Implementasi Session Management Pada Website Magang Menggunakan Teknologi MERN

Rizatul Mas Ulah[1], I Made Suartana[2]

[1,2] Program Studi S1 Teknik Informatika, Universitas Negeri Surabaya
[1]rizatul.20004@mhs.unesa.ac.id
[3]madesuartana@unesa.ac.id

***Abstrak*—Website merupakan salah satu media utama yang digunakan untuk menyebarkan berbagai informasi secara online. Informasi terkait kegiatan magang yang diselenggarakan oleh Dinas Komunikasi dan Informatika Kabupaten Lamongan akan disampaikan melalui website magang yang dirancang dengan teknologi MERN (MongoDB, Express.js, React, dan Node.js). Salah satu aspek penting dalam perancangan website adalah manajemen sesi atau session management. Sesi menjadi hal yang penting karena melibatkan informasi sensitif seperti data login, preferensi pengguna, dan riwayat aktivitas. Penerapan manajemen sesi yang baik bertujuan untuk menghindari potensi risiko keamanan di masa depan dan memastikan data atau informasi tetap aman serta terlindungi dari akses yang tidak sah. Penelitian ini berhasil mengimplementasikan autentikasi, otorisasi, dan express-session sebagai bagian dari session management menggunakan teknologi MERN. Hasil pengujian menunjukkan bahwa sistem berhasil mengelola manajemen sesi dengan baik melalui autentikasi token dan session ID. Sistem juga mampu memberikan kontrol akses yang tepat serta menjaga keamanan dan integritas data pengguna. Secara keseluruhan, sistem memenuhi skenario yang dirancang, memastikan bahwa data dan informasi pengguna tetap aman dan terlindungi dari akses yang tidak sah.**

***Kata Kunci*— Website, Authentication, Authorization, Session Management, MERN Stack, Keamanan Data.**

I. PENDAHULUAN

Perkembangan teknologi informasi dan komunikasi terus mengalami pertumbuhan pesat, menyesuaikan dengan kebutuhan manusia yang semakin maju dan modern. Perkembangan teknologi telah menghasilkan perubahan besar dalam cara berkomunikasi dan berinteraksi. Media promosi tidak lagi terbatas pada format cetak atau metode konvensional. Website menjadi salah satu media online yang digunakan untuk menyebarkan berbagai informasi di internet [1]. Dalam perancangan website, terdapat beberapa masalah yang perlu diperhatikan, salah satunya terkait manajemen sesi atau session management. Session merepresentasikan durasi waktu di mana pengguna aktif berinteraksi dengan sebuah sistem dan berakhir ketika pengguna keluar dari sistem tersebut [2]. Session management menjadi penting karena melibatkan informasi sensitif seperti penyimpanan token, informasi login, preferensi pengguna, dan riwayat aktivitas.

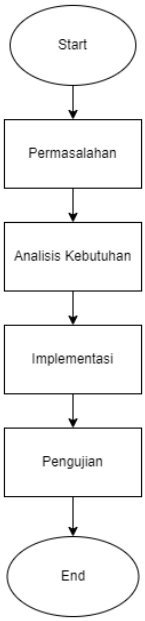
Dinas Komunikasi dan Informatika kabupaten Lamongan adalah salah satu lembaga instansi pemerintahan yang menyediakan tempat magang untuk peserta magang dari kalangan pelajar maupun mahasiswa di Kabupaten Lamongan. Proses pendaftaran, informasi penerimaan hingga informasi kegiatan selama periode magang akan disampaikan melalui website yang akan dirancang. Dalam perancangan website magang, terdapat beberapa standar keamanan yang harus diterapkan. Standar keamanan yang harus diterapkan pada website mencakup beberapa aspek penting. Mekanisme manajemen sesi dapat diterapkan sebagai bagian dari kemanan data dalam website yang akan dirancang [3]. Manajemen sesi adalah proses pengelolaan status dan data pengguna selama berinteraksi dengan sistem atau aplikasi. Tujuan manajemen sesi adalah untuk memastikan bahwa informasi dan aktivitas pengguna dapat dipertahankan dengan aman serta efisien sepanjang penggunaan aplikasi [4].

MERN Stack merupakan sebuah rangkaian teknologi yang terdiri dari empat komponen utama, yaitu MongoDB, Express.js, React, dan Node.js, yang dan digunakan bersama-sama dalam pengembangan aplikasi web modern. Stack teknologi ini menyediakan fondasi yang solid dan terintegrasi, memungkinkan pengembang untuk membangun aplikasi web yang efisien dan responsif menggunakan satu bahasa pemrograman yaitu JavaScript [5]. Website yang akan dirancang menggunakan teknologi MERN, memerlukan skema authentication sebagai proses keamanan pertama untuk melindungi data di dalamnya. Authentication atau autentikasi adalah proses verifikasi atau pembuktian identitas atau kredensial seseorang yang mencoba masuk ke dalam sebuah sistem. Hal ini bertujuan untuk memastikan bahwa individu yang mencoba masuk benar-benar adalah pemilik akun yang sah [6]. Pada website yang akan dirancang, terdapat beberapa user yang memiliki hak ases berbeda terhadap sistem. Untuk mengatasi permasalahan ini, dibutuhkan sebuah mekanisme yaitu authorization. Authorization atau Otorisasi merupakan suatu mekanisme untuk memastikan bahwa pengguna ssitem atau aplikasi hanya dapat mengakses fitur secara terbatas sesuai dengan peran atau role yang sudah ditentukan. Hal ini bertujuan agar pengguna hanya dapat mengakses data dan fitur yang relevan dengan perannya, sehingga mengurangi risiko kebocoran atau penyalahgunaan informasi [7]. Express Session adalah modul middleware yang memungkinkan pengelolaan sesi pengguna secara efektif dalam aplikasi web. Dengan Express Session, kita dapat membuat, mengelola, dan menyimpan sesi pengguna secara aman dalam server [8].

Penelitian ini bertujuan untuk merancang sebuah website magang menggunakan teknologi MERN dan mengimplementasikan manajemen sesi pengguna untuk mengelola keamanan dalam perancangan website [9]. Manajemen sesi yang diimplementasikan meliputi proses authentikasi, otorisasi serta pengelolaan sesi. Dengan mengimplementasikan autentikasi, otorisasi dan express session dalam website magang yang dirancang, informasi

sensitif seperti token autentikasi dan data login peserta magang dapat diamankan dengan baik. Sesi pengguna akan diatur dengan aman dalam server, mengurangi risiko kebocoran informasi dan mencegah serangan yang berpotensi merugikan, seperti pencurian identitas atau akses tidak sah [10]. Express Session diharapkan dapat menjadi solusi yang efektif untuk mengatasi berbagai tantangan terkait dengan pengelolaan sesi pengguna di sisi backend dan dapat meningkatkan pengalaman pengguna secara keseluruhan dalam menerima informasi seputar magang dari website serta memperbaiki pengalaman mereka selama menjalani periode magang [11]. Dengan demikian, pengguna dapat lebih nyaman dan efisien dalam mengakses informasi dan berinteraksi dengan sistem website magang tersebut.

II. METODOLOGI PENELITIAN



Gbr. 1 Alur Penelitian

Bab ini akan menjelaskan mengenai alur penelitian. Alur penelitian merupakan rencana atau langkah-langkah yang akan diambil pada suatu penelitian. Adapun pembuatan alur penelitian bertujuan untuk memberikan gambaran kerangka kerja yang terstruktur seperti pada Gbr. 1.

A. Analisis Kebutuhan

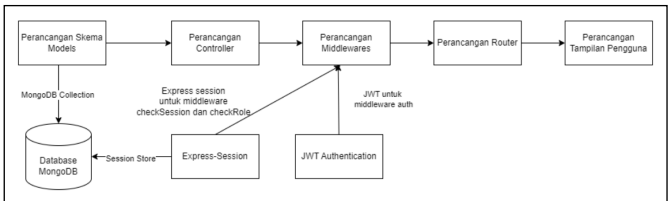
Analisis kebutuhan digunakan untuk memahami dan mendokumentasikan kebutuhan-kebutuhan pengguna serta persyaratan fungsional dan non-fungsional yang harus dipenuhi oleh sistem yang akan dikembangkan seperti pada tabel 1. Tujuan utama dari analisis kebutuhan adalah untuk menetapkan apa yang diinginkan oleh pengguna dari sistem yang akan dibangun, sehingga pengembang dapat merancang solusi yang sesuai dan memenuhi harapan pengguna.

TABEL I
KEBUTUHAN FUNGSIONAL DAN NON FUNGSIONAL

Kebutuhan Fungsional	Kebutuhan Non Fungsional
Sistem mampu memvalidasi login	Sistem menampilkan user interface yang menarik
Sistem mampu membuat token JWT dan session ID di sisi server	
Sistem mampu mengelola sesi pengguna	
Sistem mampu menghapus sesi pengguna	Tidak ada bug dalam sistem
Sistem mampu mengotorisasi user	
Sistem mampu melakukan CRUD	

B. Implementasi Sistem

MERN stack digunakan sebagai teknologi utama dalam perancangan website magang Dinas Komunikasi dan Informatika Kabupaten Lamongan. Dalam implementasi teknologi MERN, MongoDB berfungsi sebagai basis data NoSQL untuk penyimpanan data, Express.js sebagai framework untuk server-side, React untuk pengembangan antarmuka pengguna, dan Node.js untuk menjalankan server dan mengelola dependensi.



Gbr. 2 Implementasi Sistem

Perancangan website menggunakan teknologi MERN dimulai dari perancangan schema models yang disimpan di database MongoDB, perancangan controller yang menangani permintaan atau request dari pengguna, perancangan middleware yang mencakup penggunaan JWT dan Express-Session, perancangan router aplikasi hingga perancangan tampilan pengguna seperti pada gambar 2. Berikut penjelasan setiap tahapannya

1. Perancangan Schema Models

Dalam merancang schema models, langkah pertama adalah menentukan struktur data dan atribut yang diperlukan dalam database MongoDB. Models bertanggung jawab untuk mengelola data dan struktur database, memastikan data yang diterima dari pengguna valid. Setiap model akan mencakup atribut-atribut yang relevan dan validasi yang diperlukan untuk memastikan integritas data. Penggunaan Express.js sebagai framework server-side memastikan bahwa aplikasi dapat menangani permintaan dengan efisien dan skalabilitas tinggi, sementara MongoDB menyediakan solusi database yang fleksibel dan kuat untuk mengelola data yang kompleks

2. Perancangan Controller

Bagian *Controller* berfungsi menerima *request* dari pengguna, memproses data yang diperlukan dengan bantuan *Models* dan bertanggung jawab untuk menangani logika aplikasi dan mengatur interaksi antara *Model* dan *View* yang akan ditampilkan langsung kepada pengguna. Dengan memanfaatkan Models, Controller memproses data yang diminta, baik untuk membaca, menambahkan, mengubah, atau menghapus informasi dari database. Setiap Controller biasanya terdiri atas beberapa metode yang menangani berbagai jenis permintaan seperti GET, POST, PUT, dan DELETE. Perancangan Controller yang baik memastikan bahwa aplikasi tidak hanya responsif terhadap permintaan pengguna, tetapi juga mudah diintegrasikan dengan bagian-bagian lain dari aplikasi. Dengan struktur yang terdefinisi dengan baik ini, aplikasi tidak hanya akan mudah untuk dikembangkan tetapi juga mudah untuk diperluas dan dipelihara.

3. Implementasi JWT Untuk Authentication

Pada perancangan website magang, *JWT (JSON Web Token)* digunakan untuk mengautentikasi pengguna. Proses autentikasi dilakukan saat pengguna memasukkan kredensial berupa *username* dan *passwordm*yang valid pada halaman *login*. Setelah kredensial diverifikasi oleh server, server akan menghasilkan *JWT Sign* berupa token string. Token ini berisi informasi tentang identitas pengguna dan hak aksesnya. JWT ini kemudian disematkan ke dalam respons yang dikirimkan kembali ke klien setelah proses login berhasil. Klien akan menyimpan token JWT ini, dan akan menyertakannya dalam setiap permintaan yang dikirimkan ke server setelahnya. Pada sisi server, setiap permintaan yang diterima akan diverifikasi dengan memeriksa keaslian token JWT yang disertakan dalam *header* permintaan. Jika token valid dan tidak kadaluarsa, server akan mengizinkan akses ke sumber daya yang diminta oleh pengguna. Dengan menggunakan JWT, proses autentikasi dapat dijalankan secara efisien dan aman.

4. Implementasi Authorization

A*uthorization* atau otorisasi akan diimplementasikan menggunakan *middleware* yang disebut *"auth middleware"*. *Middleware* ini ditempatkan di antara permintaan yang masuk dan berfungsi untuk memeriksa izin atau hak akses pengguna sebelum memungkinkan akses ke sumber daya atau fitur tertentu. Dalam sistem yang akan dirancang, terdapat beberapa pengguna dengan hak akses yang berbeda sehingga proses otorisasi diperlukan. Untuk mendukung proses otorisasi, informasi peran (*role*) pengguna akan diambil dari token JWT (*JSON Web Token*) atau *cookie session* yang disimpan setelah proses autentikasi. Setelah pengguna berhasil login, informasi tentang pengguna, termasuk peran mereka, akan dienkripsi dan disimpan dalam token JWT dan *cookie browser*. Token dan *cookie* ini kemudian dikirim dalam header setiap permintaan berikutnya. *Middleware* otorisasi akan memverifikasi token JWT dan *cookie* untuk memastikan bahwa pengguna memiliki izin yang sesuai untuk mengakses fitur atau sumber daya tertentu dalam aplikasi.

5. Implementasi Express-Session

Express Session adalah sebuah *middleware* untuk aplikasi web Node.js yang berbasis Express.js yang digunakan untuk mengelola sesi pengguna. Sesi pengguna adalah cara untuk menyimpan data pengguna di server antara permintaan, yang berguna untuk mengidentifikasi pengguna dan menyimpan informasi tentang status masuk, preferensi pengguna, atau data sesi lainnya. Ketika pengguna mengirimkan permintaan, server dapat memeriksa informasi yang tersimpan dalam objek sesi untuk menentukan apakah pengguna memiliki izin yang cukup untuk mengakses sumber daya yang diminta. Setiap kali sesi baru dibuat, *Express Session* menghasilkan ID sesi unik. ID ini digunakan untuk mengidentifikasi sesi pengguna tertentu di server. ID sesi ini disimpan dalam *cookie* di sisi klien. Ketika pengguna melakukan permintaan selanjutnya, *cookie* ini akan dikirim kembali ke server, yang kemudian dapat digunakan untuk mengaitkan permintaan tersebut dengan sesi yang relevan.

6. Perancangan Router dan HTTP Request Method

HTTP *(Hypertext Transfer Protocol)* merupakan protokol yang dirancang untuk memfasilitasi komunikasi antara pengguna dan server di lingkungan web. Fungsi utama HTTP adalah sebagai protokol untuk merespons permintaan yang dikirimkan antara pengguna dan server. Pengguna akan mengirimkan permintaan HTTP ke server, kemudian server merespons dengan memberikan respons yang berisi informasi status terkait permintaan tersebut. Dalam pengembangan aplikasi berbasis MERN, *router* akan mengelola permintaan HTTP dan mengarahkan ke fungsi yang sesuai di dalam *controller*. Dengan menggunakan *router*, aplikasi dapat menentukan bagaimana setiap permintaan HTTP harus ditangani berdasarkan *URL* dan metode HTTP yang digunakan. *Router* memastikan bahwa setiap permintaan diarahkan ke *endpoint* yang benar, di mana logika bisnis yang sesuai dijalankan. *Router* memainkan peran penting dalam menjaga struktur dan organisasi aplikasi, memastikan bahwa setiap permintaan ditangani dengan cara yang efisien dan konsisten.

7. Session Management Schema

Ketika pengguna berhasil login, *JSON Web Token (JWT)* dan *Session ID* akan diberikan sebagai bagian dari proses autentikasi. Proses autentikasi melibatkan pencocokan kredensial *login* yang dimasukkan pengguna dengan data yang tersimpan di dalam *database*. Apabila data yang diinputkan oleh pengguna tidak sama dengan data yang ada pada *database,* maka pengguna akan diberikan pesan status error dan permintaan login dengan kredensial yang valid. Autentikasi dan otorisasi akan diterapkan di seluruh permintaan untuk memastikan pengguna tetap terautentikasi dan terotorisasi saat mengakses berbagai bagian dari aplikasi web. Karena protokol HTTP tidak menyimpan status sebelumnya *(stateless protocol),* maka untuk memastikan pengguna tetap terotentikasi di setiap halaman, diperlukan mekanisme *session* dan *cookie*.

Server akan membuat *session* sebagai tanda bahwa pengguna telah login ke dalam *website.* Session ini berupa ID unik yang berisi informasi tentang pengguna seperti id pengguna dan role. *Session ID* akan berlaku dalam jangka

waktu yang telah ditentukan. Untuk menghubungkan setiap *browser* dengan *session ID*, kita menggunakan *cookies* yang berisi *session ID* tersebut. Setiap kali terjadi transaksi data, *browser* akan mengirimkan *cookies* dan memeriksa isinya untuk dibandingkan dengan *session ID*. Jika *session ID* yang berada di dalam *cookies* sesuai dengan yang ada di server, maka pengguna dianggap terotentikasi dan dapat diotorisasi untuk mengakses halaman yang diminta. *Cookies* yang menyimpan session ID harus dienkripsi untuk mencegah pemalsuan dan pencurian. *Flag secure* pada *cookies* juga harus diatur untuk memastikan cookies hanya dikirim melalui koneksi HTTPS dan tidak dapat diakses melalui JavaScript, mengurangi risiko serangan *man-in-the-middle* dan XSS (*Cross-Site Scripting*). Dengan menggabungkan autentikasi JWT, otorisasi berdasarkan peran, dan manajemen sesi menggunakan *Express-session*, aplikasi web dapat memastikan bahwa pengguna tetap terautentikasi dan terotorisasi dengan aman di setiap halaman. Langkah-langkah keamanan ini akan membantu melindungi integritas dan kerahasiaan sesi pengguna serta memastikan aplikasi tetap aman dari berbagai ancaman.

8. Perancangan Tampilan Pengguna dan Session Management di Frontend

Pada proses perancangan tampilan pengguna, *Framework* React.js diimplementasikan menggunakan vite.js untuk membuat *frontend.* Vite.js adalah alat build modern yang mempercepat proses pengembangan dengan menyediakan waktu pemuatan yang sangat cepat dan pengalaman pengembangan yang responsif. Kelebihan Vite.js termasuk optimasi bundling yang efisien dan konfigurasi yang sederhana. Penggunaan *Bootstrap* juga diterapkan untuk menciptakan antarmuka yang menarik dan responsif. Dengan kombinasi React, Bootstrap, dan JavaScript, diharapkan dapat menghasilkan tampilan antarmuka yang menarik dan berfungsi dengan baik, sehingga pengguna dapat mengakses informasi magang dengan mudah dan nyaman. Selain itu, *session management* juga diterapkan dibagian *frontend* melalui proses autentikasi pengguna dan otorisasi berdasarkan peran. Autentikasi dan otorisasi dilakukan dengan menggunakan token JTW dan *cookie* yang disertakan pada setiap permintaan. *Middleware* di sisi frontend akan memeriksa *role* yang dibawa oleh *cookie* sebelum memberikan akses ke halaman atau fitur tertentu. Hal ini memastikan bahwa hanya pengguna dengan izin yang sudah ditentukan yang dapat mengakses sumber daya tertentu di bagian frontend. Proses ini merupakan bagian integral dari pengembangan website magang, yang bertujuan untuk memberikan pengalaman pengguna yang optimal dan memenuhi kebutuhan informasi peserta magang secara efektif.

C. Pengujian Sistem

Dalam penelitian ini, pengujian sistem dilakukan untuk memastikan bahwa sistem yang dirancang berjalan dengan baik dan sesuai dengan kebutuhan pengguna. Pengujian akan dilakukan menggunakan metode Black Box Testing dan Session Management Testing.

1. Blackbox Testing

Black Box Testing adalah pendekatan pengujian perangkat lunak yang hanya berfokuskan pada spesifikasi fungsional tanpa perlu mengeksplorasi atau menguji kode atau sisi internal programnya. Dalam pengujian blackbox, hanya aspek-aspek seperti fungsi, antarmuka, dan alur aplikasi yang diuji tanpa memperhatikan detail implementasi di dalamnya [12]. Pengujian dilakukan dengan menentukan hasil yang diharapkan ketika perangkat lunak dijalankan, tanpa memperhatikan bagaimana aplikasi mencapai hasil tersebut secara internal. Dengan pendekatan ini, pengujian dapat dilakukan secara independen dari pengembang dan dapat lebih fokus pada pengalaman pengguna dan pemenuhan spesifikasi fungsional.

2. Session Management Testing

Session Management Schema adalah struktur atau rencana yang digunakan untuk mengatur dan mengelola sesi pengguna dalam suatu sistem dengan memperhatikan keamanan [13]. Menurut OWASP (*Open Web Application Security Project*), manajemen sesi yang aman melibatkan praktik-praktik seperti penggunaan ID sesi yang unik, enkripsi cookie, waktu kadaluwarsa sesi, dan pengujian keamanan berkala terhadap sistem untuk memastikan tidak ada kerentanan dalam manajemen sesi yang dapat dieksploitasi [14]. Dalam penelitian ini, pengujian dilakukan menggunakan OWASP Web Security Testing Guide (WSTG) dengan beberapa kategori yang digunakan yaitu pengujian autentikasi, manajemen sesi dan API. Dengan berfokus pada pengujian-pengujian ini, penelitian dapat tetap efisien dalam mengidentifikasi potensi kerentanan dan mampu memberikan kontribusi signifikan dalam meningkatkan keamanan manajemen sesi.

III. HASIL DAN PEMBAHASAN

Bab ini akan menjelaskan hasil implementasi *Session Management* pada website magang Dinas Komunikasi dan Informatika Kabupaten Lamongan menggunakan teknologi MERN. Dimulai dari perancangan pada *backend* yang mencakup penggunaan *middleware auth, checkRole* dan *middleware session*, implementasi react untuk *frontend,* implementasi *session management* di *frontend* hingga unit testing untuk sistem yang sudah dirancang.

A. Perancangan Backend

1. Inisialisasi Project dan Instalasi Dependensi

Inisialisasi *project* adalah langkah awal dalam proses pengembangan aplikasi yang bertujuan untuk menyiapkan struktur dasar serta menginstalasi dependensi yang dibutuhkan. Proses ini dimulai dengan menggunakan perintah *npm init* untuk membuat file package.json, yang menyimpan metadata proyek seperti nama, versi, deskripsi, dan daftar dependensi. Setelah itu, dependensi utama diinstal menggunakan perintah *npm install* untuk mendukung fungsionalitas aplikasi. Beberapa dependensi dan versi yang digunakan ditampilkan seperti pada Gbr. 3.

```
"dependencies": {
  "bcryptjs": "^2.4.3",
  "body-parser": "^1.20.2",
  "config": "^3.3.12",
  "connect-mongo": "^5.1.0",
  "cookie-parser": "^1.4.6",
  "cors": "^2.8.5",
  "dotenv": "^16.4.5",
  "express": "^4.19.2",
  "express-session": "^1.18.0",
  "http-status": "^1.7.4",
  "imagekit": "^5.0.1",
  "jose": "^5.9.3",
  "jsonwebtoken": "^9.0.2",
  "mongodb": "^6.8.0",
  "mongoose": "^8.5.0",
  "mongoose-sequence": "^6.0.1",
  "multer": "^1.4.2",
  "nodemon": "^3.1.4"
```

Gbr. 3 Dependensi Backend

Secara keseluruhan, proses inisialisasi dan konfigurasi ini memastikan bahwa proyek memiliki struktur yang solid dan pengaturan yang aman, memfasilitasi pengembangan aplikasi yang efisien dan terkelola dengan baik.

2. Konfigurasi Database MongoDB

Konfigurasi *database* bertujuan untuk menghubungkan aplikasi Node.js dengan *database* MongoDB menggunakan Mongoose. Konfigurasi diimplementasikan pada fungsi *connectDB* dan ditampilkan seperti seperti pada Gbr 4.

```
const mongoose = require('mongoose');

const connectDB = async () => {
  try {
    await mongoose.connect(process.env.MONGO_URL, {
      useNewUrlParser: true,
      useUnifiedTopology: true
    });
    console.log('MongoDB connected successfully');
  } catch (err) {
    console.error(err.message);
    process.exit(1);
  }
};

module.exports = connectDB;
```

Gbr.4 Fungsi ConnectDB

Dengan menggunakan *mongoose.connect*, aplikasi terhubung ke MongoDB dengan opsi *useNewUrlParser* dan *useUnifiedTopology*, yang memastikan pengaturan koneksi yang modern dan stabil. Jika terjadi kegagalan koneksi, kesalahan akan ditampilkan di konsol menggunakan *console.error()*, dan aplikasi akan berhenti dengan *process.exit(1)*. Fungsi *connectDB* kemudian diekspor untuk digunakan di file utama server.js, sehingga koneksi ke *database* dapat dimulai saat aplikasi dijalankan. Perintah untuk menjalankan aplikasi adalah *npm run dev* dan tampilan aplikasi ketika berhasil terhubung dengan database, ditampilkan seperti pada Gbr. 5.

```
Server running on port 8000
MongoDB connected successfully
```

Gbr. 5 Hasil Running Backend

3. Konfigurasi Express-Session

Konfigurasi express-session ditampilkan seperti pada Gbr.6

```
const session = require('express-session');
const MongoStore = require('connect-mongo');

const sessionMiddleware = session({
  secret: process.env.SESSION_SECRET,
  resave: false,
  saveUninitialized: false,
  store: MongoStore.create({ mongoUrl: process.env.MONGO_URL }),
  cookie: {
    secure: false,
    httpOnly: true,
    sameSite: 'Lax',
    maxAge: 3600000
  }
});

module.exports = sessionMiddleware;
```

Gbr. 6 Konfigurasi Express-Session

Konfigurasi ini mengatur *middleware* sesi menggunakan Express dan MongoDB untuk mengelola sesi pengguna dalam aplikasi Node.js. *Middleware* ini dimulai dengan mengimpor modul *express-session* untuk manajemen sesi dan *connect-mongo* untuk menyimpan sesi di MongoDB. Sesi disimpan di MongoDB menggunakan *MongoStore*, dengan URL MongoDB diambil dari variabel lingkungan *MONGO_URL*. Selain itu, konfigurasi *cookie* ditetapkan untuk menggunakan *secure: true* yang berarti cookie hanya dikirim melalui HTTPS. Opsi *httpOnly*: true memastikan bahwa *cookie* tidak dapat diakses oleh JavaScript pada sisi client, sehingga melindungi aplikasi dari serangan XSS (*Cross-Site Scripting*). Sementara itu, pengaturan sameSite: 'Lax' membatasi *cookie* hanya dikirimkan pada permintaan dari domain yang sama, meningkatkan perlindungan terhadap serangan CSRF *(Cross-Site Request Forgery)* dan *maxAge* diatur untuk menyimpan *cookie* selama periode waktu yang ditentukan. Setelah sesi berakhir, *cookie* akan kadaluarsa, dan pengguna akan kehilangan session mereka.

4. Perancangan Database Schema

Schema adalah struktur atau kerangka yang mendefinisikan bagaimana data diatur dan disimpan dalam sebuah *database*. *Schema* adalah *blueprint* yang menggambarkan bagaimana data akan diorganisasikan dan dikelola dalam suatu sistem *database*. Dalam pengembangan aplikasi ini, beberapa *schema models* digunakan untuk mengatur dan mengelola data dengan baik. Beberapa schema yang dirancang dengan attribute dan tipe data yang sudah ditentukan ditampilkan seperti pada tabel 2.

TABEL II
DATABASE SCHEMA

Nama Schema	Attribute	Tipe Data
User Schema	id	number

	name, email, password, role	string
Peserta Schema	fullName, gender, instansi, noTlp, alamat, suratPengantarMagang	string
	tanggalMulaiMagang, tanggalSelesaiMagang	date
Tugas Schema	namaTugas, deskripsi, file	string
Peserta Lolos Schema	fullName, instansi, divisi	string
Submit Tugas Schema	fullName, email, keterangan, fileTugas	string
Sertifikat Schema	fullName, instansi	string

5. Implementasi Controller

Controller bertanggung jawab mengelola operasi dasar (CRUD) pada data di aplikasi dan berfungsi sebagai penghubung antara permintaan dari klien dan interaksi dengan *database* melalui model. Implementasi *controller* pada website magang yang dirancang sebagai berikut.

A. UserController : *User controller* bertugas untuk menangani berbagai permintaan HTTP yang terkait dengan pengguna, seperti permintaan untuk register pengguna baru, *login*, memperbarui profil, dan menghapus akun.
B. PesertaController : Peserta *Controller* berfungsi untuk menangani permintaan HTTP yang berkaitan dengan data registrasi oleh peserta magang seperti registrasi peserta magang dan pengambilan data registrasi yang ada.
C. TugasController : Tugas *Controller* bertanggung jawab untuk menangani permintaan HTTP yang berkaitan dengan data tugas, seperti menambahkan tugas baru, mendapatkan daftar tugas, memperbarui tugas yang ada, dan menghapus tugas.
D. PesertaLolosController : PesertaLolos *Controller* bertanggung jawab untuk menangani permintaan HTTP yang berkaitan dengan data peserta yang lolos, termasuk CRUD seperti menambahkan peserta lolos baru, mengedit data peserta lolos, mendapatkan daftar peserta lolos dan menghapus data peserta lolos.
E. SubmitTugasController : SubmitTugas *Controller* bertanggung jawab untuk menangani permintaan HTTP yang berkaitan dengan pengumpulan tugas oleh peserta magang. Termasuk penerimaan data tugas yang dikirimkan oleh peserta, file tugas dan metadata terkait.
F. SertifikatController : Sertifikat *Controller* bertanggung jawab untuk menangani berbagai permintaan HTTP yang berkaitan dengan sertifikat, seperti permintaan pembuatan sertifikat oleh peserta magang dan data permintaan sertifikat.

6. Implementasi Middleware

Dalam perancangan aplikasi berbasis Node.js dan Express, *middleware* merupakan komponen penting yang mengatur berbagai aspek dari manajemen sesi dan otorisasi pengguna. Berikut adalah penjelasan tentang *middleware* utama yang digunakan dalam aplikasi

A. Middleware Auth

Middleware Auth digunakan untuk mengelola autentikasi dalam aplikasi web. Fungsinya adalah memastikan bahwa setiap permintaan yang diterima oleh aplikasi berasal dari pengguna yang telah terautentikasi. *Middleware* ini memeriksa *header Authorization* dari permintaan untuk menemukan token yang dikirim dalam format Bearer. Jika token tidak ada atau formatnya salah, *middleware* akan mengembalikan respons dengan status 401 *Unauthorized* dan pesan kesalahan yang sesuai. Jika terjadi kesalahan lain selama verifikasi, *middleware* akan mengembalikan respons kesalahan yang sesuai dan mencatat kesalahan tersebut di konsol.

B. Middleware checkSession

Middleware checkSession berfungsi untuk memverifikasi bahwa pengguna memiliki sesi yang valid sebelum dapat mengakses rute tertentu dalam aplikasi. Middleware akan memeriksa apakah req.session.userId ada, yang menandakan bahwa pengguna telah melakukan *login* dan memiliki sesi aktif. Jika sesi tidak ditemukan, *middleware* akan mengembalikan respons 401 *Unauthorized* dengan pesan bahwa otorisasi ditolak dan pengguna harus login terlebih dahulu. Jika sesi sudah berakhir, middleware akan mengembalikan respons 401 *Unauthorized* dengan pesan bahwa sesi telah berakhir dan pengguna harus melakukan *login* ulang. Dengan *middleware* ini, aplikasi dapat menjaga agar hanya pengguna dengan sesi yang valid yang dapat mengakses sumber daya atau fitur tertentu.

C. Middleware checkRole

Middleware checkRole berfungsi untuk memeriksa apakah pengguna yang telah terautentikasi memiliki peran yang sesuai sebelum mengakses rute tertentu. *Middleware* ini menerima parameter *roles*, yaitu array yang berisi peran-peran yang diizinkan. Pertama, *middleware* memeriksa apakah sesi pengguna aktif dan apakah peran pengguna tersedia dalam *req.session.role*. Jika sesi tidak ada, *middleware* akan mengembalikan respons dengan status 401 *Unauthorized* dan pesan yang menyatakan bahwa pengguna harus *login* terlebih dahulu. Selanjutnya, jika peran pengguna tidak termasuk dalam daftar peran yang diizinkan, *middleware* akan mengembalikan status 403 *Forbidden* dengan pesan bahwa akses ditolak. Jika semua pemeriksaan berhasil, *middleware* memanggil fungsi next() untuk melanjutkan ke *middleware* atau rute berikutnya. Dengan demikian, *middleware checkRole* berfungsi sebagai pengaman yang memastikan hanya pengguna yang memiliki hak akses sesuai yang dapat mengakses sumber daya tertentu dalam aplikasi.

D. Middleware Uploader

Middleware Uploader bertanggung jawab untuk menangani proses upload file dalam aplikasi. *Middleware* ini

menggunakan modul multer untuk memproses file yang diunggah dari permintaan pengguna dan menyimpannya ke penyimpanan *cloud* seperti ImageKit. *Uploader* juga dapat menangani validasi file, seperti memeriksa jenis file dan ukuran file untuk memastikan bahwa file yang diunggah memenuhi kriteria yang ditetapkan.

7. Perancangan Router

Router adalah komponen penting dalam pengembangan aplikasi web yang bertanggung jawab untuk menangani routing, yaitu proses menentukan rute berdasarkan URL dan metode HTTP yang diterima dari klien. Setiap rute didefinisikan untuk mengarahkan permintaan ke fungsi tertentu di controller, yang bertugas memproses logika dan memberikan respons yang relevan kepada pengguna. Dalam perancangan router, middleware yang telah ditentukan sebelumnya juga diterapkan untuk memastikan pengelolaan permintaan dilakukan dengan aman dan sesuai aturan. Dengan menggunakan middleware, router menjadi lebih terstruktur dan aman, karena middleware dapat menyaring permintaan yang tidak sesuai sebelum diteruskan ke fungsi Controller. Hal ini tidak hanya meningkatkan keamanan aplikasi tetapi juga memudahkan proses debugging dan pemeliharaan kode.

8. Session Management di Backend

Saat pengguna berhasil *login*, mereka akan menerima *JSON Web Token (JWT)* dan *Session ID* dalam bentuk string. Proses *login* melibatkan pencocokan kredensial yang dimasukkan pengguna dengan data yang tersimpan dalam *database. Session management* akan diterapkan pada fungsi *loginUser*. Fungsi *loginUser* bertanggung jawab untuk mengautentikasi pengguna saat *login* dengan memeriksa kredensial mereka dan mengatur sesi serta token untuk akses lebih lanjut. Fungsi login user ditampilkan seperti pada Gbr. 7.

```
const loginUser = async (req, res, next) => {
  const { email, password } = req.body;

  try {
    let user = await User.findOne({ email });
    if (!user) {
      return res.status(400).json({
        msg: 'Invalid Email or Password' });
    }
    const isMatch = await bcrypt.compare(password, user.password);
    if (!isMatch) {
      return res.status(400).json({
        msg: 'Invalid credentials' });
    }
    const payload = {
      _id: user._id,
      role: user.role
    };
    const token = jwt.sign(payload, JWT_SECRET_KEY, { expiresIn:
'1h' });
    // Set session
    req.session.userId = user._id; // Sesuaikan dengan ID user dari
MongoDB
    req.session.role = user.role; // Simpan role user

    return res.status(200).json({
      status: true,
      message: "Login successful",
      data: {
        user: {
          email: user.email,
          role: user.role
        },
        token,
        sessionId: req.session.id
      },
    });

  } catch (err) {
    console.error(err.message);
    res.status(500).send('Server error');
  }
};
```

Gbr. 7 Fungsi Login User

Ketika pengguna memulai sesi, sessionId dihasilkan dan disimpan bersama dengan data sesi lainnya di koleksi MongoDB melalui MongoStore, yang bekerja dengan *express-session*. Server kemudian menggunakan sessionId ini untuk mengambil data sesi terkait dari MongoDB, memverifikasi validitas sesi, dan mengakses informasi pengguna yang relevan. Selanjutnya adalah fungsi *logoutUser* yang bertanggung jawab untuk menangani proses logout pengguna dan penghancuran sesi yang masi aktif. Fungsi logout user ditampilkan seperti paga Gbr. 8.

```
const logoutUser = async (req, res) => {
  try {
    await new Promise((resolve, reject) => {
      req.session.destroy((err) => {
        if (err) {
          reject(err);
        } else {
          resolve();
        }
      });
    });
    console.log('Session destroyed');
    res.clearCookie('connect.sid', { path: '/' });
    console.log('Cookie cleared');
    res.status(200).json({ msg: 'Logout berhasil' });
  } catch (err) {
    console.error('Error during logout:', err.message);
    res.status(500).json({ msg: 'Server error' });
  }
};
```

Gbr. 8 Fungsi Logout User

Manajemen sesi yang diimplementasikan pada fungsi *login* dan *logout*, melibatkan pembuatan token untuk autentikasi dan otorisasi serta pengelolaan *session ID* di server. Pada proses *login*, setelah pengguna berhasil memverifikasi kredensial berupa email dan *password*. Aplikasi akan membuat token JWT dan *sessionId*. Token dan *SessionId* ini digunakan untuk autentikasi dan otorisasi pengguna dalam setiap permintaan berikutnya. *Session management* juga mencakup pengelolaan masa berlaku sesi. Token memiliki masa berlaku yang terbatas sesuai dengan waku yang sudah ditentukan. Pada server, sesi disimpan dan dikelola dalam *session-store*, yang memastikan sesi tetap valid selama masa berlakunya. Saat

pengguna melakukan logout, token atau session ID dihancurkan. Sesi akan dihapus dari penyimpanan server, sedangkan token dihapus dari klien dan server tidak lagi menerima token tersebut. Dengan demikian, *logout* memastikan bahwa sesi pengguna tidak lagi aktif dan akses ke aplikasi ditolak sampai pengguna melakukan login kembali.

B. Perancangan Frontend

1. Inisialisasi dan Instalasi Dependensi

Inisialisasi React dengan Vite.js adalah langkah awal untuk membangun antarmuka pengguna dengan efisien dan cepat. Proses inisialisasi dimulai dengan perintah npm create vite@latest. Setelah inisialisasi dilakukan seperti perintah pada kode 4.31, selanjutnya adalah instalasi dependensi yang digunkan untuk merancang frontend aplikasi. Dependensi frontend dan versinya ditampilkan seperti pada Gbr. 9.

```
"dependencies": {
  "axios": "^1.7.4",
  "bootstrap": "^5.3.3",
  "chart.js": "^4.4.4",
  "datatables.net-dt": "^2.1.4",
  "jquery": "^3.7.1",
  "jwt-decode": "^4.0.0",
  "leaflet": "^1.9.4",
  "react": "^18.3.1",
  "react-bootstrap": "^2.10.4",
  "react-calendar": "^5.0.0",
  "react-chartjs-2": "^5.2.0",
  "react-data-table-component": "^7.6.2",
  "react-dom": "^18.3.1",
  "react-icons": "^5.3.0",
  "react-leaflet": "^4.2.1",
  "react-redux": "^9.1.2",
  "react-router-dom": "^6.26.1",
  "redux": "^5.0.1"
},
```

Gbr. 9 Dependensi Frontend

Selanjutnya adalah proses inisialisasi css sebagai bagian dari *styling frontend*. Inisialisasi termasuk bootstrap versi @5.3 yang digunakan sebagai *framework* CSS untuk mempercepat pengembangan antarmuka pengguna dengan menyediakan berbagai komponen dan utilitas *styling*, import file CSS untuk *Bootstrap Icons* versi 1.5.0 dari CDN dan import file CSS untuk *DataTables* versi 1.13.4 dari CDN. Aplikasi dapat dijalankan menggunakan peritah npm run dev pada terminal. Hasil *running frontend* seperti pada Gbr. 10.

```
VITE v5.3.5  ready in 4345 ms

➜  Local:   http://localhost:5173/
➜  Network: use --host to expose
➜  press h + enter to show help
```

Gbr. 10 Hasil Running Frontend

2. Session Management di Frontend

A. Autentikasi

Autentikasi pada *frontend* atau *client side* menggunakan react dan axios diimplementasikan pada *handleSubmit* form *login* dan ditampilkan seperti pada Gbr. 11.

```
const handleSubmit = async (event) => {
  event.preventDefault();
  try {
    setIsLoading(true); // Start loading
    const response = await axios.post(
      'http://localhost:8000/api/v1/users/login',
      {
        email,
        password
      },
      { withCredentials: true },
    );
    localStorage.setItem("token", response.data.data.token);
    const user = response.data.data.user;
    const role = user.role;
    const sessionId = response.data.data.sessionId;

    if (role === 'Admin') {
      navigateTo('/dashboard');
    } else if (role === 'User') {
      navigateTo('/homepage');
    }
  } catch (error) {
    console.error('Login error:', error.response ? error.response.data :
error.message); // Debug log
    setError('Login failed. Please check your credentials.');
  }
  setIsLoading(false); // Stop loading
};
```

Gbr. 11 Autentikasi di Frontend

Perintah axios.post akan mengirim permintaan **POST** ke URL *login*. Perintah ini mengirimkan data *login* berupa email dan *password*. Perintah *withCredentials: true:* memastikan *cookies* dikirim bersama dengan permintaan. Apabila terdapat kesamaan dengan data dalam database, proses login berhasil dan local storage akan menyimpan token yang sudah degenerate oleh server dan session Id pada cookie untuk memeriksa role pengguna sebelum dinavigasikan ke halaman yang sesuai.

B. Otorisasi

Otorisasi pada *client side* menggunakan fungsi *checkAcces* ditampilkan seperti pada Gbr. 12.

```
useEffect(() => {
  async function checkAccess() {
    try {
      const response = await fetch('http://localhost:8000/api/check-access',
{
        method: 'GET',
        credentials: 'include'
      });

      if (response.ok) {
        const data = await response.json();
        const userRole = data.role;

        if (userRole === 'Admin') {
          setIsAuthorized(true);
        } else {
          console.log('Its not user role');
          navigate('/');
        }
```

```
        } else {
          console.log('Unauthorized access');
          navigateTo('/login');
        }
      } catch (error) {
        console.error('Error fetching access data:', error);
        navigateTo('/login');
      }
      setLoading(false);
    }
    checkAccess();
}, [navigateTo]);
```

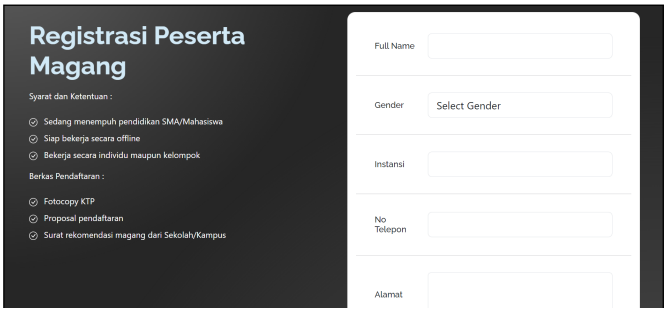
Gbr. 12 Fungsi checkAcces

Fungsi *checkAccess* bertujuan untuk memastikan bahwa pengguna aplikasi memiliki hak akses yang sesuai sebelum melanjutkan ke bagian tertentu dalam aplikasi. Fungsi ini melakukan verifikasi otentikasi dengan mengirimkan permintaan ke *endpoint check-access*, dan jika pengguna tidak memiliki sesi yang valid, mereka akan diarahkan ke halaman *login*. Setelah memastikan bahwa pengguna terautentikasi, fungsi ini memeriksa peran pengguna yang diterima dari server. Fungsi ini akan diimplementasikan pada setiap halaman untuk memastikan bahwa hanya pengguna yang sudah terautentikasi dan juga memiliki hak akses yang sesuai yang dapat mengakses *page* atau halaman tertentu.

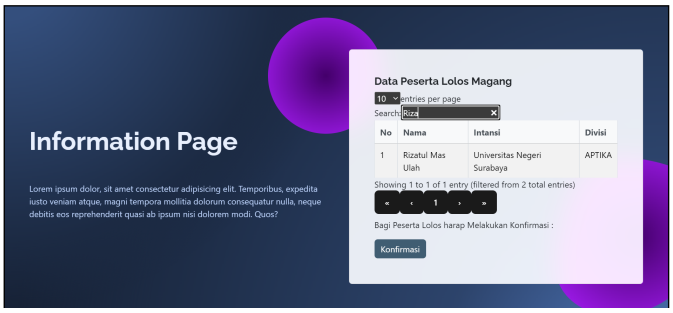
3. Hasil.

Hasil tampilan antarmuka pengguna yang dirancang menggunakan *React* dan *Bootstrap* ditampilkan seperti pada Gbr. 13-22.

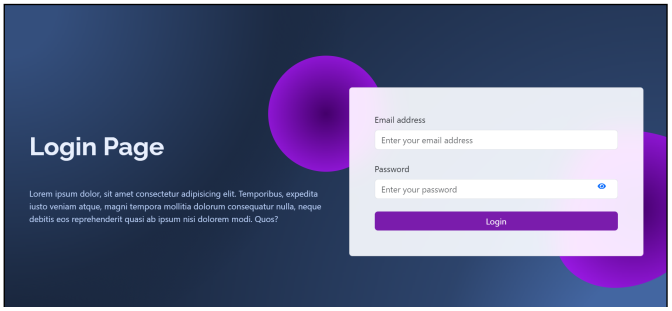
Gbr. 13 Halaman Landing



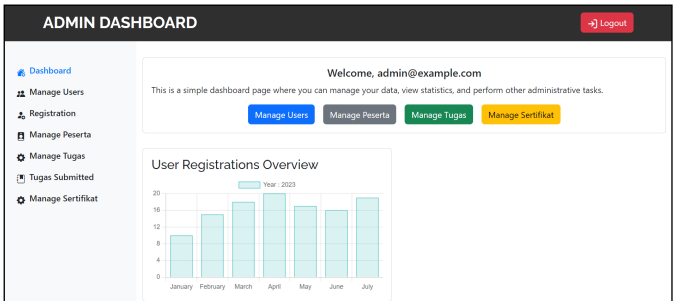
Gbr. 14 Halaman Registrasi



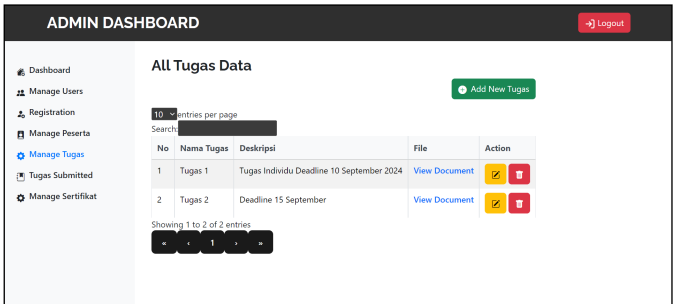
Gbr.15 Halaman Peserta Lolos



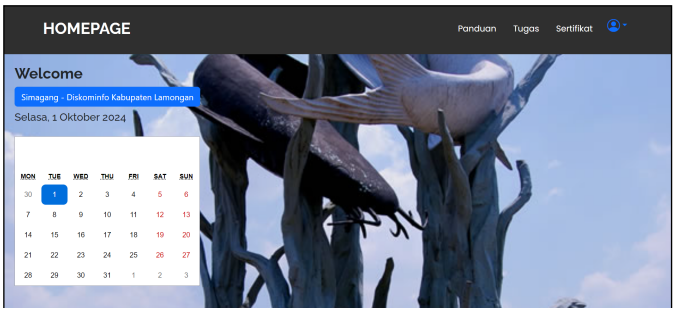
Gbr. 16 Halaman Login



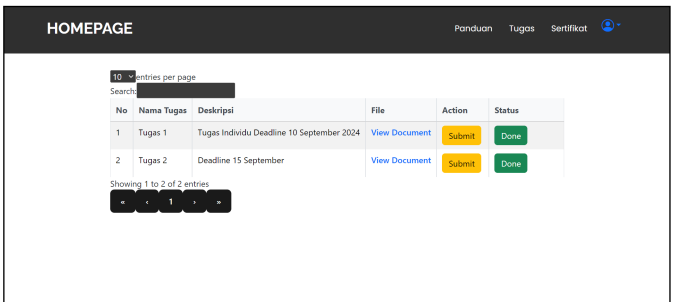
Gbr. 17 Halaman Dashboard Admin



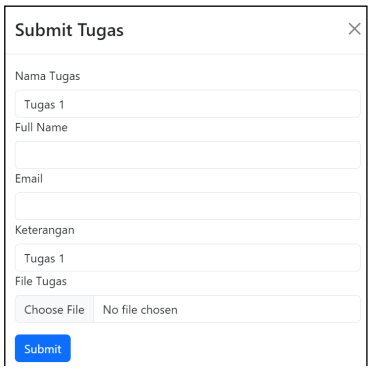
Gbr. 18 Halaman Tugas



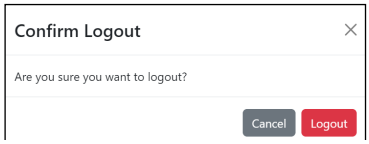
Gbr. 19 Homepage Peserta



Gbr. 20 Halaman Tugas Peserta



Gbr. 21 Form Submit Tugas



Gbr. 22 Modal Konfirmasi Logout

Antarmuka pengguna aplikasi dikembangkan dengan menggunakan *React,* sebuah pustaka *JavaScript* yang memungkinkan pembuatan komponen UI yang dinamis dan responsif. Untuk mendukung tata letak yang terstruktur dan tampilan yang modern, *Bootstrap* diintegrasikan ke dalam proyek. Integrasi *React* dan *Bootstrap* dalam antarmuka pengguna memastikan bahwa aplikasi tidak hanya berfungsi dengan baik tetapi juga menawarkan pengalaman pengguna yang intuitif dan menyenangkan.

C. Pengujian Sistem

1. Blackbox Testing

Blackbox testing dilakukan dengan menguji fungsionalitas sistem berdasarkan *input* dan *output* tanpa memeriksa struktur atau kode internal. Pengujian hanya fokus pada spesifikasi dan persyaratan sistem, mengamati apakah hasil yang dihasilkan sesuai dengan harapan. Hasil pengujian dengan blackbox testing ditampilkan pada tabel 3.

TABEL III
HASIL BLACKBOX TESTING

No	Fitur Yang Diuji	*Tes Case*	Hasil Yang Diharapkan	Hasil pengujian
1	*Login*	*Login* dengan email dan *password* valid	*Login* berhasil, diarahkan menuju halaman *dashboard/homepage*	*Login* berhasil, diarahkan menuju halaman *dashboard /homepage* (Berhasil)
		Login dengan email dan *password* invalid	*Login* gagal, muncul pesan *error* dan tetap berada di halaman *login*	*Login* gagal, muncul pesan *error* dan tetap berada di halaman *login* (Berhasil)
2	*Role*	*Login* sebagai admin	*Role* admin akan diarahkan ke halaman *dashboard*	Role admin akan diarahkan ke halaman *dashboard* (Berhasil)
		Login sebagai user	*Role user* akan diarahkan ke halaman *homepage*	*Role user* akan diarahkan ke halaman *homepage* (Berhasil)
3	Registrasi Peserta Magang	Mengisi setiap field form dengan data valid	Berhasil melakukan registrasi dan diarahkan ke halaman informasi penerimaan	Berhasil melakukan registrasi dan diarahkan ke halaman informasi penerimaan (Berhasil)
		Mengosongkan field tertentu	Pesan *error* validasi muncul, memberitahukan bahwa semua field wajib diisi.	Pesan *error* validasi muncul, memberitahukan bahwa semua field wajib

				diisi (Berhasil)
4	Upload file	*Upload* file dengan format jpg/pdf	File berhasil diupload ke imagekit	File berhasil diupload ke imagekit (Berhasil)
		Upload dengan format selain jpg/pdf	Pesan *error* muncul yang menyatakan format file tidak diizinkan	Pesan *error* muncul yang menyatakan format file tidak diizinkan (Berhasil)
5	*Create* data tugas	*Login* sebagai admin	Berhasil mengakses halaman tugas dan melakukan operasi CRUD Tugas	Berhasil mengakses halaman tugas dan melakukan operasi CRUD Tugas (Berhasil)
		Login sebagai *user*	Tidak dapat mengakses halaman tugas dan dikembalikan ke halaman *login*	Tidak dapat mengakses halaman tugas dan dikembalikan ke halaman *login* (Berhasil)
6	Register *New User*	Mengisi data register dengan email baru	Sukses *create new user*	Sukses *create new user* (Berhasil)
		Mengisi data register dengan email yang sudah ada	Muncul pesan *error "User allready exist"*	Muncul pesan *error "User allready exist"* (Berhasil)
7	Delete User	Menghapus data user dengan *role* admin	Muncul *allert "Cant delette admin user".* Data tidak dapat dihapus	*Muncul allert "Cant delette admin user".* Data tidak dapat dihapus (Berhasil)
		Menghapus data *user* dengan *role user*	Muncul modal sukses, data user terhapus dari database	Muncul modal sukses, data user terhapus dari database (Berhasil)
8	*Logout*	*Logout* dari sistem	Muncul modal konfirmasi *logout* dan diarahkan ke halaman *landingpage*	Muncul modal konfirmasi *logout* dan diarahkan ke halaman *landingpage* (Berhasil)

Pengujian ini menunjukkan bahwa sistem mampu menangani input yang valid dan memberikan output yang sesuai, sekaligus menangani kesalahan input dengan tepat. Fungsi utama aplikasi berjalan sesuai spesifikasi dan kebutuhan pengguna.

2. Pengujian Tingkat Keamanan Dengan Session Management Testing

Session management testing dilakukan untuk menguji penanganan autentikasi, otorisasi, dan pengelolaan sesi yang telah diimplementasikan dalam aplikasi. Pengujian manajemen sesi dilakukan menggunakan aplikasi postman dan browser dengan hasil pengujian ditampilkan seperti pada tabel 4.

TABEL IV
HASIL SESSION MANAGEMENT TESTING

Session Management Testing	***Test Name***	***Test Case***	***Test Result & Status***
WSTG-AUTH-01	*User credential*	Inputkan username dan *passoword* valid Inputkan username dan *password* invalid	Muncul pesan *error "invalid credential".* Proses login gagal. (Berhasil) Proses *login* berhasil, *Generate* token jwt di sisi klien dan *session store* di sisi server. (Berhasil)
WSTG-SEC-01	*Security control*	Periksa HTTP Status untuk proses login berhasil Periksa HTTP status untuk proses login gagal	Status 200 Ok, Berhasil login ke dalam aplikasi. (Berhasil) Status 400, bad request. Dengan pesan error "Invalid credentials". (Berhasil)
WSTG-SESS-01	*Session management testing*	Akses URL tertentu tanpa token atau *session ID*	Muncul pesan error *"Authorization denied. Please login first.".* Permintaan

		Menggunakan token dengan format salah Menggunakan token dengan status kadaluarsa Menggunakan token dan *session ID* yang valid untuk mengakses URL tertentu Menggunakan token dan *session ID* valid, namun tidak sesuai *role*	ditolak. (Berhasil) Muncul pesan error "Format token salah". Permintaan ditolak. (Berhasil) Muncul pesan error "Token tidak valid". Permintaan ditolak. (Berhasil) Berhasil akses url dan memunculkan data yang diinginkan. (Berhasil) Muncul pesan error "*Access denied. You are not authorized to access this resource.* ". Permintaan ditolak (Berhasil)
WSTG-SESS-02	*Testing for logout functionality*	Menganalisis *cookie* sesi dan memastikan sesi benar-benar dihentikan setelah *logout*	Fungsi *logout* berhasil menghapus *cookie connect sid* di sisi klien dan menghapus *session store* di sisi server. (Berhasil)
WSTG-SESS-03	*Testing session timeout*	Menguji cookie dan token yang kadaluarsa untuk akses URL tertentu	Muncul pesan error dengan status 401 *unauthorized* apabila akses url menggunakan token dan sesi kadaluarsa. Permintaan ditolak. (Berhasil)
WSTG-API-01	*API Authentication Testing*	Mengirim permintaan ke endpoint API tanpa token Mengirim permintaan ke endpoint API dengan format token salah Mengirim permintaan ke endpoint API dengan format token kadaluarsa	Server menanggapi dengan status code 401 Unauthorized. Permintaan ditolak. (Berhasil)
WSTG-API-02	*API Authorization Testing*	Mengakses endpoint API dengan session dan role sesuai Mengakses endpoint API dengan session dan role tidak sesuai	Status 200 ok berhasil Muncul pesan error *"Access denied. You are not authorized to access this resource.* " status code 403 forbidden yang berarti akses dilarang karena role yang tersimpan dalam cookie tidak sesuai. (Berhasil)

Hasil pengujian menunjukkan bahwa *session management* dalam aplikasi telah berfungsi dengan sangat baik dan sesuai dengan skenario yang diharapkan, memastikan keamanan dan kontrol akses yang efektif bagi pengguna.

IV. KESIMPULAN

Penelitian ini berhasil menerapkan *authentication, authorization,* dan *express-session* sebagai bagian dari *session management* pada website magang Dinas Komunikasi dan Informatika Kabupaten Lamongan yang dirancang menggunakan teknologi MERN (MongoDB, Express.js, React, dan Node.js). Sistem berhasil mengintegrasikan autentikasi berbasis JWT (JSON Web Token) dan manajemen sesi menggunakan *cookie* serta *session ID.* Sistem meningkatkan keamanan dengan memastikan bahwa akses hanya diberikan kepada pengguna yang telah terautentikasi dan memiliki hak akses yang sesuai. *Cookie* diatur dengan opsi *httpOnly* dan *secure: true* untuk memastikan bahwa *cookie* hanya dapat

diakses melalui protokol HTTPS dan tidak dapat diakses atau dimanipulasi oleh JavaScript di sisi klien. Hal ini dapat membantu mencegah berbagai upaya serangan seperti *Cross-Site Scripting* (XSS) dan *Man-in-the-Middle* (MITM), yang dapat digunakan oleh penyerang untuk mencuri atau memanipulasi *session cookie*.

Pengujian dilakukan menggunakan metode *blackbox testing* dan *session management testing*. Hasil pengujian menunjukkan bahwa sistem berfungsi sesuai dengan skenario yang diharapkan. Sistem berhasil menghasilkan token dan *cookie session ID* yang valid saat pengguna melakukan *login.* Token dan *cookie* disimpan dengan benar di sisi klien, dan informasi ID pengguna serta peran dapat digunakan untuk mengakses halaman yang sesuai. Akses ke URL tertentu tanpa login memunculkan respons error 401 *Unauthorized,* yang memastikan bahwa akses tanpa sesi yang valid tidak diperbolehkan. Token yang salah format atau sudah kadaluarsa juga menghasilkan error 401 *Unauthorized*, menunjukkan bahwa validasi token berfungsi dengan baik. Proses *logout* berhasil menghapus sesi di server dan *cookie* di sisi klien, serta akses dengan session ID yang sudah dihapus menghasilkan error 401 *Unauthorized,* menegaskan bahwa sesi tersebut telah terhapus. Secara keseluruhan, hasil pengujian menunjukkan bahwa sistem yang dirancang berfungsi sesuai dengan skenario yang diharapkan, memberikan kontrol akses yang tepat, serta menjaga keamanan dan integritas data pengguna.

REFERENSI

[1] Yunita Trimarsiah & Muhajir Arafat, "Analisis dan Perancangan Website sebagai Sarana Informasi pada Lembaga Bahasa Kewirausahaan dan Komputer AKMI Baturaja,".

[2] Syabdan Dalimunthe, Emansa Hasri Putra, & Muhammad Arif Fadhly Ridha, "RESTful API Security Using JSON Web Token (JWT) with HMAC-SHA512 Algorithm in Session Management,", IT Journal Research and Development (ITJRD), vol. 8, no. 1, August 2023, doi: 10.25299/itjrd.2023.1202981.

[3] Kementerian Kesehatan Republik Indonesia, "Standar keamanan aplikasi berbasis website," 2024. [Online]. Available: https://csirt.kemkes.go.id/detail/13-Standar-Keamanan-Aplikasi-Berbasis-Website

[4] R. Moch Makruf Puja Pradana, Mahendra Data, & Dany Primanita Kartikasari, "Implementasi Shared Session dalam Klaster Server Web Menggunakan PHP dan MySQL", Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer, vol. 3, no. 5, pp. 4856-4864, Mei 2019. [Online]. Available: http://j-ptiik.ub.ac.id.

[5] Nadya Hanifah Kamal & Aris Gunaryati, "Implementasi Pengembangan Web Menggunakan Teknologi MERN Stack pada Sistem Informasi Akademik Siswa Berbasis Web", Jurnal Sistem dan Teknologi Informasi, vol. 11, no. 3, pp. 458-465, Juli 2023, doi: 10.26418/justin.v11i3.53350.

[6] [6] M. Arif Rahman, "Pengembangan Sistem Otentikasi pada Sebuah Aplikasi yang Berbasiskan Web", Syntax: Journal of Software Engineering, Computer Science and Information Technology, vol. 1, no. 1, pp. 1-10, Tahun 2020.

[7] I. Gusti Ngurah Ady Kusuma, "Aplikasi Pencatatan Inventori Berbasis Website dengan Skema Autentikasi dan Otorisasi Stateless Sederhana" , Journal of Innovation Research and Knowledge, vol. 1, no. 9, pp. 1111-1120, Februari 2022, ISSN 2798-3471 (Cetak), ISSN 2798-3641 (Online).

[8] Md Maruf Hassan, "Broken Authentication and Session Management Vulnerability: A Case Study of Web Application", International Journal of Simulation: Systems, Science & Technology, vol. 19, no. 2, April 2018, doi: 10.5013/IJSSST.a.19.02.06.

[9] Clinton Hatta Pradigi, Tintin Harlina, & Solehatin, "Implementation of Express JS to Build REST API Website STIKOM PGRI Banyuwangi", Jurnal Informatika dan Komputer, ISSN: 2597-372X (Online).

[10] U. B. Astowo & A. Sujarwo, "Penerapan JSON Web Token sebagai strategi pengamanan data pada aplikasi MultiMasjid", Innovative: Journal of Social Science Research, vol. 3, no. 6, pp. 5279-5292, 2023, E-ISSN 2807-4238, P-ISSN 2807-4246.

[11] B. Adam, "Audit system session management testing siakad," Bisa Ai, [Online]. Available: https://bisa.ai/portofolio/detail/NDc#:~:text=Session%20Management%20Schema%20adalah%20dimana,lainnya%20yang%20tidak%20memiliki%20sesi.

[12] Soetam Rizky Wicaksono, "Blackbox Testing Teori dan Studi Kasus," Seribu Bintang, January 2022, ISBN: 978-623-7000-40-2, doi: 10.5281/zenodo.7659674.

[13] A. I. Rafeli, H. B. Seta, & I. W. Widi, "Pengujian celah keamanan menggunakan metode OWASP Web Security Testing Guide (WSTG) pada website XYZ", Jurnal Informatik, vol. 18, no. 2, p. 97, 2022.

[14] Aditya Wibisono Kuncoro, "Analisis Metode Open Web Application Security Project (OWASP) pada Pengujian Keamanan Website: Literature Review."