario distribusi dan skala besar. *Redis* juga memiliki ekosistem yang kuat serta dukungan *cloud* seperti *Redis Cloud*, yang kompatibel dengan layanan *hosting* seperti *Heroku*.

2. *Caffeine*, merupakan pustaka *caching* lokal berbasis Java yang ringan dan efisien, cocok untuk digunakan pada aplikasi berskala kecil atau jika cache hanya dibutuhkan secara lokal dalam *instance* server. Namun, kelemahannya adalah tidak mendukung skenario distribusi dan tidak bisa diakses oleh *instance* lain dalam sistem yang terdistribusi.

3. *Ehcache*, salah satu pustaka *cache* Java yang sudah matang dan banyak digunakan dalam berbagai proyek Java. *Ehcache* mudah dikonfigurasi dan mendukung integrasi dengan Spring. Meskipun demikian, performa dan skalabilitasnya masih di bawah *Redis* untuk kasus penggunaan yang memerlukan ketersediaan tinggi dan distribusi lintas layanan.

Setelah mempertimbangkan kebutuhan sistem yang memerlukan cache yang dapat diakses oleh banyak layanan serta mendukung skalabilitas dan performa tinggi, maka *Redis* dipilih sebagai solusi *caching* utama. Redis merupakan *in-memory data store* yang berjalan secara independen di luar aplikasi utama dan mendukung skala horizontal. Hal ini sangat penting karena layanan *templating service* dirancang untuk melayani permintaan pengguna dari berbagai undangan digital secara bersamaan, sehingga dibutuhkan sistem *cache* yang mampu bekerja lintas instansi aplikasi dan menjaga konsistensi data di berbagai lingkungan produksi. Selain itu, *Redis* memiliki performa yang sangat tinggi dalam melakukan operasi baca dan tulis data secara *real-time*. Ini memberikan keunggulan dalam hal kecepatan pengambilan data HTML hasil *render* yang sudah disimpan sebelumnya, sehingga dapat mempercepat waktu respon sistem dan mengurangi beban pada *content service*. *Redis* tidak hanya mampu menyimpan hasil *render* HTML secara efisien, tetapi juga mudah diintegrasikan dengan *Spring-Boot* melalui dukungan anotasi seperti `@Cacheable`. Oleh karena itu, *Redis* dianggap sebagai solusi paling tepat untuk mendukung arsitektur sistem *templating service* yang stabil dan responsif.

Jika dibandingkan dengan *Caffeine dan Ehcache* yang bersifat *embedded* dalam aplikasi Java, *Redis* memiliki keunggulan dalam hal fleksibilitas dan visibilitas. *Redis* dapat diakses dan dimonitor secara eksternal melalui berbagai alat bantu, serta mendukung berbagai bahasa pemrograman lain jika diperlukan integrasi lintas sistem. Ini menjadikannya lebih cocok untuk kebutuhan *caching* dalam arsitektur berbasis mikro-servis atau sistem terpisah seperti yang digunakan pada proyek ini. Dengan mempertimbangkan kebutuhan skalabilitas, performa, dan fleksibilitas dalam lingkungan produksi yang dijalankan di *Heroku*, *Redis* menjadi

pilihan yang paling sesuai untuk diimplementasikan sebagai sistem *caching* pada layanan *templating service*.

Pada minggu ke-8 dan 9, implementasi awal konfigurasi *Redis* ke dalam proyek *Spring Boot* kemudian dilakukan. Tahapan yang dilakukan pertama kali adalah menambahkan *dependency Redis* di `pom.xml`. Lalu mengatur koneksi dasar seperti *host, port*, dan autentikasi *Redis*. Setelah itu dibuat juga konfigurasi *custom* untuk `CacheManager`, termasuk pengaturan *TTL (time-to-live)* untuk setiap *cache entry*, serta pemilihan *serializer* agar objek Java bisa disimpan dan dibaca kembali dari *Redis*.

Setelah konfigurasi selesai, pada minggu ke-10 pengujian dilakukan untuk memastikan *caching* bekerja sesuai harapan. Pengujian ini mencakup:

- Validasi *cache hit* dan *miss*: apakah permintaan yang sama kedua kali akan menggunakan *cache*, dan apakah permintaan pertama akan masuk proses *render* normal.
- Validasi *TTL*: apakah data *cache* akan kadaluarsa sesuai waktu yang ditentukan.
- *Fallback logic*: Mekanisme ini dirancang untuk memastikan sistem tetap dapat merespons permintaan meskipun data yang diminta belum tersedia dalam *cache* (*cache miss*). Ketika sistem menerima permintaan render HTML dan mendeteksi bahwa data dengan `templateID` tertentu belum ada di dalam *cache*, maka sistem akan secara otomatis menjalankan proses pengambilan data dari *content service*. Setelah data berhasil diambil, sistem akan melakukan proses *rendering* untuk menghasilkan halaman HTML. Hasil render ini tidak hanya dikembalikan langsung ke pengguna, tetapi juga disimpan ke dalam *cache* (misalnya Redis) untuk keperluan permintaan berikutnya. Dengan pendekatan ini, setiap permintaan pertama tetap dapat dilayani secara normal, dan permintaan selanjutnya dapat dilayani dengan lebih cepat melalui data *cache*. Mekanisme *fallback* ini juga disertai dengan penanganan error apabila *content service* gagal merespons, seperti memberikan pesan kesalahan yang informatif atau menampilkan halaman alternatif (template cadangan), guna menjaga pengalaman pengguna tetap optimal.

Pada minggu ke-11, dibangun sebuah *endpoint* baru yang berfungsi untuk menghapus *cache* berdasarkan `templateId`. Tujuannya adalah untuk memberikan

fleksibilitas dalam proses *cache invalidation*, terutama saat terjadi perubahan data konten atau *template*. Setelah *endpoint* selesai dibuat, dilakukan pengujian untuk memastikan bahwa *cache* benar-benar terhapus dan sistem melakukan *render* ulang saat data diminta kembali.

Pada minggu ke-12, fokus ditempatkan pada pengujian dan *debugging* fitur *caching* yang sebelumnya telah diimplementasikan. Pengujian ini dilakukan untuk memastikan bahwa saat sebuah *template* sudah pernah di-*render* dan hasilnya sudah disimpan dalam *Redis cache*, maka permintaan selanjutnya akan langsung menggunakan data *cache* tanpa *render* ulang. Hal ini penting untuk memastikan sistem bekerja dengan optimal dan tidak membebani proses *render* yang tidak diperlukan.

Pada minggu ke-13, dilakukan pengembangan *endpoint* baru untuk melakukan *render* langsung dengan parameter `driveLink`, tanpa menggunakan `templateId`. *Endpoint* ini ditujukan untuk fleksibilitas penggunaan, seperti *preview template* dari luar sistem. Selama proses ini, *debugging* dan penyesuaian terhadap struktur *response* serta validasi parameter juga dilakukan agar tetap sesuai standar yang telah ditetapkan sebelumnya.

Pada minggu ke-14, proses *debugging* dan perbaikan *bug* pada *endpoint* `driveLink` yang baru ditambahkan dilanjutkan. Beberapa isu seperti eror saat *parsing* parameter dan struktur *response* yang tidak konsisten berhasil diatasi. Penyesuaian dilakukan agar *endpoint* tersebut lebih stabil dan dapat digunakan dalam berbagai skenario dengan hasil yang sesuai harapan.

Pada minggu ke-15, dilakukan *deployment* awal proyek `templating-service` ke *environment staging Heroku*. Proses ini melibatkan konfigurasi agar layanan bisa berjalan di lingkungan *cloud*, termasuk penyesuaian terhadap *environment variable* dan struktur proyek. *Deployment* ini menjadi tahap penting untuk menguji apakah sistem berjalan dengan baik di luar lingkungan lokal.

Pada minggu ke-16, terdapat beberapa error yang muncul saat proses *deployment*, khususnya yang berkaitan dengan koneksi ke *Redis dan Heroku*, konfigurasi *serializer*, dan pengaturan TTL. Permasalahan ini memerlukan penyesuaian konfigurasi *Redis* baik dari sisi kode maupun *file* `.properties` agar sistem bisa terhubung dan menyimpan *cache* dengan benar di lingkungan *staging*.

Pada minggu ke-17, mekanisme untuk menonaktifkan verifikasi *peer certificate* secara khusus hanya pada *environment testing* atau *development* ditambahkan. Hal ini dilakukan untuk mempermudah pengujian fitur *SSL* dan

memastikan bahwa fungsi `disablePeerVerification()` tidak diaktifkan di lingkungan produksi secara tidak sengaja, sehingga keamanan tetap terjaga.

Pada minggu ke-18, dilakukan serangkaian pengujian menyeluruh pasca-*deployment*. Diverifikasi bahwa semua *endpoint* dapat diakses dengan baik di Heroku dan *cache Redis* bekerja sebagaimana mestinya. Selain itu, dilakukan simulasi skenario *cache hit* dan *miss* untuk memastikan sistem dapat menangani proses *render* dan penyimpanan *cache* secara dinamis dan efisien.

Pada minggu ke-19, dilakukan evaluasi terhadap struktur proyek secara keseluruhan, termasuk kode dan konfigurasi yang telah diterapkan. Selain itu, pengujian integrasi *end-to-end* antara `templating-service` dengan *service* lain dalam sistem juga dilakukan untuk memastikan bahwa alur data dan proses *render* berjalan dengan lancar, stabil, dan sesuai kebutuhan sistem.

Pada minggu ke-20, pengujian integrasi secara menyeluruh antara *templating service* dengan layanan lain yang terhubung dalam sistem utama Minyma dilakukan. Pengujian ini dilakukan untuk memastikan bahwa setiap alur komunikasi antar-*service*, mulai dari permintaan data hingga proses *rendering* HTML, berjalan dengan lancar tanpa adanya gangguan. Selain itu, evaluasi terhadap performa sistem *caching* menggunakan *Redis* juga dilakukan untuk memastikan bahwa data yang telah disimpan dalam cache dapat diakses kembali dengan cepat dan sesuai dengan Time To Live (TTL) yang telah ditentukan.

Memasuki minggu ke-21 mulai berfokus pada penambahan dokumentasi teknis akhir, termasuk penjelasan struktur kode, deskripsi fungsionalitas, serta cara kerja masing-masing endpoint yang telah dibuat selama masa magang. Dokumentasi ini ditujukan untuk memudahkan proses *handover* kepada tim pengembang internal, agar mereka dapat melanjutkan pengembangan atau melakukan *maintenance* terhadap sistem yang telah dibangun. Selain dokumentasi, struktur kode juga dirapikan agar kode lebih jelas dan terstruktur.

Pada minggu ke-22, dilakukan review bersama supervisor untuk meninjau keseluruhan proses magang, mulai dari perencanaan, pelaksanaan, hingga hasil akhir.

3.3.1 Perancangan Sistem

Sistem *templating service* ini dirancang sebagai bagian dari pengembangan website utama Minyma dan *internal dashboard* menggunakan *framework Spring Boot* yang menerapkan arsitektur *MVC (Model-View-Controller)*. Ditulis dalam

bahasa pemrograman Java. Dalam proses perancangan sistem ini, dibutuhkan pemahaman menyeluruh terhadap alur bisnis, struktur data, dan kebutuhan fungsional yang telah ditentukan oleh perusahaan agar sistem yang dihasilkan dapat berjalan optimal serta memenuhi ekspektasi pengguna.

Templating service memiliki peran penting dalam mengelola dan merender halaman undangan digital yang bersifat dinamis berdasarkan input data tertentu. Oleh karena itu, perancangan sistem ini tidak hanya mencakup alur teknis seperti pengelolaan *template* dan *caching* dengan Redis, tetapi juga mencakup mekanisme *autentikasi*, pengendalian akses berbasis *role*, dan integrasi dengan layanan pihak ketiga seperti *Redis* dan layanan *deployment Heroku*.

A Requirements

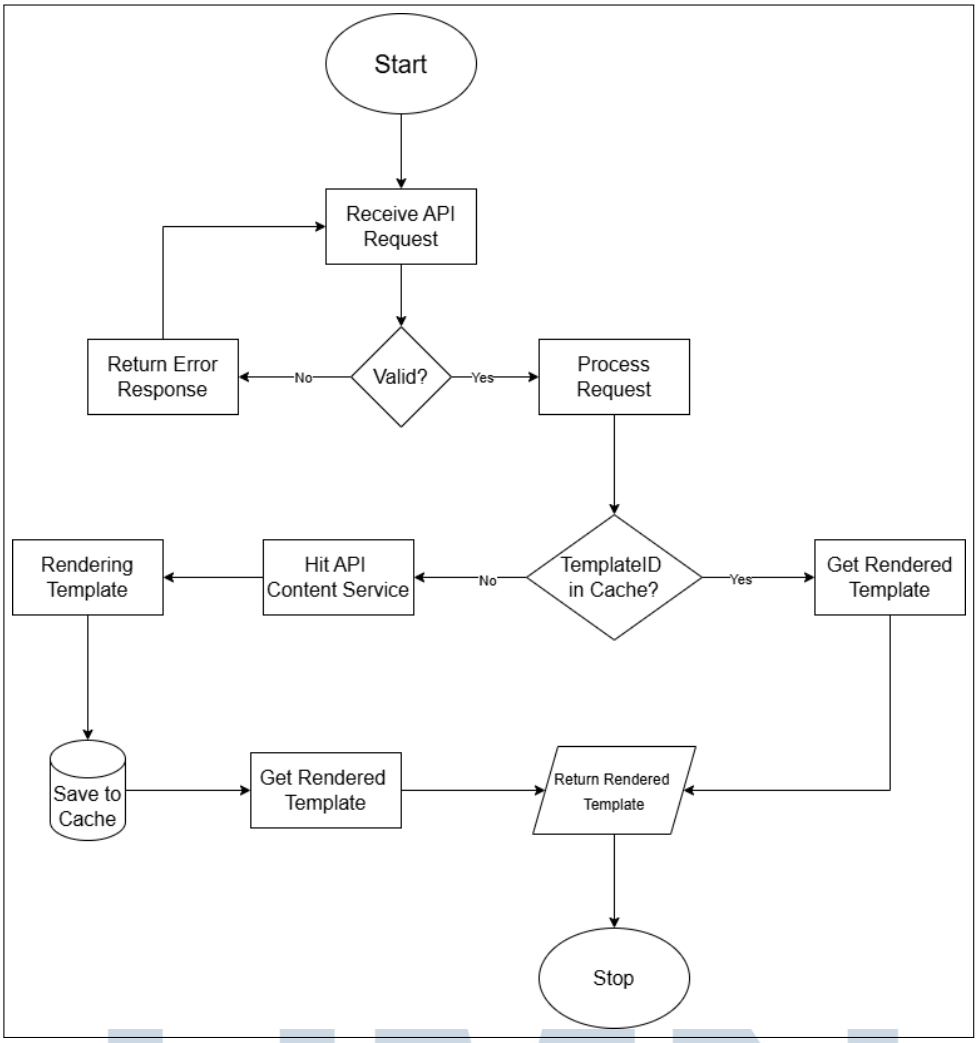
Dalam proses perancangan dan pengembangan *templating service* pada *internal dashboard* PT Jaya Santoso Teknologi, terdapat sejumlah kebutuhan sistem (*requirements*) yang harus dipenuhi untuk mendukung pengelolaan *template* secara efisien dan terkontrol berdasarkan hak akses pengguna. Adapun rincian kebutuhan tersebut adalah sebagai berikut:

- Pengguna yang belum *login* tetap dapat melihat daftar *template* yang tersedia secara publik.
- Pengguna yang belum *login* tidak dapat membuat, mengubah, atau menghapus *template*.
- Sistem membedakan hak akses pengguna berdasarkan *role: Super Admin, Internal, External (Guest/User), dan Admin (Finance Team).*
- *Super Admin* dan *Internal* dapat membuat *template* baru melalui halaman
- Super Admin*Create Template.*
- *Super Admin dan Internal dapat melihat, mengedit, dan menghapus template yang telah dibuat.*
- *Kartu template yang diklik akan me-redirect ke domain template sesuai dengan URL yang telah disimpan.*
- *Saat pengguna mengakses kartu template pada dashboard, sistem akan melakukan redirect ke domain template yang sesuai dengan URL yang telah ditentukan.*

- *Sistem wajib melakukan validasi role pengguna sebelum mengizinkan akses terhadap fitur manajemen template untuk memastikan hanya pengguna berwenang yang dapat memanipulasi data.*
- *Sistem harus mampu merender halaman HTML berdasarkan data dinamis yang diperoleh dari layanan lain, seperti content service, agar setiap undangan bersifat unik dan personal.*
- *Untuk meningkatkan performa dan mengurangi beban permintaan ke content service, sistem harus menerapkan strategi cache-first dalam proses rendering, yaitu menyimpan HTML yang sudah di-render agar bisa digunakan kembali saat ada permintaan berikutnya.*
- *Proses rendering harus tetap mempertahankan struktur desain yang konsisten agar identitas visual platform tetap terjaga, meskipun konten yang ditampilkan bersifat dinamis dan berbeda-beda untuk setiap pengguna.*

B Flowchart Templating Service

Untuk memberikan pemahaman yang lebih jelas mengenai mekanisme kerja sistem *templating service*, berikut disajikan diagram alur kerja yang menggambarkan proses secara menyeluruh. Diagram pada gambar 3.1 menunjukkan tahapan-tahapan utama mulai dari permintaan pengguna hingga proses perenderan halaman HTML yang telah dioptimalkan menggunakan strategi *cache-first*. Penyajian alur ini bertujuan untuk memperlihatkan bagaimana sistem berinteraksi dengan layanan lain dalam mengelola data dinamis serta bagaimana struktur dan proses teknis dirancang untuk menghasilkan undangan digital yang responsif, efisien, dan sesuai standar desain.



Gambar 3.1. Flowchart cara kerja Templating Service

Proses kerja *templating service* dimulai ketika sistem menerima permintaan melalui endpoint *API* dengan format `GET /template/{templateId}`. Tahapan alur teknisnya dimulai dari validasi akses pengguna. Sistem terlebih dahulu memverifikasi identitas pengguna yang melakukan permintaan. Jika pengguna tidak memiliki otorisasi yang valid (misalnya bukan bagian dari *role* internal atau *super admin*), maka sistem akan mengembalikan respons eror berupa kode HTTP 403 *(Forbidden)* dan proses dihentikan.

Jika pengguna valid, sistem akan mengekstrak `templateId` dari parameter *URL* dan melakukan pengecekan ke *Redis cache* untuk mengetahui apakah hasil *render* dari *template* tersebut sudah tersedia. Jika tersedia, sistem akan langsung mengambil data HTML dari *cache* dan mengirimkannya sebagai halaman yang sudah di-*render* ke pengguna. Namun, jika tidak tersedia, sistem akan melanjutkan ke tahap berikutnya.

Pada tahap selanjutnya, sistem akan memanggil fungsi khusus untuk melakukan proses *rendering* HTML berdasarkan `templateId` dan data yang diambil dari *content service*. Setelah proses *rendering* selesai, hasil HTML tersebut akan ditampilkan ke pengguna dan sekaligus disimpan ke dalam *Redis cache* dengan *key* yang sesuai dengan `templateId`, sehingga apabila ada permintaan yang sama di kemudian hari, proses dapat dilakukan lebih cepat tanpa perlu *render* ulang.

3.3.2 Implementasi

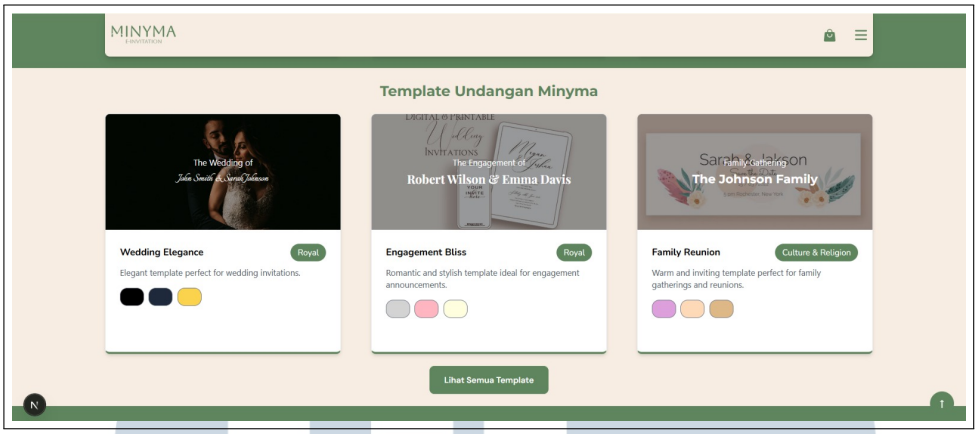
Setelah perancangan sistem dan alur kerja dijelaskan melalui flowchart, tahap selanjutnya adalah implementasi dari *templating service* ke dalam lingkungan pengembangan yang telah disediakan oleh PT Jaya Santoso Teknologi. Pada tahap ini, komponen-komponen sistem mulai direalisasikan sesuai dengan kebutuhan fungsional dan non-fungsional yang telah ditetapkan sebelumnya.

Gambaran implementasi berikut menunjukkan bagaimana struktur sistem diintegrasikan dengan *frontend* dan layanan-layanan yang relevan, serta bagaimana proses *rendering* HTML berjalan menggunakan data dinamis yang diperoleh dari *content service*. Selain itu, strategi *cache-first* juga diterapkan untuk mendukung efisiensi waktu respon dan stabilitas sistem dalam menangani permintaan dari berbagai pengguna.

A Halaman Template pada Website Utama Minyma

Sistem *templating service* terintegrasi dengan website utama Minyma untuk memberikan pengalaman pengguna yang lebih interaktif dan informatif. Salah satu implementasi nyatanya dapat dilihat pada halaman *Template* di website utama, yang dirancang untuk menampilkan daftar desain undangan digital yang tersedia secara publik.

Halaman ini memungkinkan calon pengguna maupun pengunjung umum untuk melihat berbagai pilihan desain undangan yang telah dibuat menggunakan sistem *templating service*. Tujuannya adalah memberikan kebebasan dalam memilih desain yang sesuai dengan preferensi masing-masing pengguna, sekaligus memperkenalkan kualitas visual dan variasi produk yang ditawarkan oleh Minyma.



Gambar 3.2. Tampilan Halaman Template pada Website Utama Minyma

Gambar 3.2 menunjukkan tampilan *website* utama Minyma yang bisa diakses oleh semua pengguna tanpa perlu *login* terlebih dahulu. Pada *website* utama Minyma, pengguna dapat mengakses berbagai pilihan *template* undangan digital yang telah disediakan oleh admin. Halaman ini dirancang sebagai katalog visual yang memudahkan pengguna dalam menelusuri dan memilih desain undangan sesuai dengan selera dan kebutuhan mereka.

Setiap *template* ditampilkan dalam bentuk pratinjau sehingga pengguna bisa langsung melihat tampilan awal dari masing-masing desain. Tidak hanya itu, pengguna juga diberikan kebebasan untuk menyesuaikan elemen warna dari *template* melalui palet warna yang interaktif. Dengan fitur ini, pengguna dapat bereksperimen dengan kombinasi warna yang berbeda-beda hingga menemukan komposisi warna yang paling sesuai dengan preferensi pribadi atau tema acara mereka.

Gambar 3.3. *See Details Template*

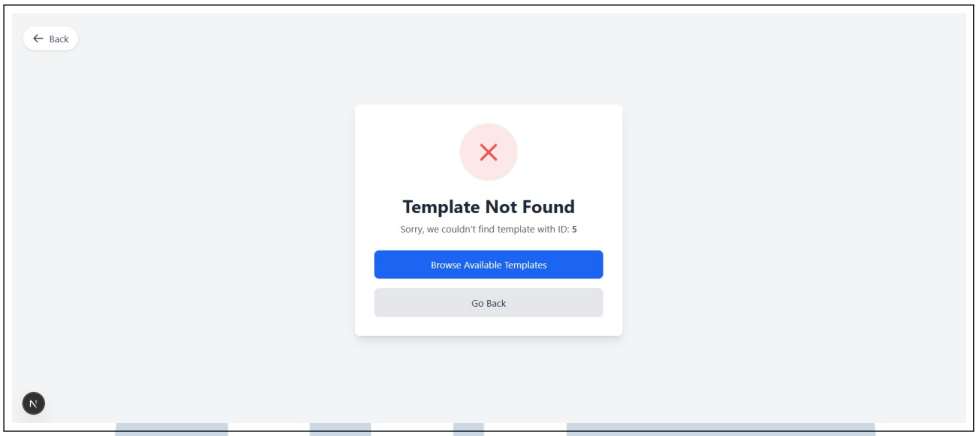
Pada gambar 3.3 menunjukkan detail *preview* undangan digital. Setiap *template* undangan digital yang tersedia memiliki tombol *Open Invitation* yang

memungkinkan pengguna untuk melihat versi akhir dari undangan berdasarkan *template* yang mereka pilih. Ketika tombol *Open Invitation* ditekan, sistem akan secara otomatis menjalankan proses *HTML rendering* dengan memanfaatkan layanan *templating service* yang telah dibangun di sisi *backend*.

Secara teknis, ketika pengguna menekan tombol tersebut, permintaan HTTP akan dikirimkan ke *endpoint API* `/template` yang disediakan oleh *templating service*. Permintaan ini mencakup parameter berupa `templateID` untuk memastikan bahwa sistem dapat me-*render* undangan yang sesuai dengan *template* yang digunakan oleh pengguna tersebut. Sebelum permintaan diterima, sistem melakukan validasi terhadap data permintaan, termasuk memeriksa apakah `templateID` yang diminta tersedia dan terdaftar dalam sistem. Setelah validasi berhasil, layanan *templating service* akan memeriksa apakah hasil *render* dari `templateID` tersebut sudah tersedia di dalam *Redis cache*. Jika tersedia, maka HTML hasil *rendering* akan langsung diambil dari *cache* dan dikirimkan sebagai respons ke pengguna. Proses ini mempercepat waktu respons karena tidak perlu melakukan pengambilan data ulang dari *content service*. Jika belum tersedia, *templating service* akan mengirimkan permintaan HTTP *GET* ke layanan *content service* untuk mengambil data konten yang berkaitan. Setelah data konten diperoleh, sistem akan me-*render* halaman HTML sesuai struktur dan desain yang telah ditentukan. Hasil HTML yang sudah di-*render* kemudian dikembalikan sebagai respons ke pengguna, dan sekaligus disimpan ke dalam *Redis* untuk permintaan berikutnya.

Hadirnya halaman *template* tidak hanya berfungsi sebagai galeri desain, namun juga menjadi pintu awal dalam proses personalisasi undangan digital yang cepat dan intuitif bagi pengguna Minyma, serta memberikan efisiensi melalui penggunaan *cache*, serta modularitas arsitektur sistem yang memisahkan logika rendering dari sistem utama *website*.

Apabila terjadi kendala seperti kesalahan dalam pengambilan data dari *Content Service*, masalah koneksi, atau *template ID* yang tidak valid yang menyebabkan proses *rendering* tidak berhasil dilakukan, sistem akan secara otomatis menampilkan pesan error yang informatif agar pengguna tidak bingung.



Gambar 3.4. Pesan Error

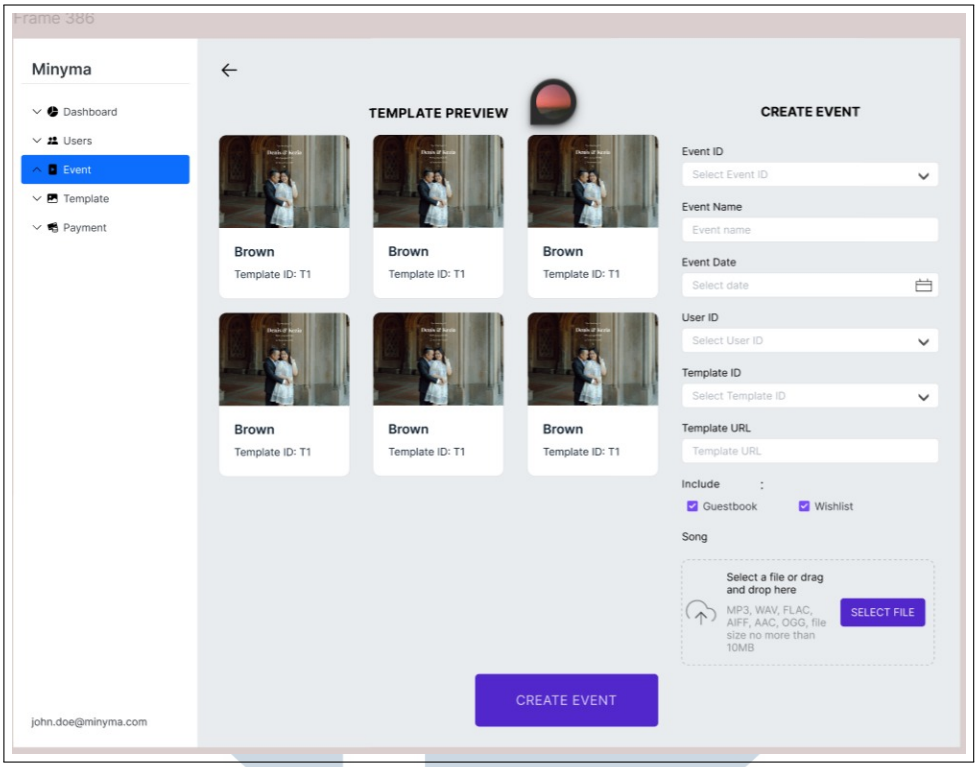
Gambar 3.4 menunjukkan *template* pesan error yang berfungsi sebagai cadangan tampilan untuk menjaga pengalaman pengguna agar tidak terputus secara tiba-tiba. Sistem akan menampilkan pesan error yang menjelaskan bahwa halaman undangan tidak dapat dirender atau tidak ditemukan untuk sementara waktu, disertai saran agar pengguna mencoba beberapa saat lagi.

B Halaman Internal Dashboard

Sistem *templating service* juga diimplementasikan secara langsung dalam halaman *Internal Dashboard*, yang digunakan oleh tim internal Minyma untuk mengelola seluruh data terkait *template*. Dashboard ini dirancang untuk memberikan antarmuka yang intuitif dan fungsional, sehingga mempermudah proses pengelolaan *template* oleh pengguna dengan peran tertentu seperti *Super Admin* dan *Internal*.

B.1 Halaman Event

Pada halaman *Event* di *website* Minyma, admin dapat melihat daftar acara yang sedang aktif berdasarkan tanggal pelaksanaan. Halaman ini berfungsi untuk menampilkan *preview* dari template undangan digital yang digunakan oleh masing-masing acara. Sebuah acara akan tetap ditampilkan dan dapat diakses melalui halaman *Event* hingga hari pelaksanaan tiba.



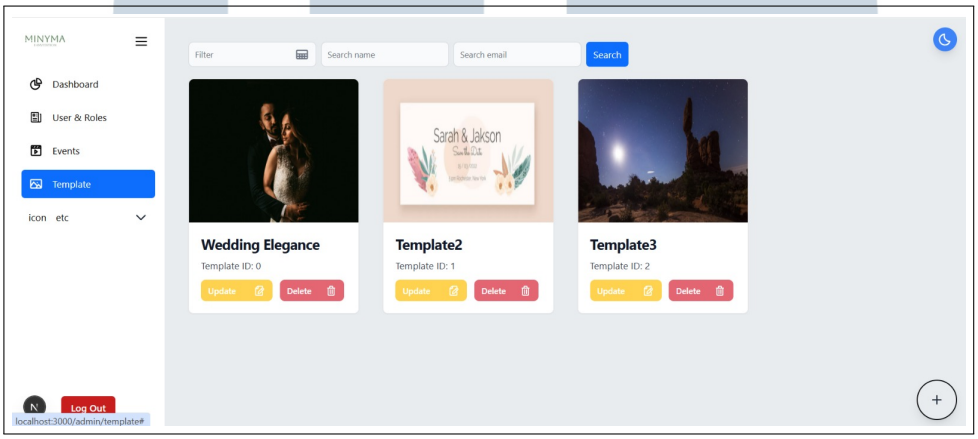
Gambar 3.5. *Template Preview*

Untuk menampilkan *preview template* pada halaman *Event*, admin harus menambahkan *Event* terlebih dahulu di halaman *Create Event*. Selanjutnya, admin akan mengisi beberapa informasi penting terkait acara tersebut. Informasi yang wajib diisi meliputi `eventID`, `event name`, `event date`, `userID`, `templateID`, dan `templateURL`. `eventID` sendiri akan secara otomatis dibuat oleh sistem jika pengguna eksternal telah melakukan pembayaran lunas. Informasi ini tidak dapat diinput secara manual oleh admin. `event name` dan `event date` diinput langsung oleh admin sesuai dengan permintaan pengguna eksternal. `userID` merupakan identitas dari pengguna eksternal, sementara `templateID` merujuk pada jenis *template* undangan yang dipilih. Untuk `templateURL`, admin dapat membuat *URL* berdasarkan struktur yang telah ditentukan.

Selain informasi tersebut, terdapat juga beberapa informasi opsional yang dapat diaktifkan sesuai kebutuhan pengguna, yaitu fitur *guestbook* untuk menampilkan kolom ucapan tamu, *wishlist* untuk menampilkan daftar hadiah, dan *lagu* sebagai latar suara undangan. Setelah *event* berhasil dibuat dan disimpan, maka informasi acara beserta *template* undangan digitalnya akan muncul di halaman *Event*.

B.2 Halaman Template

Salah satu komponen utama dalam *Internal Dashboard* adalah halaman *Template*, yang berfungsi sebagai pusat pengelolaan seluruh data *template* undangan digital. Melalui halaman ini, pengguna internal seperti *Super Admin* dan *Internal* dapat melihat daftar seluruh *template* yang telah dibuat, serta melakukan berbagai aksi seperti menambah, mengedit, atau menghapus *template* sesuai kebutuhan. Berikut ini disajikan tampilan dari halaman *Template Internal Dashboard* Minyma.



Gambar 3.6. Halaman *Template* di *internal dashboard*

Gambar 3.6 menunjukkan tampilan pada halaman *Template* dalam sistem Minyma. Halaman ini menampilkan daftar *template* undangan digital yang dapat dilihat pada halaman utama *website* Minyma. Halaman ini bertujuan untuk mengatur *template* apa saja yang ingin kita tawarkan kepada pengguna eksternal. Daftar *template* ini ditampilkan dalam bentuk kartu (*card*) yang dirancang untuk memberikan tampilan visual dari masing-masing *template* secara jelas dan menarik. Tampilan kartu ini telah dirancang terlebih dahulu melalui prototipe pada *Figma* dan digunakan sebagai acuan dalam pengembangan antarmuka.

Ketika salah satu kartu *template* diklik oleh pengguna, sistem akan menampilkan keseluruhan tampilan undangan digital secara utuh berdasarkan desain dan konfigurasi yang telah ditentukan. Proses ini dilakukan dengan memanfaatkan layanan *templating service*. Sama seperti sebelumnya, *templating service* akan menerima permintaan berdasarkan `templateID` yang dipilih, kemudian mengambil data dari *content service*, melakukan proses *HTML rendering*, dan menyajikannya dalam bentuk halaman undangan digital yang dapat diakses secara langsung oleh pengguna. Dengan mekanisme ini, pengguna dapat melihat

undangan versi akhir, termasuk elemen visual dan informasi yang akan muncul jika mereka memilih menggunakan *template* tersebut.

Setiap kartu *template* hanya dapat dilihat, diedit, atau dihapus oleh pengguna dengan peran (*role*) Internal dan *Super* admin. Hal ini bertujuan untuk menjaga agar hanya pengguna dengan hak akses tertentu yang dapat melakukan perubahan pada daftar *template* yang tersedia di sistem. Ketika salah satu kartu *template* ditekan, maka pengguna akan diarahkan ke halaman undangan digital sesuai dengan *templateURL* yang telah ditentukan, yaitu *domain* atau alamat *URL* dari *template* tersebut.

Untuk menambahkan *template* baru, pengguna yang memiliki akses dapat menekan tombol tambah (+), yang akan mengarahkan ke halaman *Create Template*. Pada halaman ini, pengguna diminta untuk mengisi beberapa informasi penting terkait *template* yang akan ditambahkan, yaitu nama template (`template name`), alamat URL template (`templateURL`), serta gambar representatif dari *template* tersebut yang akan ditampilkan pada kartu di halaman utama. Setelah semua informasi diisi dan disimpan, maka sistem akan secara otomatis menghasilkan `templateID` sebagai identitas unik dari *template* tersebut. *Template* yang telah berhasil ditambahkan akan muncul dalam daftar dan ditampilkan dalam bentuk kartu pada halaman utama, lengkap dengan fungsionalitas akses ke *template* secara langsung melalui URL yang telah ditentukan.

3.4 Spesifikasi Sistem

Dalam proses perancangan dan pengembangan sistem *templating service* serta website Minyma, diperlukan dukungan dari berbagai perangkat lunak dan teknologi modern guna memastikan efisiensi dalam implementasi serta performa sistem yang optimal. Pemilihan teknologi didasarkan pada kebutuhan pengembangan berbasis *microservice*, pengelolaan data dinamis, dan kecepatan *rendering* halaman. Adapun spesifikasi sistem yang digunakan secara keseluruhan adalah sebagai berikut:

- *Spring Boot*
  Merupakan kerangka kerja berbasis Java yang digunakan sebagai fondasi utama dalam membangun sistem *backend*. Spring Boot dipilih karena menyediakan konfigurasi otomatis, arsitektur modular, serta dukungan penuh terhadap pembuatan RESTful API. Dalam proyek ini, Spring Boot menangani logika bisnis, manajemen data, autentikasi pengguna, serta integrasi dengan *templating engine* dan sistem cache.

- *IntelliJ IDEA*

  Digunakan sebagai *Integrated Development Environment (IDE)* dalam menulis dan mengembangkan kode Java. IntelliJ menawarkan fitur yang lengkap seperti *code suggestions*, refactoring tools, integrasi dengan Git, dan fitur debugging yang membantu proses pengembangan menjadi lebih efisien dan minim kesalahan.

- *Heroku*

  Heroku adalah platform berbasis cloud (*Platform as a Service*) yang digunakan untuk melakukan *deployment* aplikasi secara *online*. Dengan menggunakan Heroku, aplikasi *templating service* dapat diakses secara publik melalui URL yang ditentukan, sehingga dapat digunakan oleh sistem undangan digital dan berinteraksi langsung dengan pengguna.

- *Redis*

  Redis berfungsi sebagai sistem *in-memory data store* yang diimplementasikan untuk kebutuhan *caching*. HTML yang telah dirender disimpan sementara di Redis agar dapat diakses ulang dengan cepat, tanpa perlu melakukan *re-rendering* dari awal. Hal ini secara signifikan meningkatkan waktu respon dan mengurangi beban pemrosesan dari *Content Service* yang bertanggung jawab terhadap penyediaan data dinamis.

- *PostgreSQL*

  Digunakan sebagai basis data utama untuk menyimpan data pengguna, data *template*, detail acara, serta konfigurasi tambahan lainnya. PostgreSQL dipilih karena stabilitasnya, dukungan relasional yang kuat, serta kemampuannya untuk diintegrasikan dengan framework Spring Boot secara lancar. Penggunaan `UUID` dan relasi antar tabel juga mendukung struktur data yang kompleks namun teratur.

- *GitHub*

  Seluruh proses pengembangan dikelola menggunakan GitHub sebagai sistem kontrol versi. Dengan GitHub, kolaborasi antar anggota tim dapat dijalankan secara sinkron melalui *branching*, *pull request*, dan *issue tracking*, sekaligus mencatat setiap histori perubahan kode agar proses pengembangan tetap terdokumentasi dengan baik.

- *Discord dan Notion*

  Digunakan sebagai media komunikasi dan koordinasi kerja antar anggota

tim. Discord dimanfaatkan untuk diskusi teknis, pelaporan progres harian, dan penyelesaian kendala secara langsung, sedangkan Notion difungsikan sebagai papan manajemen proyek, dokumentasi teknis, pembagian tugas, dan pencatatan hasil rapat.

Dengan kombinasi teknologi-teknologi tersebut, sistem yang dibangun mampu bekerja secara efisien, stabil, dan mudah dikembangkan lebih lanjut sesuai kebutuhan pengguna dan perusahaan.

3.5 Kendala dan Solusi yang Ditemukan

3.5.1 Kendala

1. Komunikasi sulit karena sistem kerja *Work From Home* (WFH).
   Koordinasi menjadi tidak efisien, terutama saat menemukan *error* atau membutuhkan klarifikasi fitur.

2. Adanya perubahan alur (*flow*) aplikasi di tengah pengembangan.
   Hal ini membuat implementasi menjadi membingungkan karena dokumentasi dan arahan awal sudah tidak relevan.

3. *Workflow* pengembangan lambat karena saling menunggu antara *backend* dan *frontend*.
   Proses pengerjaan tidak bisa berjalan paralel karena adanya ketergantungan antara kedua sisi.

4. Kesulitan dalam konfigurasi Redis *Cache* di lingkungan produksi Heroku.
   Error serialization dan koneksi Redis terjadi saat proses *deployment* ke Heroku, terutama karena perbedaan *environment* lokal dan *cloud*.

3.5.2 Solusi

1. Menjadwalkan *meeting* rutin melalui *Discord*.
   Untuk menjaga komunikasi tetap lancar dan mengurangi hambatan kolaborasi saat *WFH*.

2. Konfirmasi ulang kebutuhan dan perubahan fitur kepada *Supervisor*.
   Agar implementasi tetap relevan dengan arah pengembangan terbaru.

3. Membagi tugas secara modular antara *frontend* dan *backend*.
   Sehingga pengembangan bisa dilakukan secara paralel dan mengurangi waktu tunggu antar tim.

4. Mengganti konfigurasi Redis dan menyesuaikan *environment* di Heroku.
   Mengatasi *error* dengan memahami perbedaan konfigurasi lokal dan *cloud*, serta melakukan penyesuaian seperti mengganti *serializer* dan menambahkan pengaturan koneksi *Redis Cloud* secara manual.