PENGEMBANGAN *FRONTEND* DALAM MIGRASI *MULTI PAGE APPLICATION* KE *SINGLE PAGE APPLICATION* DENGAN PENDEKATAN *ITERATIVE INCREMENTAL* PADA STUDI KASUS SOFI MODUL PENGAJUAN

1st Andrian Saputra
Industrial Engineering Faculty
Telkom University
Bandung, Indonesia
andrians@telkomuniversity.ac.id

Ekky Novriza Alam
Industrial Engineering Faculty
Telkom University
Bandung, Indonesia
ekkynovrizalam@telkomuniversity.ac.id

Tien Fabrianti Kusumasari,
Industrial Engineering Faculty
Telkom University
Bandung, Indonesia
tienkusumasari@telkomuniversity.ac.id

Abstrak* — Revolusi industri 4.0 mendorong sektor pendidikan untuk mengadopsi teknologi terkini seperti pembelajaran jarak jauh dan platform digital. Universitas Telkom mengembangkan platform SOFI untuk memonitor Sidang Tugas Akhir di Fakultas Rekayasa Industri. Awalnya menggunakan arsitektur monolitik dan Multi Page Application (MPA), aplikasi ini kini memerlukan peningkatan untuk menangani jumlah pengguna yang besar dan memperbaiki pengalaman pengguna. Penelitian ini bertujuan mengembangkan front-end aplikasi SOFI dengan menerapkan Single Page Application (SPA) dan arsitektur microservices. SPA dipilih untuk mengurangi waktu respon dan meningkatkan interaktivitas serta penanganan error, sedangkan microservices menawarkan fleksibilitas dan skalabilitas yang lebih tinggi. Metodologi yang digunakan adalah System Development Life Cycle (SDLC) dengan pendekatan Iterative Incremental, karena mampu mengakomodasi umpan balik pengguna dan perubahan kebutuhan selama proses pengembangan. Hasil penelitian menunjukkan bahwa migrasi ke SPA dan microservices meningkatkan performa dan pengalaman pengguna aplikasi SOFI serta mempermudah pengembangan dan pemeliharaan kode. Penelitian ini memberikan kontribusi signifikan terhadap digitalisasi layanan pendidikan, khususnya dalam efisiensi dan kenyamanan proses sidang tugas akhir.

***Kata kunci* — Digitalisasi Pendidikan, Microservices, Iterative Incrementa, Multi Page Application, Single Page Application , Systems Development Life Cycle**

I. PENDAHULUAN

Sektor pendidikan telah mengalami transformasi besar dengan mengadopsi konsep industri 4.0, yang semakin dipercepat dengan adanya COVID-19. Kemajuan seperti pembelajaran jarak jauh, pembelajaran adaptif, dan penggunaan platform digital telah mengubah cara mahasiswa dan pendidik berinteraksi dan mengakses informasi, meningkatkan fleksibilitas dan aksesbilitas. Revolusi industri 4.0 ini menuntut institusi pendidikan untuk terus berkembang dengan mengikut perkembangan teknologi dan memanfaatkan teknologi informasi dan komunikasi untuk mendukung proses belajar mengajar. Salah satu layanan yang memiliki potensi besar untuk didigitalisasi adalah sidang tugas akhir, yang merupakan ujian terbuka bagi mahasiswa untuk penetapan status kemajuan studi mahasiswa. Digitalisasi sidang tugas akhir tidak hanya memungkinkan proses yang lebih efisien dan terorganisir tetapi juga memberikan fleksibilitas dan akses yang lebih besar bagi mahasiswa, pembimbing, dan penguji, sejalan dengan tren modern dalam pendidikan tinggi.

Universitas Telkom merupakan salah satu perguruan tinggi yang telah menyesuaikan diri dengan industri 4.0, dan salah satu upaya Universitas Telkom dalam menjawab tantangan ini adalah digitalisasi pada layanan sidang tugas akhir. Bukti nyata dari digitalisasi ini adalah *website* SOFI, sebuah platform berbasis web yang digunakan untuk memonitor Sidang Tugas Akhir Fakultas Rekayasa Industri di Universitas Telkom. Beberapa proses bisnis pada aplikasi SOFI adalah pendaftaran sidang, penjadwalan sidang, pelaksanaan sidang, revisi sidang, dan penilaian sidang.

Aplikasi SOFI dikembangkan menggunakan arsitektur monolitik. Penelitian terdahulu yang berjudulkan "A Comparative Review of Microservices and Monolithic Architectures" [1] membuktikan bahwa arsitektur monolitik cocok digunakan jika pengguna masih pada skala yang kecil dan juga mudah untuk dikembangkan. Namun, arsitektur monolitik tidak dapat memenuhi kebutuhan website SOFI yang memerlukan arsitektur kode yang memudahkan pengembang dalam memahami kode serta mampu menangani pengguna dalam jumlah besar. Karena website SOFI tidak memiliki pengembang tetap dan memiliki pengguna dengan skala yang besar, diperlukan perubahan ke arsitektur yang lebih fleksibel dan skalabel. Hal ini menyebabkan aplikasi SOFI sering kali mengalami keluhan dari pengguna.

Aplikasi SOFI membutuhkan teknologi yang dapat menangangi jumlah pengguna yang tinggi secara bersamaan, arsitektur monolitik tidak dapat memenuhi kebututuhan tersebut. Oleh karena itu, arsitektur yang diharapkan dapat memenuhi kebutuhan *website* SOFI adalah arsitektur *microservice*, arsitektur *microservice* adalah arsitektur aplikasi dengan memecah basis kode menjadi unit pecahan yang lebih kecil dan spesifik. Penelitian terdahulu yang berjudulkan "The Comparison of Microservice and Monolithic Architecture" [2] membuktikan bahwa arsitektur *microservice* lebih efisien jika aplikasi harus menangani pengguna dalam jumlah yang besar. Penelitian tersebut juga membuktikan bahwa arsitektur *microservice* lebih mudah untuk di pelihara oleh pengembang, karena lebih mudah

untuk dipahami oleh pengembang, setiap fungsi utama terpisah dan kesalahan fungsi *microservice* hanya mempengaruhi *microservice* itu sendiri.

Saat ini, *website* SOFI juga masih menggunakan konsep MPA *(Multi Page Application)*. Dalam konsep MPA, setiap kali pengguna melakukan navigasi antar halaman, seluruh konten *website* harus dimuat ulang. Hal ini menyebabkan *website* menjadi lebih berat dan memperpanjang waktu muat. Namun, MPA memiliki keunggulan dalam hal SEO, yang memudahkan *website* untuk ditemukan diinternet. Meskipun demikian, kelebihan MPA dalam hal SEO seringkali bertentangan dengan kebutuhan untuk meningkatkan pengalaman pengguna. Terlebih bagi, untuk *website* yang hanya digunakan secara internal oleh organisasi, keunggulan SEO tersebut menjadi kurang relevan. Kelemahan lain dari MPA dari sisi pengguna termasuk peningkatan penggunaan bandwidth karena setiap navigasi memuat ulang halaman sepenuhnya, yang bisa menjadi masalah bagi pengguna dengan koneksi internet lambat. Selain itu, transisi antar halaman yang tidak mulus dapat mengganggu kenyamanan pengguna dan membuat interaksi terasa lambat dan terputus-putus. Kelemahan-kelemahan ini juga berkorelasi dengan masalah penanganan error dari sisi pengguna dalam MPA. Setiap kali halaman dimuat ulang, ada potensi kehilangan data yang telah diinput oleh pengguna sebelum error terjadi. Misalnya, jika terjadi error selama proses pengisian formulir atau pendaftaran, pengguna mungkin harus mengulangi seluruh proses dari awal, yang menyebabkan frustrasi dan ketidakpuasan. Error handling yang tidak efisien ini sering kali mengakibatkan pengguna kehilangan data yang telah dimasukkan, memperburuk pengalaman pengguna secara keseluruhan. Dengan demikian, masalah bandwidth yang tinggi, transisi halaman yang tidak mulus, dan penanganan error yang buruk berkontribusi terhadap pengalaman pengguna yang kurang optimal dalam MPA.

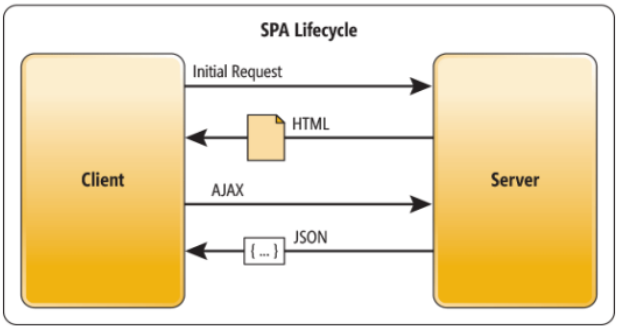
Pengguna SOFi memiliki rentang usia yang sangat luas, mencakup mahasiswa muda hingga dosen senior. Karena perbedaan dalam tingkat keterampilan teknologi dan preferensi pengguna, penting untuk memperhatikan kebutuhan dan preferensi dari kedua kelompok ini. Intuitifitas, responsivitas, dan penanganan error dalam aplikasi SOFi menjadi semakin penting untuk memastikan bahwa semua pengguna dapat dengan mudah mengakses dan menggunakan platform tersebut sesuai dengan kebutuhan mereka.

Oleh karena itu, terdapat konsep SPA *(Single Page Application)*. Aplikasi SPA akan meningkatkan interaktivitas pengguna karena mereka akan merespon dalam waktu yang lebih singkat. Ini akan memakan waktu lebih lama saat mengunggah untuk pertama kalinya tetapi untuk tindakan berikutnya, waktu yang dibutuhkan kurang dari aplikasi MPA [3]. Dalam SPA, hanya bagian-bagian tertentu dari website yang diperbarui saat pengguna berinteraksi, sehingga tidak perlu memuat ulang seluruh halaman. Keuntungan SPA bagi pengguna adalah peningkatan performa dan penanganan error yang lebih efisien. Ketika terjadi *error*, data yang telah diinput tidak akan hilang karena halaman tidak perlu dimuat ulang sepenuhnya, yang mengurangi frustrasi dan ketidakpuasan pengguna. Namun, kelemahan utama dari SPA adalah kurangnya keunggulan dalam hal SEO, karena konten yang terus-menerus diperbarui secara dinamis mungkin tidak selalu terindeks dengan baik oleh mesin pencari. Selain itu, aplikasi SOFI juga akan dimigrasi dari arsitektur monolitik ke *microservices*, yang memberikan fleksibilitas dan skalabilitas yang tinggi. Menggabungkan SPA dengan arsitektur *microservices* memberikan beberapa keuntungan. SPA dapat memanfaatkan *microservices* untuk memisahkan berbagai komponen aplikasi menjadi layanan-layanan kecil yang dapat dikembangkan, diimplementasikan, dan diskalakan secara independen. Hal ini memungkinkan pengembangan yang lebih cepat dan responsif terhadap perubahan kebutuhan pengguna, serta penanganan *error* yang lebih baik karena setiap layanan dapat dikelola dan diperbaiki secara terpisah tanpa mempengaruhi keseluruhan aplikasi. Dengan demikian, kombinasi SPA dan arsitektur microservices memastikan performa yang optimal dan pengalaman pengguna yang lebih baik pada aplikasi SOFI.

Untuk mendukung penelitian ini, pendekatan yang digunakan adalah SDLC *(System Development Life Cycle)* dengan metode *Iterative Incremental*. SDLC adalah model klasik yang bersifat sistematis dan berurutan dalam membangun perangkat lunak. Dalam pengembangan sistem, SDLC sangat penting karena memecah seluruh siklus pengembangan perangkat lunak menjadi tahapan-tahapan yang memungkinkan evaluasi setiap bagian dengan lebih mudah, serta memungkinkan programmer bekerja secara bersamaan pada setiap tahapannya [4]. Untuk mendukung penelitian ini, pendekatan yang digunakan adalah SDLC *(System Development Life Cycle)* dengan metode *Iterative Incremental*. Metode *Iterative Incremental* dipilih karena menggabungkan elemen-elemen dari metode *waterfall* dalam pendekatan yang iteratif dan fleksibel. Metode ini sangat sesuai untuk pengembangan perangkat lunak karena memungkinkan adanya penyesuaian berdasarkan umpan balik pengguna dan perubahan kebutuhan selama proses pengembangan. Fleksibilitas ini sangat penting untuk memastikan setiap iterasi dapat mengakomodasi inovasi dan ide-ide baru yang muncul, sehingga menghasilkan solusi yang lebih responsif dan sesuai dengan tujuan akhir. Penelitian ini menggunakan metode *Iterative Incremental* dari SDLC karena keunggulannya dalam memungkinkan penyesuaian berkelanjutan dan peningkatan terus-menerus selama siklus pengembangan. Metode ini sangat diperlukan untuk menangani kompleksitas migrasi arsitektur dan memastikan performa serta pengalaman pengguna yang optimal pada website SOFI.

II. KAJIAN TEORI

A. *Single Page Application*

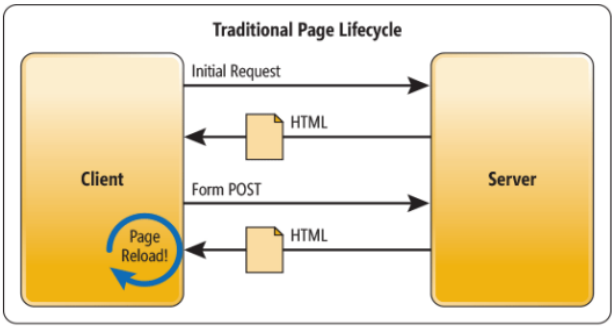


GAMBAR 1
LIFE CYCLE SINGLE PAGE APPLICATION

Single Page Application (SPA) adalah konsep web yang memuat satu halaman HTML dan memperbarui konten secara dinamis sesuai interaksi pengguna tanpa memuat ulang seluruh halaman dari server [5]. SPA menggunakan teknologi Javascript asynchronous untuk pengembangan frontend dan backend, memungkinkan interaksi pengguna tanpa berpindah antar halaman. SPA mempermudah pengembangan kode karena komponen aplikasi berdiri sendiri dan tidak saling bergantung, sehingga lebih modular dan skalabel [6].

SPA meminimalisir ketergantungan pada manipulasi *Document Object Model* (DOM) di sisi klien, meningkatkan performa aplikasi. Secara teknis, SPA berkomunikasi dengan server melalui Restful JSON API, memperbarui tampilan halaman tanpa memuat ulang seluruh halaman, meningkatkan efisiensi dan memberikan pengalaman pengguna yang lebih responsif [5]. Dibandingkan dengan aplikasi MPA, SPA menawarkan interaksi yang lebih cepat dan responsif, karena konten baru dimuat secara dinamis menggunakan Javascript, sementara MPA memuat ulang halaman HTML baru untuk setiap perubahan konten [6].

B. Multi Page Application



GAMBAR 2
LIFE CYCLE MULTI PAGE APPLICATION

Multi Page Application (MPA) adalah konsep aplikasi yang memuat ulang seluruh halaman HTML dari server pada setiap permintaan. Sebelum AJAX diperkenalkan, MPA sangat populer dengan berbagai bahasa pemrograman server seperti PHP, Ruby, dan Java yang bertanggung jawab memuat kerangka langsung di backend dan memberikan halaman HTML mentah sebagai respon. Dengan AJAX, pengembang mulai menerapkan pemuatan asinkron untuk transisi mulus dan interaksi UI yang lebih baik, namun tidak semua halaman dimuat asinkron. Hal ini menyebabkan campuran pendekatan pemuatan sinkron dan asinkron serta peningkatan kode JavaScript yang signifikan di sisi klien, yang sering kali menjadi sulit dikelola [7].

Usaha oleh *library* seperti jQuery dan *framework* sisi server untuk membuat kode JavaScript lebih mudah dan rapi masih belum cukup baik dibandingkan dengan apa yang ditawarkan SPA. Meskipun MPA memiliki keunggulan dalam kesederhanaan dan kemudahan penerapan awal, MPA sering kali kalah dalam hal performa dan pengalaman pengguna dibandingkan dengan SPA [7].

C. Software Development Life Cycle

Software Development Life Cycle (SDLC) adalah metodologi dengan proses yang didefinisikan untuk menciptakan perangkat lunak berkualitas tinggi, efektif dari segi biaya, dan dapat diandalkan. SDLC mencakup berbagai aktivitas dan tugas selama pengembangan perangkat lunak, membuat proses lebih sistematis dan terstruktur [8].

Model SDLC membantu membuat pengembangan perangkat lunak lebih efisien melalui perencanaan yang tepat, mencakup identifikasi kebutuhan, pengembangan kasus bisnis, dan implementasi solusi sistem. Analis sistem, pengembang, dan desainer menggunakan SDLC untuk merencanakan dan mengimplementasikan aplikasi, memastikan pengiriman sistem tepat waktu dan dengan anggaran murah [9].

Beberapa model SDLC yang umum digunakan termasuk Waterfall, Spiral, V, Agile, Iteratif, dan Rapid Application Development (RAD), masing-masing dengan kelebihan dan keterbatasannya sendiri [8]. SDLC mencakup tahapan utama seperti analisis kebutuhan, desain sistem, implementasi, pengujian, dan pemeliharaan, memastikan setiap iterasi dapat mengakomodasi inovasi baru untuk menghasilkan solusi yang lebih responsif dan sesuai dengan tujuan akhir [9].

D. Iterative and Incremental Development

Iterative and Incremental Development (IID) adalah metode pengembangan perangkat lunak yang menggabungkan pendekatan *iterative* dan *incremental*. IID, sebagai bagian dari SDLC, digunakan untuk merencanakan, mengembangkan, menguji, dan memelihara perangkat lunak, termasuk dalam model seperti Waterfall, Spiral, V-Model, dan Agile.

Pendekatan *iterative* membangun bagian kecil proyek secara bertahap, mengungkap masalah lebih awal dan memungkinkan timbal balik dari stakeholder di setiap iterasi, sehingga setiap peningkatan didasarkan pada kebutuhan nyata [10]. Pendekatan *incremental* menambahkan fungsionalitas baru pada setiap rilis, mendukung desain, implementasi, dan pengujian sistem secara bertahap. Fitur baru diuji dan diverifikasi sebelum digabungkan ke dalam sistem yang lebih besar, memungkinkan penambahan kebutuhan baru secara iteratif hingga produk selesai [10].

Dalam praktiknya, IID melibatkan identifikasi kebutuhan, analisis, spesifikasi desain, pengkodean, dan pengumpulan timbal balik dari stakeholder sebelum melanjutkan ke siklus *incremental* berikutnya. Komunikasi konstan antara pengembang dan stakeholder memastikan perangkat lunak yang dihasilkan berkualitas dan memenuhi kebutuhan pengguna.

E. Unified Modeling Language

Unified Modeling Language (UML) adalah bahasa standar dalam rekayasa perangkat lunak untuk merancang, menggambarkan, dan mendokumentasikan sistem perangkat lunak berorientasi objek secara visual. UML digunakan untuk dua tujuan utama dengan menghasilkan dua jenis diagram: structural diagrams (seperti class diagram) yang merujuk pada arsitektur proses atau entitas, dan behavioral diagrams (seperti use case, sequence, dan activity diagrams) yang mengacu pada aspek fungsional dari proses atau entitas [11].

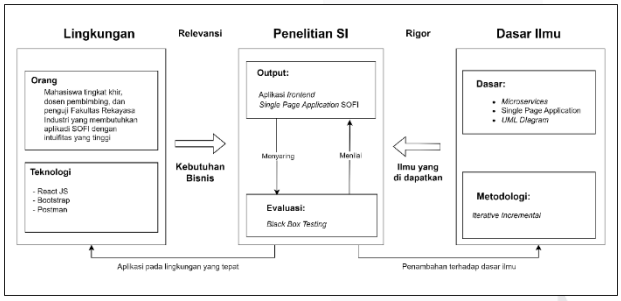
UML dikembangkan sebagai bahasa pemodelan umum yang terbuka dan berdasarkan kesepakatan mayoritas komunitas komersial, mampu menangani tantangan pengembangan perangkat lunak modern seperti skala besar, distribusi, pola, dan pengembangan tim. UML dirancang untuk tetap sederhana namun cukup ekspresif untuk memodelkan berbagai sistem praktis, termasuk konsep seperti concurrency, distribusi, encapsulation, dan komponen

[11]. Model bisnis yang baik mencakup semua informasi yang diperlukan untuk memahami struktur bisnis dan mode operasi, sehingga dapat menemukan solusi yang dapat diimplementasikan secara *real-time* [12].

F. User Acceptance Testing

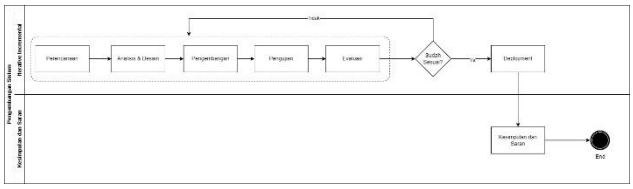
User Acceptance Testing (UAT) adalah pengujian akhir dalam pengembangan aplikasi untuk memvalidasi bahwa sistem sesuai dengan kebutuhan dan harapan pengguna. UAT memungkinkan tim uji, termasuk sponsor, untuk mengevaluasi kinerja perangkat lunak dan memastikan persyaratan sponsor diterjemahkan dengan akurat ke dalam desain sistem, sesuai dengan dokumen persyaratan yang disetujui [13]. UAT membantu dalam memvalidasi kebutuhan pengguna dengan memastikan aplikasi yang dikembangkan sesuai dengan kebutuhan dan harapan pengguna. Melalui UAT, bug dan masalah yang belum terdeteksi selama tahapan pengujian sebelumnya dapat diidentifikasi dan diperbaiki sebelum dideploy. Selain itu, UAT membantu dalam mengurangi risiko dan biaya dengan mengidentifikasi dan memperbaiki masalah sebelum aplikasi diluncurkan, sehingga mengurangi risiko kegagalan dan biaya tambahan untuk perbaikan setelah peluncuran. Terakhir, UAT juga membantu dalam meningkatkan kepuasan pengguna dengan melibatkan mereka secara langsung dalam pengembangan aplikasi, sehingga memastikan bahwa aplikasi yang dihasilkan sesuai dengan kebutuhan dan harapan mereka.

III. METODE



GAMBAR 3
MODEL KONSEPTUAL

Pada Gambar 3 dijelaskan kerangka berpikir yang digunakan pada penelitian ini, yang bertujuan untuk menghasilkan aplikasi SOFI berbasis SPA dengna menggunakan *library* React Js. Penelitian ini menggunakan metode *Iterative Incremental* untuk pengembangan bertahap dan berulang, serta evaluasi melalui *black box* yaitu UAT guna memastikan setiap fungsionalitas berjalan dengan baik dan memenuhi kebutuhan pengguna. Kerangka ini terdiri dari tiga bagian utama: Lingkungan, yang melibatkan mahasiswa tingkat akhir, dosen, dan penguji dari Fakultas Rekayasa Industri serta teknologi seperti React JS, Bootstrap, dan Postman; Penelitian SI, yang berfokus pada pengembangan dan penilaian aplikasi; dan Dasar Ilmu, yang melibatkan konsep-konsep seperti *microservices*, SPA, dan diagram UML, dengan metodologi *Iterative Incremental* untuk meningkatkan pengetahuan dasar yang diperoleh selama penelitian.



GAMBAR 4
SISTEMATIKA PENYELESAIAN MASALAH

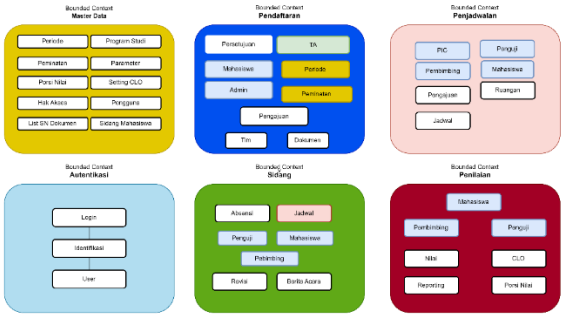
Pada Gambar 4 menjelaskan metode penelitian *Iterative Incremental* yang digunakan dalam pengembangan aplikasi. Proses ini dimulai pada tahap perencanaa, dimana kebutuhan dan tujuan ditentukan. Selanjutnya, dilakukan analisis dan desain untuk merancang solusi teknis untuk memenuhi kebutuhan yang telah diidentifikasi. Tahap berikutnya adalah pengembangan, di mana aplikasi dibangun secara bertahap dan berulang. Setelah itu, dilakukan pengujian untuk memastikan setiap fungsionalitas iterasi dapat bebas dari kesalahan. Evaluasi dilakukan untuk menilai kinerja dan kualitas aplikasi. Jika diperlukan, proses itasi akan terus berulang sampai setiap fungsionalitas dapat berjalan lancar dan sesuai dengan kebutuhan pengguna. Setelah evaluasi berhasil, aplikasi masuk ke tahap *deployment*. Di akhir, penelitian ini disimpulkan dengan memberikan kesimpulan dan saran berdasarkan hasil yang diperoleh salama pengembangan aplikasi.

IV. HASIL DAN PEMBAHASAN

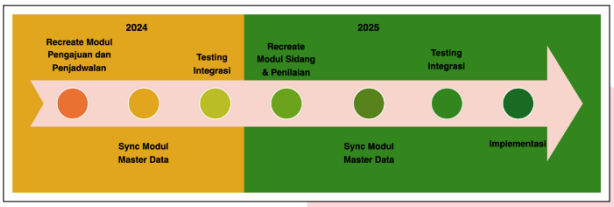
Pengembangan migrasi aplikasi SOFI dari MPA ke menggunakan metode pengembangan *iterative incremental* dengan penerapan *Domain-Driven Design*. Pendekatan ini menjamin solusi yang tidak hanya fungsional tetapi juga berfokus pada pengguna dengan memperhatikan kebutuhan bisnis secara mendalam. Domain-Driven Design memungkinkan pengembangan yang lebih fleksibel dan adaptif terhadap perubahan, dengan pemahaman yang mendalam tentang domain bisnis yang memungkinkan kolaborasi lebih efektif antara tim pengembang dan pemangku kepentingan.dan pengalaman secara mendalam.

A. Identifikasi Layanan

Analisis *Domain-Driven Design* (DDD) melibatkan diskusi intensif dengan ahli domain untuk mengaitkan konsep bisnis dengan implementasi teknis secara mendalam. Tujuan utama DDD adalah mempercepat pengembangan perangkat lunak yang berkaitan erat dengan domain bisnis yang kompleks, seperti aplikasi sidang fakultas SOFI. Dalam menangani masalah pengguna dan mengatasi perbedaan dalam konteks domain saat mengembangkan perangkat lunak skala besar, pendekatan *bounded context* digunakan. Pendekatan ini membagi model besar menjadi konteks-konteks kecil yang lebih mudah dikelola dan mendefinisikan hubungan antar setiap konteks.



GAMBAR 5
ANALISIS *DOMAIN DRIVEN DESIGN*

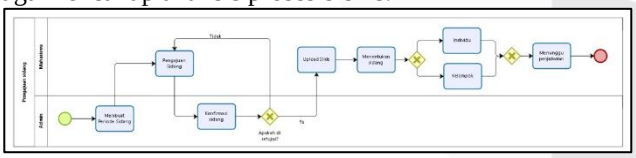


GAMBAR 6
ALUR MIGRASI APLIKASI SOFI

Pada Gambar 5 dan Gambar 6, menunjukkan hasil analisis DDD dengan *bounded context* pada aplikasi SOFI, dengan fokus pada *bounded context* Pengajuan dan Penjadwalan karena peran strategisnya dalam arsitektur sistem. Proses pengajuan dan penjadwalan tidak hanya penting untuk operasi bisnis utama tetapi juga berfungsi sebagai penghubung data penting untuk fungsi sistem lainnya. Penelitian ini berfokus pada pengembangan *bounded context* Pengajuan berdasarkan diskusi dengan ahli domain. Pengembangan awal *bounded context* Pengajuan dianggap kritis untuk memastikan aplikasi beroperasi secara efektif dan mencapai tujuan bisnis.

B. Initial Planning

Tahap *Initial Planning* merupakan tahap identifikasi kebutuhan fungsional untuk pengembangan aplikasi dengan menetapkan kebutuhan fungsionalitas atau kebutuhan aplikasi. Selain menetapkan kebutuhan fungsional, tahap ini juga mencakup analisis proses bisnis.



GAMBAR 7
PROSES BISNIS PENGAJUAN SIDANG

Pada Gambar 7, menjelaskan proses bisnis pengajuan sidang akhir pada aplikasi, dimana proses bisnis ini menggambarkan proses melakukan pengajuan sidang akhir pada aplikasi.

C. Tahap Perencanaan

Pada setiap fase iterasi, dilakukan tahap perencanaan yang bertujuan untuk merancang rencana migrasi aplikasi SOFI dari konsep MPA ke SPA. Kebutuhan fungsional aplikasi, yang tercantum dalam Tabel 1, diidentifikasi berdasarkan percobaan langsung pada aplikasi eksisting yang masih menggunakan konsep SPA. Tahap perencanaan ini memastikan setiap iterasi migrasi direncanakan dengan hati-hati agar transisi ke SPA berjalan lancar dan efisien, serta kebutuhan fungsional dapat terpenuhi sesuai dengan analisis mendalam dari aplikasi yang ada.

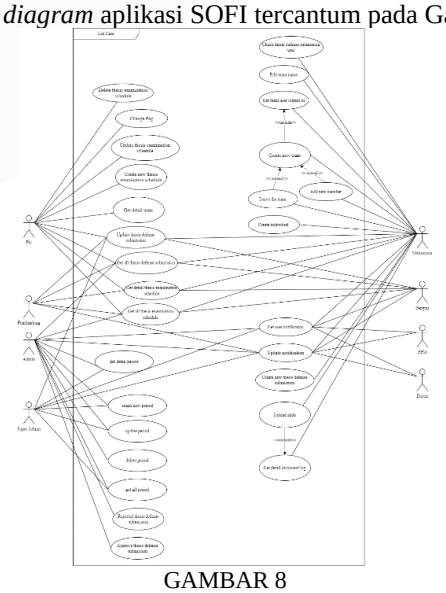
TABEL 1
KEBUTUHAN FUNGSIONAL

ID	Kebutuhan
REQ-02.01	*Get all thesis defense submission*
REQ-02.02	*Check thesis defense submission user*
REQ-02.03	*Create new thesis defense submission*
REQ-02.04	*Update thesis defense submission*
REQ-03.01	*Approve thesis defense submission*
REQ-03.02	*Rejected thesis defense submission*
REQ-04.01	*Get detail team*
REQ-04.02	*Get team user logged in*
REQ-04.03	*Create new team*
REQ-04.04	*Create individual*
REQ-04.05	*Add new member*
REQ-04.06	*Leave the team*
REQ-04.07	*Edit team name*
REQ-05.01	*Get detail document logs*
REQ-05.02	*Upload slide*
REQ-06.01	*Get user notification*
REQ-06.02	*Update notification*

D. Tahap Analisis

Tahap kedua dalam *Iterative Incremental* adalah tahap Analisis. Pada tahap ini, dilakukan analisis mendalam serta perancangan kebutuhan yang telah ditetapkan pada tahap sebelumnya. Analisis ini bertujuan untuk memahami secara detail kebutuhan fungsional yang harus dipenuhi oleh aplikasi, dan menghasilkan output berupa use case diagram.

Use case diagram digunakan untuk menggambarkan fitur-fitur yang tersedia bagi setiap pengguna dalam aplikasi SOFI. Perancangan *use case diagram* ini dibuat berdasarkan kelompok kebutuhan fungsional yang sudah ditetapkan, memberikan gambaran visual yang jelas tentang interaksi pengguna dengan sistem serta fitur-fitur yang dapat diakses oleh setiap jenis pengguna. *Use case diagram* aplikasi SOFI tercantum pada Gambar 8.



GAMBAR 8
USE CASE DIAGRAM

E. Tahap Pengembangan

Tahap ketiga dalam *Iterative Incremental* adalah tahap Pengembangan. Pada tahap ini, *frontend* dikembangkan sesuai dengan kebutuhan fungsional, serta rancangan UML yang telah disiapkan pada tahap perencanaan, analisis, dan perancangan sebelumnya. Pengembangan *frontend* dilakukan menggunakan bahasa pemrograman Javacript dengan *library* React js.

Proses pengembangan ini melibatkan kolaborasi dengan tim *frontend*, menggunakan *repository* di GitHub untuk memfasilitasi kerja sama dan pengelolaan kode.

F. Tahap Pengujian

Setelah tahap pengembangan selesai, tahap berikutnya adalah pengujian terhadap sistem yang telah dimigrasi. Pengujian ini bertujuan untuk memastikan bahwa fungsionalitas yang telah dimigrasi berjalan sesuai harapan. Dari sisi *frontend*, pengujian ini dilakukan menggunakan metode *User Acceptance Testing* (UAT). Tabel 2 hingga Tabel 8 menjelaskan hasil pengujian UAT dari layanan yang telah dimigrasi.

TABEL 2
HASIL UAT FASE PERTAMA

Test Case ID	ID	Status
TC-01.01	REQ-02.01	*Pass* ✅
TC-01.02		*Pass* ✅
TC-01.03		*Pass* ✅
TC-01.04	REQ-02.02	*Pass* ✅
TC-01.05	REQ-02.03	*Pass* ✅
TC-01.06	REQ-02.04	*Pass* ✅
TC-02.01	REQ-03.01	*Pass* ✅
TC-02.02		*Pass* ✅
TC-02.03	REQ-03.02	*Pass* ✅
TC-02.04		*Pass* ✅

TABEL 3
HASIL UAT FASE KEDUA

Test Case ID	ID	Status
TC-02.01	REQ-04.02	*Pass* ✅
TC-02.02	REQ-04.03	*Pass* ✅
TC-02.03	REQ-04.04	*Pass* ✅
TC-02.04		*Pass* ✅
TC-02.05	REQ-04.05	*Pass* ✅
TC-02.06		*Pass* ✅
TC-02.07	REQ-04.06	*Pass* ✅
TC-02.08		*Pass* ✅
TC-02.09	REQ-04.07	*Pass* ✅
TC-02.10	REQ-05.01	*Pass* ✅
TC-02.11	REQ-05.02	*Pass* ✅
TC-02.12	REQ-06.01	*Pass* ✅
TC-02.13	REQ-06.02	*Pass* ✅

Berdasarkan hasil UAT yang telah dilakukan, dapat disimpulkan bahwa semua fitur yang dikembangkan pada fase pertama dan kedua telah berfungsi dengan baik sesuai dengan fungsionalitas dan tujuan yang telah ditetapkan sebelumnya. Hasil pengujian yang dijelaskan pada Tabel 2 dan Tabel 3 menunjukkan bahwa semua *test case* yang diuji telah berhasil mencapai tingkat keberhasilan 100% dalam memenuhi kebutuhan pengguna. Dengan demikian, fase berikutnya akan difokuskan pada pengembangan fungsionalitas atau tujuan yang telah ditetapkan untuk fase tersebut, memastikan setiap aspek sistem yang sedang dikembangkan memenuhi ekspektasi dan kebutuhan pengguna dengan sempurna. Iterasi akan dilanjutkan jika di masa mendatang ditemukan fitur baru yang perlu dikembangkan kembali.

G. Tahap Evaluasi

Berdasarkan hasil UAT dengan metode *iterative incremental*, dapat disimpulkan bahwa semua fitur yang dikembangkan pada fase pertama telah berfungsi dengan baik sesuai dengan tujuan yang telah ditetapkan sebelumnya. Hasil pengujian menunjukkan bahwa setiap test case yang diuji berhasil memenuhi kebutuhan pengguna, sebagaimana dijelaskan dalam Tabel 2, dengan tingkat keberhasilan mencapai 100%. Hal ini menunjukkan bahwa semua fungsionalitas yang diharapkan dari sistem telah terpenuhi tanpa kekurangan atau bug yang signifikan, sehingga tim pengembang dapat melanjutkan ke fase berikutnya.

Pada tahap pengujian fase kedua, hasil evaluasi menunjukkan bahwa semua fitur yang dikembangkan telah berfungsi dengan baik dan sesuai dengan kebutuhan serta fungsionalitas yang diharapkan oleh pengguna. Semua test case yang diuji juga berhasil mencapai tingkat keberhasilan 100%, sebagaimana dijelaskan dalam Tabel 3. Dengan demikian, fase kedua ini dinyatakan selesai karena semua fitur yang direncanakan telah berhasil diimplementasikan dan diuji dengan sukses. Iterasi berikutnya akan tetap menggunakan pendekatan iterative incremental untuk memastikan setiap fitur baru memenuhi standar dan kebutuhan pengguna dengan baik, terus meningkatkan kualitas sistem di setiap iterasi.

H. Tahap *Deployment*

Tahap *Deployment* dalam *iterative incremental* dilakukan setelah semua iterasi selesai. Aplikasi SOFI di-deploy menggunakan platform Vercel, yang menyediakan hosting gratis untuk aplikasi web modern dengan spesifikasi termasuk *bandwidth*, *build*, *serverless* functions, dan penyimpanan yang memadai untuk skala kecil. Integrasi dengan GitHub memungkinkan proses build dan deployment otomatis. Vercel menyediakan dashboard untuk memantau status deployment dan logs, serta opsi peningkatan sistem melalui model "*pay as you go*" untuk skalabilitas yang lebih besar seiring perkembangan aplikasi.

V. KESIMPULAN

Penelitian ini menunjukkan bahwa migrasi aplikasi SOFI dari MPA ke SPA berhasil dan memenuhi kebutuhan pengguna, terbukti dari hasil UAT yang menunjukkan 100% keberhasilan test case pada dua iterasi. Penerapan metode *Iterative Incremental* efektif dalam memfasilitasi migrasi, memastikan pengembangan yang terstruktur dan sesuai dengan kebutuhan bisnis. Dengan memecah sistem menjadi bagian-bagian kecil yang dapat dikelola, setiap langkah dapat dievaluasi dan disesuaikan secara berkelanjutan, meningkatkan kemampuan aplikasi untuk menanggapi

perubahan kebutuhan bisnis dan umpan balik pengguna secara fleksibel.

REFERENSI

[1] O. Al-Debagy and P. Martinek, "A comparative review of microservices and monolithic architectures," in *2018 IEEE 18th International Symposium on Computational Intelligence and Informatics (CINTI)*, IEEE, 2018, pp. 149–154.

[2] K. Gos and W. Zabierowski, "The Comparison of Microservice and Monolithic Architecture," in *International Conference on Perspective Technologies and Methods in MEMS Design*, 2020, pp. 150–153. doi: 10.1109/MEMSTECH49584.2020.9109514.

[3] N. S. T. R. Sangati, "Web Application Development using SpringBoot and Angular," *INTERANTIONAL JOURNAL OF SCIENTIFIC RESEARCH IN ENGINEERING AND MANAGEMENT*, vol. 06, no. 06, 2022, doi: 10.55041/ijsrem14292.

[4] O. J. Okesola, A. A. Adebiyi, A. A. Owoade, O. Adeaga, O. Adeyemi, and I. Odun-Ayo, *Software Requirement in Iterative SDLC Model*, vol. 1224 AISC, no. November. Springer International Publishing, 2020. doi: 10.1007/978-3-030-51965-0_2.

[5] H. Asrohah, M. Khusnu Milad, A. T. Wibowo, and E. I. Rhofita, "Improvement of Academic Services using Mobile Technology based on Single Page Application," *Telfor Journal*, vol. 12, no. 1, pp. 62–66, 2020, doi: 10.5937/TELFOR2001062A.

[6] S. Abrahamsson, "A model to evaluate front-end frameworks for single page appli- cations written in JavaScript," 2023, [Online]. Available: https://www.diva-portal.org/smash/get/diva2:1758858/FULLTEXT01.pdf

[7] I. Khadka, "Converting Multipage Application to Single Page Application," no. March, 2016, [Online]. Available: https://www.theseus.fi/bitstream/handle/10024/106770/Coverting+Multipage+Application+to+Single+Page+Application.pdf?sequence=1

[8] A. Gupta, A. Rawal, and Y. Barge, "Comparative Study of Different SDLC Models," no. November, 2021.

[9] O. E. Olorunshola and F. N. Ogwueleka, "Review of system development life cycle (SDLC) models for effective application delivery," in *Information and Communication Technology for Competitive Strategies (ICTCS 2020) ICT: Applications and Social Interfaces*, vol. 1, no. Ictcs, Springer Singapore, 2022, pp. 281–289. [Online]. Available: https://d1wqtxts1xzle7.cloudfront.net/78365340/Comparative_Study_of_Different_SDLC_Models-libre.pdf?1641675307=&response-content-disposition=inline%3B+filename%3DComparative_Study_of_Different_SDLC_Mode.pdf&Expires=1720777392&Signature=AfVC9YrRXCxrb22i8g4

[10] I. M. Ibrahim, O. F. Nonyelum, and I. R. Saidu, "Iterative and Incremental Development Analysis Study of Vocational Career Information Systems," *International Journal of Software Engineering & Applications*, vol. 11, no. 5, pp. 13–24, 2020, doi: 10.5121/ijsea.2020.11502.

[11] J. R. G. Jacobson, L., & Booch, *The unified modeling language reference manual.*, vol. 44, no. 8. 2021. doi: 10.1088/1751-8113/44/8/085201.

[12] C. A. Borcosi, "The importance of business modeling using the unified modeling language (UML)," vol. 2, no. November, pp. 91–101, 2022, doi: 10.38173/RST.2022.24.2.7.

[13] S. Gordon *et al.*, "Best Practice Recommendations: User Acceptance Testing for Systems Designed to Collect Clinical Outcome Assessment Data Electronically," *Ther Innov Regul Sci*, vol. 56, no. 3, 2022, doi: 10.1007/s43441-021-00363-z.