Pengembangan *Chat* Bot Telegram untuk Admisi UKDW

Andersen[1], Willy Sudiarto Raharjo[2], Matahari Bhakti Nendya[3]
Informatika,Universitas Kristen Duta Wacana
Jl. Dr. Wahidin Sudirohusodo No.5-25, Yogyakarta
[1]andersen@ti.ukdw.ac.id
[2]didanendya@staff.ukdw.ac.id
[3]willysr@ti.ukdw.ac.id

Abstract*— The Admissions and Promotion Office of Universitas Kristen Duta Wacana (UKDW) Yogyakarta serves as a service unit for the admission and registration of prospective students. Currently, information related to new student admissions is provided through physical services, direct contact, and social media. However, using social media requires human resources and time to respond to every inquiry from prospective students. To improve responsiveness, an automated system in the form of a Telegram Bot has been developed. This Telegram Bot is designed to provide fast and automated new student admission information services. Through the integration of AWS services, such as Lambda and DynamoDB, the bot can process user inputs in real-time and deliver relevant answers without manual intervention. This system is expected to support the Admissions Office in providing modern services to prospective students.

Intisari— Kantor Unit Admisi dan Promosi Universitas Kristen Duta Wacana (UKDW) Yogyakarta berperan sebagai unit pelayanan penerimaan dan pendaftaran calon mahasiswa baru. Saat ini, informasi terkait penerimaan mahasiswa baru diberikan melalui layanan fisik, kontak langsung, maupun media sosial. Namun, penggunaan media sosial memerlukan sumber daya manusia dan waktu untuk merespons setiap pertanyaan dari calon mahasiswa. Untuk meningkatkan responsivitas, dikembangkan sebuah sistem otomasi dalam bentuk Bot Telegram. Bot Telegram ini dirancang untuk menyediakan layanan informasi penerimaan mahasiswa baru secara cepat dan otomatis. Dengan integrasi layanan AWS, seperti Lambda dan DynamoDB, bot ini mampu memproses input pengguna secara real-time dan memberikan jawaban yang relevan tanpa intervensi manual. Sistem ini diharapkan dapat mendukung Unit Admisi dalam memberikan layanan yang modern kepada calon mahasiswa.

***Kata Kunci*—** *Telegram Bot* , *AWS* , *DynamoDB*

I. PENDAHULUAN

Kantor Unit Admisi dan Promosi merupakan unit pelayanan pihak Universitas Kristen Duta Wacana (UKDW) Yogyakarta yang melayani penerimaan maupun pendaftaran bagi calon mahasiswa. Saat ini apabila seorang calon mahasiswa ingin mendapatkan informasi terkait penerimaan mahasiswa baru, calon mahasiswa dapat memperoleh informasi tersebut melalui Unit Admisi secara fisik ataupun melalui kontak ataupun media sosial yang dimiliki.

Penggunaan media sosial saat ini menggunakan *resource* dalam bentuk sumber daya manusia dan membutuhkan waktu untuk merespon pada tiap calon mahasiswa yang ingin bertanya. Oleh sebab itu penulis ingin membuat sebuah sistem yang dapat otomisasi proses tersebut dengan membangun sebuah Bot Telegram.

Pembangunan bot Telegram untuk Unit Admisi Universitas Kristen Duta Wacana (UKDW) merupakan upaya untuk menyediakan layanan informasi penerimaan mahasiswa baru yang lebih responsif. Dengan adanya bot ini, calon mahasiswa dapat memperoleh jawaban atas pertanyaan terkait penerimaan secara cepat, tanpa harus menunggu respon manual. Proses pengembangan bot ini akan difokuskan pada integrasi dengan layanan AWS seperti Lambda dan DynamoDB untuk memproses input dari pengguna secara real-time.

A. Tinjauan Pustaka

Beberapa penelitian terdahulu menunjukkan relevansi yang signifikan terhadap pengembangan sistem chatbot berbasis layanan cloud computing, khususnya dalam konteks pelayanan informasi akademik.

A. Lubis dan I. Sumartono [1] dan Alfiansyah et al. [2] mengembangkan chatbot sebagai sarana layanan akademik di lingkungan perguruan tinggi guna mengatasi permasalahan umum seperti keterbatasan sumber daya manusia dan lambatnya respon terhadap pengguna. Dalam penelitian Lubis et al. [1], implementasi arsitektur sistem memanfaatkan beberapa layanan dari Amazon Web Services (AWS), termasuk AWS Lambda, API Gateway, dan DynamoDB. Studi ini relevan dengan penelitian yang dikembangkan karena menunjukkan bagaimana layanan cloud computing dapat diintegrasikan secara efisien dalam pengembangan sistem chatbot.

F. Alfaiz dan Maryam [3] merancang sistem berbasis Bot Telegram yang bertujuan untuk mempercepat proses registrasi pengguna. Pengujian sistem dilakukan menggunakan metode blackbox untuk mengevaluasi fungsionalitas komponen perangkat lunak. Pendekatan ini memberikan dasar yang kuat dalam pemilihan metode pengujian pada penelitian ini, yang juga menerapkan blackbox testing sebagai metode validasi sistem.

Studi yang dilakukan oleh Lestari et al. [4] berfokus pada pemanfaatan chatbot dalam meningkatkan partisipasi masyarakat melalui penyediaan informasi seperti jadwal kegiatan dan notifikasi penting. Hasil penelitian tersebut menunjukkan bahwa chatbot dapat menjadi media penyampaian informasi yang efisien, terutama dalam

konteks layanan publik yang menuntut akses cepat dan mudah.

Selanjutnya, Lenardo et al. [5] mengembangkan Bot Telegram sebagai alternatif dari sistem informasi berbasis web untuk mempermudah akses siswa terhadap data akademik. Pendekatan tersebut sejalan dengan tujuan penelitian ini, yaitu membangun sistem bot sebagai solusi penyampaian informasi akademik secara cepat, praktis, dan terintegrasi.

Penelitian yang dilakukan oleh M. Qalimaturrrahmah dan D. Santoso [6] mengidentifikasi permasalahan yang umum ditemui di lingkungan pendidikan tinggi, yaitu rendahnya tingkat literasi mahasiswa terhadap informasi penting yang telah tersedia di situs web fakultas. Hal ini disebabkan oleh kurangnya kebiasaan mahasiswa dalam mengakses secara rutin situs web resmi institusi. Untuk mengatasi permasalahan tersebut, mereka mengembangkan sebuah chatbot berbasis platform Telegram yang dirancang untuk menyampaikan informasi akademik secara lebih cepat dan efisien. Pendekatan ini relevan dengan penelitian ini, mengingat kedua penelitian memiliki tujuan yang sama, yaitu meningkatkan aksesibilitas informasi akademik dengan memanfaatkan teknologi chatbot sebagai sarana komunikasi langsung antara sistem dan pengguna.

Penelitian yang dilakukan oleh H. Mulyo dan G. Mohammad [7] menunjukkan keterkaitan yang erat dengan penelitian ini, khususnya dalam upaya meningkatkan efisiensi akses informasi akademik bagi mahasiswa. Sistem Informasi Akademik Mahasiswa (SIAMA) yang mereka kembangkan pada awalnya berbasis web, namun dinilai kurang efisien karena memerlukan beberapa tahapan akses serta bergantung pada konsumsi data internet yang cukup tinggi. Untuk mengatasi permasalahan tersebut, mereka mengintegrasikan layanan Bot Telegram sebagai sistem penjawab otomatis dengan memanfaatkan *library* PHP Telegram Bot dan *scheduler* Crontab Linux. Pendekatan ini memiliki kesamaan dengan penelitian ini, yang juga memanfaatkan Bot Telegram sebagai media untuk menyampaikan informasi akademik secara lebih cepat dan praktis. Perbedaan terletak pada penggunaan teknologi pendukung, di mana penelitian ini menggunakan layanan komputasi awan (cloud computing) dari AWS seperti Lambda dan DynamoDB, namun keduanya memiliki tujuan yang sejalan dalam meningkatkan aksesibilitas informasi secara efisien.

II. LANDASAN TEORI

A. Event-Driven Architecture

Event-Driven Architecture (EDA) merupakan pendekatan arsitektur sistem terdistribusi yang beroperasi secara asinkron, di mana suatu peristiwa atau kejadian tertentu dapat memicu pemrosesan atau tindakan khusus dalam sistem [8]. Arsitektur ini dibangun di atas sejumlah prinsip utama yang membedakannya dari pendekatan tradisional, salah satunya adalah *loose coupling*, yakni keterpisahan antar komponen sistem yang memungkinkan setiap bagian beroperasi secara independent [9]. Keterpisahan ini memberikan fleksibilitas tinggi terhadap proses pengembangan dan pemeliharaan, karena perubahan pada satu komponen tidak secara langsung memengaruhi komponen lainnya [9].

Pendekatan EDA memberikan kemampuan bagi sistem untuk mencapai skalabilitas, fleksibilitas, serta toleransi terhadap kesalahan, sehingga menjadikannya elemen penting dalam desain sistem modern, terutama pada sektor-sektor seperti keuangan, e-commerce, layanan kesehatan, dan Internet of Things (IoT) [10]. Dalam pola ini, sistem dirancang untuk merespons *event*, yang dapat berupa perubahan status, interaksi pengguna, atau sinyal yang dihasilkan oleh sistem. Setiap *event* dipandang sebagai kejadian penting yang dapat memicu aksi lanjutan atau perubahan pada komponen lain, dengan komunikasi antar komponen berlangsung secara asinkron [10].

Dalam konteks chatbot informasi penerimaan mahasiswa baru yang dikembangkan, pendekatan EDA sangat relevan karena mendukung pemrosesan input pengguna sebagai *event* yang secara langsung memicu pemanggilan layanan terkait melalui fungsi-fungsi seperti AWS Lambda, sehingga memastikan interaksi berlangsung cepat, responsif, dan terpisah secara modular.

B. Serverless Architecture

Komputasi serverless merupakan paradigma baru dalam layanan cloud yang memungkinkan pengguna menjalankan aplikasi tanpa perlu menyediakan atau mengelola infrastruktur server secara langsung [11]. Berbeda dengan model tradisional, dalam arsitektur serverless, kode dijalankan sebagai respons terhadap *event* tertentu, seperti permintaan HTTP atau penjadwalan tugas (*cron job*), dan pengguna hanya dikenakan biaya berdasarkan jumlah sumber daya komputasi yang benar-benar digunakan. Seluruh pengelolaan alokasi sumber daya serta penempatan tugas menjadi tanggung jawab penyedia layanan cloud [11].

Model ini memiliki sejumlah karakteristik utama yang menjadikannya solusi unggul dalam pengembangan sistem modern. Pertama, serverless mendukung *on-demand self-service*, yaitu kemampuan pengguna untuk secara langsung menyediakan sumber daya komputasi seperti memori dan kapasitas pemrosesan melalui antarmuka web tanpa interaksi manual dengan penyedia layanan . Kedua, adanya *broad network access* memungkinkan layanan diakses dari berbagai perangkat dan lokasi melalui koneksi internet. Ketiga, serverless memanfaatkan *resource pooling*, yakni penyatuan sumber daya seperti komputasi, penyimpanan, dan bandwidth yang dibagikan secara efisien antar pengguna, namun tetap terisolasi menggunakan teknologi virtualisasi. Keempat, model ini mendukung *rapid elasticity*, yaitu kemampuan untuk menyesuaikan penggunaan sumber daya secara otomatis sesuai dengan perubahan beban kerja. Terakhir, serverless menerapkan *measured service*, di mana penggunaan sumber daya dipantau dan dicatat secara akurat untuk mendukung sistem penagihan yang transparan berdasarkan konsumsi aktual [11].

Salah satu implementasi populer dari model ini adalah AWS Lambda, yang memungkinkan eksekusi kode sebagai respons terhadap event tanpa memerlukan pengelolaan server secara manual. Layanan ini sangat selaras dengan prinsip Event-Driven Architecture (EDA), di mana setiap event dapat langsung memicu pemrosesan melalui fungsi

JUTEI Edisi Volume. 9 No.1 April 2025
ISSN 2579-3675, e-ISSN 2579-5538
DOI 10.21460/jutei.2025.91.397

Lambda, sehingga mendukung sistem yang responsif, fleksibel, dan mudah diskalakan secara otomatis.

C. NoSQL

Selama lebih dari satu dekade terakhir, basis data NoSQL telah menunjukkan pertumbuhan yang signifikan sebagai alternatif dari sistem basis data relasional (SQL) tradisional. Perbedaan utama antara keduanya terletak pada fleksibilitas struktur data. Basis data SQL mengharuskan skema yang kaku dan harus ditentukan sejak awal, sehingga menyulitkan proses modifikasi data dan membatasi kemampuan untuk penskalaan horizontal. Sebaliknya, NoSQL menawarkan struktur tanpa skema (schema-less) yang memungkinkan representasi data berkembang secara dinamis sesuai kebutuhan. Selain itu, NoSQL dirancang untuk mendukung penskalaan horizontal melalui mekanisme replikasi dan sharding data pada klaster besar, sehingga lebih adaptif terhadap aplikasi modern yang menuntut fleksibilitas serta performa tinggi [12].

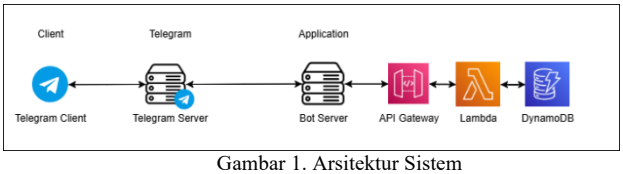
Salah satu dasar teoretis penting dalam memahami karakteristik sistem basis data NoSQL dalam konteks sistem terdistribusi adalah Teorema CAP (Consistency, Availability, Partition Tolerance), yang pertama kali dikemukakan oleh Eric Brewer. Teorema ini menyatakan bahwa dalam sistem terdistribusi, tidak mungkin untuk secara bersamaan menjamin konsistensi, ketersediaan, dan toleransi terhadap partisi secara penuh. Oleh karena itu, setiap sistem harus memilih dua dari tiga aspek tersebut sebagai prioritas, bergantung pada kebutuhan dan konteks penggunaannya. Banyak sistem NoSQL modern mengadopsi kombinasi yang berbeda dari ketiga aspek ini guna mengakomodasi beragam skenario aplikasi.

Selain itu, basis data NoSQL juga unggul dalam menangani karakteristik data besar (big data), yang umumnya digambarkan melalui empat dimensi utama: volume (jumlah data yang sangat besar), velocity (kecepatan aliran data), variety (keragaman format data seperti teks, gambar, atau dokumen), dan veracity (tingkat ketidakpastian atau inkonsistensi data) [12]. Tidak seperti basis data relasional yang mengandalkan struktur tabel dan skema tetap, pendekatan NoSQL memungkinkan penyimpanan data dalam format yang lebih fleksibel seperti key-value, dokumen, kolom, maupun graf. Kemampuan ini menjadikan NoSQL sangat efektif untuk menangani data tidak terstruktur, kompleks, dan tersebar secara luas dalam sistem yang elastis dan terdistribusi.

III. METODOLOGI PENELITIAN

A. Perancangan Sistem

Percangan Arsitektur Sistem memanfaatkan beberapa layanan AWS dan dapat digambarkan pada diagram dibawah ini.



Gambar 1. Arsitektur Sistem

Client pada diagram merepresentasikan calon pengguna aplikasi, dimana *client telegram* merupakan platform dimana Telegram diakses, seperti Telegram Web ataupun Aplikasi Telegram. *Input* yang dikirimkan oleh pengguna akan diteruskan pada server Telegram. *Input* dari pengguna dapat berupa pesan yang dikirim ataupun pemilihan tombol pada *inline keyboard.*

Telegram Server berperan dalam menerima *input* dari pengguna agar dapat diproses oleh Bot. *Input* yang dikirimkan oleh pengguna akan diterima oleh Server Telegram beserta informasi seperti pesan yang dikirim, nama dan id pengirim yang dapat kemudian digunakan Bot Server untuk diproses dan membalas kembali kepada pengguna dengan data yang diperlukan.

Bot Server merupakan aplikasi *python* yang dijalankan pada Amazon EC2, peran aplikasi ini adalah untuk mengelola logika dasar dari aplikasi seperti menangani input dari pengguna seperti teks ataupun *inline keyboard*. Bot disini berperan sebagai perantara antara Telegram Server dan komponen *backend* AWS lainnya.

API Gateway adalah layanan AWAS yang menyediakan *endpoint* HTTP untuk berkomunikasi dengan fungsi lambda. Fungsi utama komponen ini adalah untuk meneruskan permintaan dari Bot Server kepada *endpoint* yang didapat untuk memperoleh data dari DynamoDB. HTTP API akan digunakan dalam pengembangan karena hanya memerlukan fitur yang sederhana.

Dalam arsitektur sistem bot *Telegram*, penggunaan *HTTP API* yang diintegrasikan dengan komponen AWS lainnya adalah pendekatan modular dan efisien untuk mengelola berbagai jenis data. Setiap tabel pada DynamoDB akan memiliki *API HTTP* khusus dengan rute yang seragam. Berikut adalah uraian rinci mengenai struktur dan implementasinya untuk setiap API yang tersedia:

TABEL I
DEFINISI RUTE API BIAYA PERKULIAHAN

Attribute Name	Method	Description
/items	PUT	Menambahkan/Memperbarui item pada tabel
/items	GET	Mengambil semua item yang tersedia pada tabel
/items/{prodi}	GET	Mengambil item sesuai dengan prodi yang terdapat dalam parameter
/items/{prodi}	DELETE	Menghapus item dengan prodi yang terdapat dalam parameter

TABEL II
DEFINISI RUTE API BEASISWA

Attribute Name	Method	Description
/items	PUT	Menambahkan/Memperbarui item pada tabel
/items	GET	Mengambil semua item yang tersedia pada tabel

/items/{jenis_beasiswa}	GET	Mengambil item sesuai dengan jenis beasiswa yang terdapat dalam parameter
/items/{nama_beasiswa}	DELETE	Menghapus item dengan nama beasiswa yang terdapat dalam parameter

TABEL III
DEFINISI RUTE API JADWAL

Attribute Name	Method	Description
/items	PUT	Menambahkan/Memperbarui item pada tabel
/items	GET	Mengambil semua item yang tersedia pada tabel
/items/{jalur}	GET	Mengambil item sesuai dengan jalur yang terdapat dalam parameter
/items/{jalur}/{periode}	GET	Mengambil item dengan jalur dan periode yang terdapat dalam parameter
/items/{jalur}/{periode}	DELETE	Menghapus item dengan jalur dan periode yang terdapat dalam parameter

TABEL IV
DEFINISI RUTE API STATUS KELULUSAN

Attribute Name	Method	Description
/items	PUT	Menambahkan/Memperbarui item pada tabel
/items	GET	Mengambil semua item yang tersedia pada tabel
/items/{no_pendaftaran}	GET	Mengambil item sesuai dengan no pendaftaran yang terdapat dalam parameter
/items/{no_pendaftaran}	DELETE	Menghapus item dengan no pendaftaran yang terdapat dalam parameter

Adanya implementasi rute diatas memungkinkan penulis untuk melakukan operasi CRUD pada masing-masing tabel yang terdapat pada DyanmoDB.

Lambda adalah fungsi serverless yang bertujuan untuk memproses logika bisnis. Fungsi lambda dalam arsitektur adalah untuk mengakses data dari DynamoDB dan mengembalikan data tersebut kepada Bot Server melalui API Gateway. Sebuah lambda akan dibuat untuk setiap tabel yang ada pada DynamoDB untuk mengelola data pada tabel yang terkait. Fungsi tersebut adalah sebagai berikut:

TABEL V
TABEL FUNGSI LAMBDA BIAYA PERKULIAHAN

Trigger	Input	Output	Associated Resource
API Gateway: GET /items	-	Semua item yang terdapat pada tabel	DynamoDB: Biaya
API Gateway: GET /items/{prodi}	Prodi	Semua item dengan prodi yang dipilih	DynamoDB: Biaya
API Gateway: PUT /items	Data Biaya perkuliahan dalam bentuk JSON	Penambahan item biaya perkuliahan pada tabel	DynamoDB: Biaya
API Gateway: DELETE /items/{prodi}	Prodi	Penghapusan item biaya perkuliahan sesuai prodi pada tabel	DynamoDB: Biaya

TABEL VI
TABEL FUNGSI LAMBDA BEASISWA

Trigger	Input	Output	Associated Resource
API Gateway: GET /items	-	Semua item yang terdapat pada tabel	DynamoDB: Beasiswa
API Gateway: GET /items/{jenis_beasiswa}	Jenis Beasiswa	Semua item dengan jenis beasiswa yang dipilih	DynamoDB: Beasiswa
API Gateway: PUT /items	Data beasiswa JSON	Penambahan item beasiswa pada tabel	DynamoDB: Beasiswa
API Gateway: DELETE /items/{nama_beasiswa}	Nama Beasiswa	Penghapusan item beasiswa sesuai nama beasiswa pada table	DynamoDB: Beasiswa

TABEL VII
TABEL FUNGSI LAMBDA JADWAL

Trigger	Input	Output	Associated Resource
API Gateway: GET /items	-	Semua item yang terdapat pada tabel	DynamoDB: Beasiswa
API Gateway:	Jenis Beasiswa	Semua item dengan	DynamoDB: Beasiswa

JUTEI Edisi Volume. 9 No.1 April 2025
ISSN 2579-3675, e-ISSN 2579-5538
DOI 10.21460/jutei.2025.91.397

GET /items/{jenis_beasiswa}		jenis beasiswa yang dipilih	
API Gateway: PUT /items	Data beasiswa JSON	Penambahan item beasiswa pada tabel	DynamoDB: Beasiswa
API Gateway: DELETE /items/{nama_beasiswa}	Nama Beasiswa	Penghapusan item beasiswa sesuai nama beasiswa pada table	DynamoDB: Beasiswa

TABEL VIII
TABEL FUNGSI LAMBDA STATUS KELULUSAN

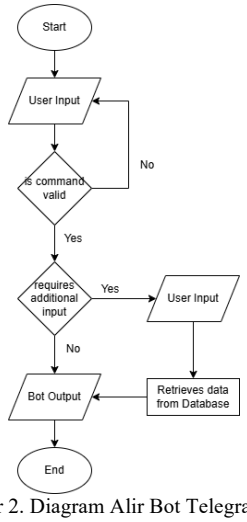
Trigger	Input	Output	Associated Resource
API Gateway: GET /items	-	Semua item yang terdapat pada tabel	DynamoDB: Status
API Gateway: GET/items/{no_pendaftaran}	Nomor Pendaftaran	Item dengan nomor pendaftaran yang dipilih	DynamoDB: Status
API Gateway: PUT /items	Data Status Calon Mahasiswa JSON	Penambahan status calon mahasiswa pada tabel	DynamoDB: Status
API Gateway: DELETE /items/{no_pendaftaran}	Nomor Pendaftaran	Penghapusan item status calon mahasiswa sesuai dengan nomor pendaftaran pada tabel	DynamoDB: Status

DynamoDB adalah sebuah basis data NoSQL yang digunakan untuk menyimpan data yang perlu ditampilkan pada Bot. Data yang disimpan pada basis data ini terdiri dari tabel berikut:

TABEL IX
LIST TABEL PADA DYNAMODB

Table Name	Partition Key	Sort Key	Description
Beasiswa	Tahun_ajaran (String)	Nama_beasiswa (String)	Menyimpan data terkait informasi beasiswa
Biaya	Tahun_ajaran (String)	Prodi (String)	Menyimpan data terkait biaya
Jadwal	Tahun_ajaran (String)	Jalur#Periode (String)	Menyimpan data terkait jadwal ujian
Status_kelulusan	Tahun_ajaran (String)	No_Pendaftaran (String)	Menyimpan data terkait status kelulusan ujian calon mahasiswa

B. Diagram Alir Sistem



Gambar 2. Diagram Alir Bot Telegram

Gambar di atas merupakan alur diagram secara umum dari sistem yang akan dikembangkan. Sistem akan dimulai dengan penerimaan input dari pengguna berdasarkan menu yang ada, jika input dari pengguna bukan merupakan command yang valid maka user akan diminta untuk melakukan input lagi. Jika sebuah command adalah valid dan memerlukan input tambahan dari pengguna, maka data yang diperlukan akan diambil dari database berdasarkan input yang diterima dan Bot akan mengembalikan output kepada pengguna. Jika tidak memerlukan input tambahan maka bot akan mengembalikan output langsung kepada pengguna.

C. Perancangan Pengujian

Pengujian dilakukan dengan menggunakan metode *blackbox testing,* dimana pengujian berfokus pada fungsionalitas sistem tanpa mengetahui detail internal, struktur, atau kode program. Penguji hanya memeriksa apakah input menghasilkan output yang sesuai dengan spesifikasi tanpa memedulikan bagaimana proses tersebut terjadi di dalam sistem. Dengan scenario pengujian berdasarkan blackbox dapat dijabarkan dalam tabel dibawah ini:

TABEL X
TABEL SKENARIO PENGUJIAN

No	Scenario	Expected Result	Validity
1	Pengguna mengirimkan perintah /start	/start	Bot menampilkan pesan selamat datang beserta menu utama dalam bentuk inline keyboard
2	Pengguna memilih opsi "Biaya Perkuliahan" pada inline keyboard	Klik tombol "Biaya Perkuliahan"	Bot menampilkan daftar program studi yang tersedia
3	Pengguna memilih program studi tertentu untuk melihat biaya perkuliahan	Klik salah satu program studi dari daftar yang tersedia	Bot menampilkan rincian biaya perkuliahan sesuai dengan program studi yang dipilih
4	Pengguna memilih opsi "Jadwal Tes" pada inline keyboard	Klik tombol "Jadwal Tes"	Bot menampilkan daftar jadwal tes berdasarkan kategori seleksi
5	Pengguna memilih jadwal tes tertentu	Klik salah satu jadwal dari daftar yang tersedia	Bot menampilkan informasi detail mengenai jadwal tes yang dipilih
6	Pengguna memilih opsi "Beasiswa" pada inline keyboard	Klik tombol "Beasiswa"	Bot menampilkan informasi mengenai program beasiswa yang tersedia
7	Pengguna memilih opsi "Cek Status Kelulusan" pada inline keyboard	Klik tombol "Cek Status Kelulusan"	Bot meminta pengguna untuk memasukkan nomor pendaftaran untuk mengecek status kelulusan

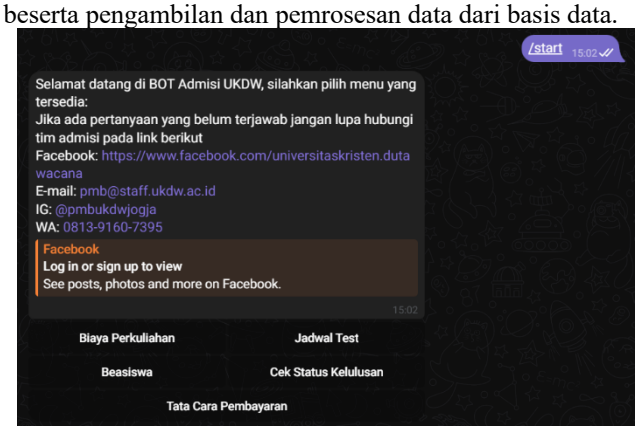
IV. Implementasi dan Pembahasan

A. EC2 Server

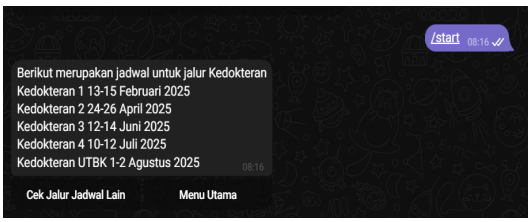
Implementasi awal dari sebuah server EC2 dimulai dari membuat sebuah *instance* EC2 baru beserta beberapa konfigurasinya seperti berikut:

- Nama : telegram_bot
- OS : Amazon Linux 2023 (64-bit x86)
- Type : t2.micro
- Storage: 8 Gib

Setelah konfigurasi awal dari EC2, akan dilakukan instalasi modul yang diperlukan untuk menjalankan Bot Telegram. Codebase dari Bot Telegram akan diclone dari GitHub pribadi penulis yang berisi semua logika dalam bot beserta pengambilan dan pemrosesan data dari basis data.



Gambar 3. Tampilan utama ketika pengguna memanggil /

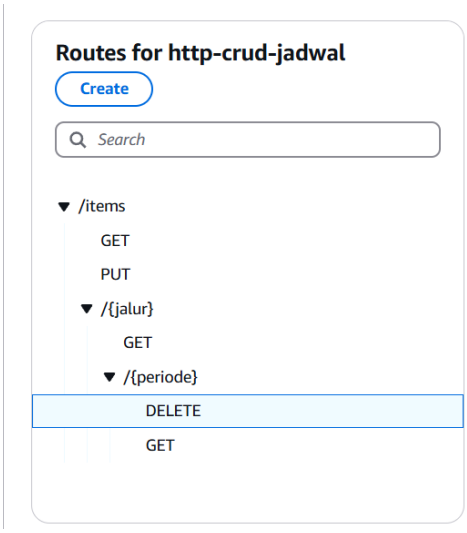


Gambar 4. Tampilan pilih jadwal tes kedokteran

B. API Gateway

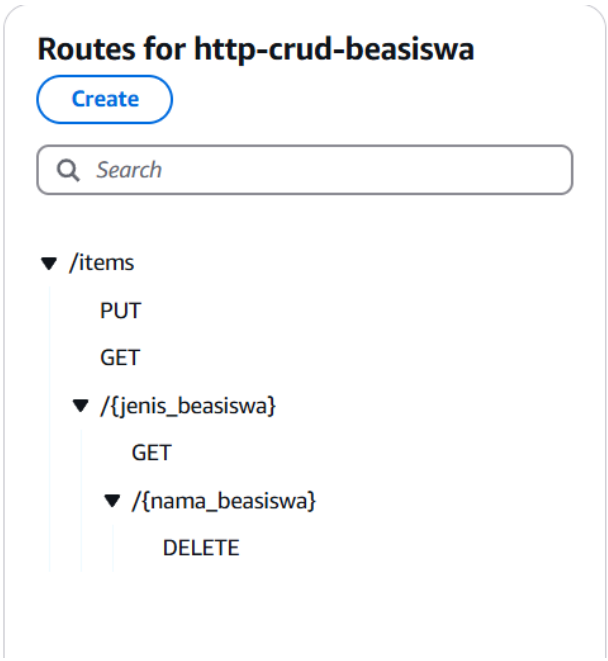
API Gateway merupakan sebuah perantara dari Bot Sever dengan layanan AWS lainnya. Implementasi dari API Gateway diawali dengan pembuatan HTTP API untuk setiap tabel yang ada pada DynamoDB. Implementasi route dan metode dari masing-masing API adalah sebagai berikut:

JUTEI Edisi Volume. 9 No.1 April 2025
ISSN 2579-3675, e-ISSN 2579-5538
DOI 10.21460/jutei.2025.91.397



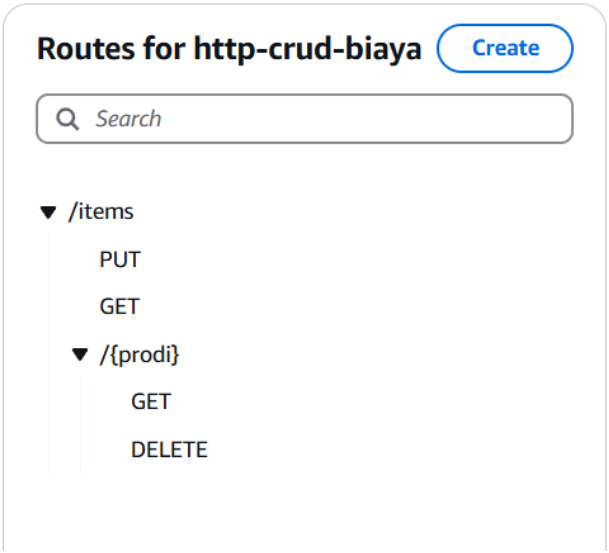
Gambar 5. Definisi *route* untuk API jadwal

Struktur *route* pada gambar 5 memungkinkan bot untuk *create*, , *update*, dan *delete* data jadwal berdasarkan parameter yang spesifik. *Route* "/items GET" memungkinkan bot untuk mengambil semua data yang tersedia dengan tahun ajaran terbaru, sedangkan "/items PUT" memungkinkan pengguna (admin) untuk menambahkan jadwal baru pada tampilan antarmuka khusus admin. Route "/items/{jalur} GET" memungkinkan bot untuk mengambil data sesuai dengan jalur yang dipilih. Route "/items/{jalur}/{periode} GET" memungkinkan bot untuk mengambil data dengan jalur dan periode sesuai spesifikasi dan Route "/items/{jalur}/{periode} DELETE" memungkinkan pengguna (admin) untuk melakukan penghapusan pada jadwal dengan jalur dan periode sesuai spesifikasi.



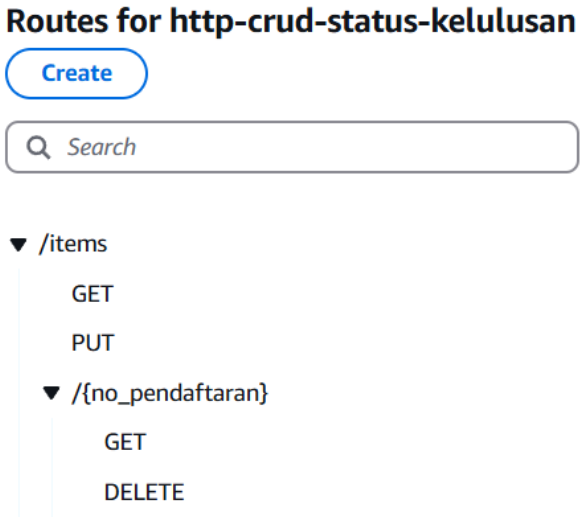
Gambar 6. Definisi *route* untuk API beasiswa

Struktur *route* ini memungkinkan bot untuk *create*, , *update*, dan *delete* data beasiswa berdasarkan parameter yang spesifik. *Route* "/items GET" memungkinkan bot untuk mengambil semua data yang tersedia dengan tahun ajaran terbaru, sedangkan "/items PUT" memungkinkan pengguna (admin) untuk menambahkan beasiswa baru pada tampilan antarmuka khusus admin. Route "/items/{jenis_beasiswa} GET" memungkinkan bot untuk mengambil data sesuai dengan jenis beasiswa yang dipilih. Route "/items/{jenis_beasiswa}/{nama_beasiswa} DELETE" memungkinkan pengguna (admin) untuk melakukan penghapusan pada beasiswa dengan nama beasiswa sesuai spesifikasi.



Gambar 7. Definisi *route* untuk API biaya

Struktur *route* ini memungkinkan bot untuk *create*, , *update*, dan *delete* data biaya berdasarkan parameter yang spesifik. *Route* "/items GET" memungkinkan bot untuk mengambil semua data yang tersedia dengan tahun ajaran terbaru, sedangkan "/items PUT" memungkinkan pengguna (admin) untuk menambahkan biaya baru pada tampilan antarmuka khusus admin. Route "/items/{prodi} GET" memungkinkan bot untuk mengambil data sesuai dengan prodi yang dipilih. Route "/items/{prodi}/ DELETE" memungkinkan pengguna (admin) untuk melakukan penghapusan pada biaya dengan prodi sesuai spesifikasi.



Gambar 8. Definisi *route* untuk API status kelulusan

Struktur *route* ini memungkinkan bot untuk *create*, , *update*, dan *delete* data status kelulusan berdasarkan parameter yang spesifik. *Route* "/items GET" memungkinkan bot untuk mengambil semua data yang tersedia dengan tahun ajaran terbaru, sedangkan "/items PUT" memungkinkan pengguna (admin) untuk menambahkan status kelulusan baru pada tampilan antarmuka khusus admin. Route "/items/{no_pendaftaran} GET" memungkinkan bot untuk mengambil data sesuai dengan nomor pendaftaran yang dipilih. Route "/items/{no_pendaftaran}/ DELETE" memungkinkan pengguna (admin) untuk melakukan penghapusan pada status calon mahasiswa dengan nomor pendaftaran sesuai spesifikasi.

C. *Lambda*

Implementasi dari Lambda sebagai proses logika bisnis mencakup beberapa hal seperti penerimaan data dari DynamoDB, menambahkan data, menghapus data dan mengambil data dengan parameter spesifik seperti prodi untuk tabel biaya. Penerapan Lambda juga membawa kelebihan dalam bentuk CloudWatch Logs, yang merupakan sebuah layanan dari AWS yang berfungsi untuk mencatat semua panggilan API beserta data yang diambil. Operasi basisdata untuk Lambda guna melakukan operasi *CRUD* untuk setiap tabel yang terdapat pada DynamoDB akan digambarkan dalam Gambar 9 sampai Gambar 25.

```
if event['routeKey'] == "DELETE /items/{jalur}/{periode}":
    jalur_periode = event['pathParameters']['jalur'] + "#" + event['pathParameters']['periode']
    print(jalur_periode)
    table.delete_item(
        Key={'jalur#periode': jalur_periode, 'tahun_ajaran': tahun_ajaran})
    body = 'Deleted item ' + jalur_periode
```

Gambar 9. Operasi *DELETE* untuk tabel jadwal

Kode pada gambar 9 merupakan bagian dari fungsi AWS Lambda yang menangani request DELETE untuk menghapus item dari tabel berdasarkan parameter jalur dan periode yang dikirim melalui API Gateway.

```
elif event['routeKey'] == "GET /items/{jalur}":
    response = table.query(
        KeyConditionExpression=Key('tahun_ajaran').eq(tahun_ajaran),
        FilterExpression=Attr('jalur').eq(event['pathParameters']['jalur'])
    )

    if 'Items' in response:
        body = response['Items']
    else:
        body = []
```

Gambar 10. Operasi READ dengan parameter jalur untuk tabel jadwal

Kode pada gambar 10 merupakan bagian dari fungsi AWS Lambda yang menangani request GET untuk mengambil item dari tabel berdasarkan parameter jalur yang dikirim melalui API Gateway.

```
elif event['routeKey'] == "GET /items/{jalur}/{periode}":
    response = table.query(
        KeyConditionExpression=Key('tahun_ajaran').eq(tahun_ajaran),
        FilterExpression=Attr('jalur').eq(event['pathParameters']['jalur']) &
        Attr('periode').eq(event['pathParameters']['periode'])
    )

    if 'Items' in response:
        body = response['Items']
    else:
        body = []
```

Gambar 11. Operasi dengan parameter jalur dan periode untuk tabel jadwal

Kode pada gambar 11 merupakan bagian dari fungsi AWS Lambda yang menangani request GET untuk mengambil item dari tabel berdasarkan parameter jalur dan periode yang dikirim melalui API Gateway.

```
elif event['routeKey'] == "GET /items":
    body = table.scan()
    body = body["Items"]
    print("ITEMS----")
    print(body)
    responseBody = []
    for items in body:
        responseBody.append(items)
    body = responseBody
```

Gambar 12. Operasi semua item untuk tabel jadwal

Kode pada gambar 12 merupakan bagian dari fungsi AWS Lambda yang menangani request GET untuk mengambil semua item dari tabel yang terdapat pada *database*.

```
elif event['routeKey'] == "PUT /items":
    requestJSON = json.loads(event['body'])
    table.put_item(Item=requestJSON)
    body = 'Put item ' + requestJSON['tahun_ajaran']
```

Gambar 13. Operasi *CREATE/UPDATE* untuk tabel jadwal

Kode pada gambar 13 merupakan bagian dari fungsi AWS Lambda yang menangani request PUT untuk menambahkan atau memperbarui item dalam tabel berdasarkan data yang dikirim melalui body request di API Gateway.

```
if event['routeKey'] == "DELETE /items/{nama_beasiswa}":
    table.delete_item(
        Key={'nama_beasiswa': event['pathParameters']['nama_beasiswa'], 'tahun_ajaran': tahun_ajaran})
    body = 'Deleted item ' + event['pathParameters']['nama_beasiswa']
```

Gambar 14. Operasi *DELETE* untuk tabel beasiswa

Kode pada gambar 14 merupakan bagian dari fungsi AWS Lambda yang menangani request DELETE untuk menghapus item dari tabel berdasarkan parameter nama beasiswa yang dikirim melalui API Gateway.

```
elif event['routeKey'] == "GET /items/{jenis_beasiswa}":
    response = table.query(
        KeyConditionExpression=Key('tahun_ajaran').eq(tahun_ajaran),
        FilterExpression=Attr('jenis_beasiswa').eq(event['pathParameters']['jenis_beasiswa'])
    )

    if 'Items' in response:
        body = response['Items']
    else:
        body = []
```

Gambar 15. Operasi READ dengan parameter jenis_beasiswa pada tabel beasiswa

JUTEI Edisi Volume. 9 No.1 April 2025
ISSN 2579-3675, e-ISSN 2579-5538
DOI 10.21460/jutei.2025.91.397

Kode pada gambar 15 merupakan bagian dari fungsi AWS Lambda yang menangani request GET untuk mengambil item dari tabel berdasarkan parameter jenis beasiswa yang dikirim melalui API Gateway.

```
elif event['routeKey'] == "GET /items":
    body = table.scan()
    body = body["Items"]
    print("ITEMS----")
    print(body)
    responseBody = []
    for items in body:
        responseBody.append(items)
    body = responseBody
```

Gambar 16. Operasi READ untuk semua item pada tabel beasiswa

Kode pada gambar 16 merupakan bagian dari fungsi AWS Lambda yang menangani request GET untuk mengambil semua item dari tabel yang terdapat pada *database*.

```
    body = responseBody
elif event['routeKey'] == "PUT /items":
    requestJSON = json.loads(event['body'])
    table.put_item(Item=requestJSON)
    body = 'Put item ' + requestJSON['jenis_beasiswa']
```

Gambar 17. Operasi CREATE/UPDATE pada tabel beasiswa

Kode pada gambar 17 merupakan bagian dari fungsi AWS Lambda yang menangani request PUT untuk menambahkan atau memperbarui item dalam tabel berdasarkan data yang dikirim melalui body request di API Gateway.

```
if event['routeKey'] == "DELETE /items/{prodi}":
    table.delete_item(
        Key={'prodi': event['pathParameters']['prodi'], 'tahun_ajaran': tahun_ajaran})
    body = 'Deleted item ' + event['pathParameters']['prodi']
```

Gambar 18. Operasi *DELETE* pada tabel biaya

Kode pada gambar 18 merupakan bagian dari fungsi AWS Lambda yang menangani request DELETE untuk menghapus item dari tabel berdasarkan parameter prodi yang dikirim melalui API Gateway.

```
elif event['routeKey'] == "GET /items/{prodi}":
    response = table.query(
        KeyConditionExpression=Key('tahun_ajaran').eq(tahun_ajaran) & Key('prodi').eq(event['pathParameters']['prodi'])
    )

    if 'Items' in response:
        body = response['Items']
    else:
        body = []
```

Gambar 19. Operasi READ dengan parameter prodi untuk tabel biaya

Kode pada gambar 19 merupakan bagian dari fungsi AWS Lambda yang menangani request GET untuk mengambil item dari tabel berdasarkan parameter prodi yang dikirim melalui API Gateway.

```
elif event['routeKey'] == "GET /items":
    body = table.scan()
    body = body["Items"]
    print("ITEMS----")
    print(body)
    responseBody = []
    for items in body:
        responseBody.append(items)
    body = responseBody
```

Gambar 20. Operasi READ untuk semua item untuk tabel biaya

Kode pada gambar 20 merupakan bagian dari fungsi AWS Lambda yang menangani request GET untuk mengambil semua item dari tabel yang terdapat pada *database*.

```
elif event['routeKey'] == "PUT /items":
    requestJSON = json.loads(event['body'])
    table.put_item(Item=requestJSON)
    body = 'Put item ' + requestJSON['prodi']
```

Gambar 21. Operasi CREATE/UPDATE untuk tabel biaya

Kode pada gambar 21 merupakan bagian dari fungsi AWS Lambda yang menangani request PUT untuk menambahkan atau memperbarui item dalam tabel berdasarkan data yang dikirim melalui body request di API Gateway.

```
if event['routeKey'] == "DELETE /items/{no_pendaftaran}":
    table.delete_item(
        Key={'no_pendaftaran': event['pathParameters']['no_pendaftaran'], 'tahun_ajaran': tahun_ajaran})
    body = 'Deleted item ' + event['pathParameters']['no_pendaftaran']
```

Gambar 22. Operasi *DELETE* untuk tabel status kelulusan

Kode pada gambar 22 merupakan bagian dari fungsi AWS Lambda yang menangani request DELETE untuk menghapus item dari tabel berdasarkan nomor pendaftaran yang dikirim melalui API Gateway.

```
elif event['routeKey'] == "GET /items/{no_pendaftaran}":
    response = table.scan(
        FilterExpression=Attr('no_pendaftaran').eq(event['pathParameters']['no_pendaftaran'])
    )

    if 'Items' in response:
        body = response['Items']
    else:
        body = []
```

Gambar 23. Operasi *READ* dengan parameter nomor pendaftaran pada tabel status kelulusan

Kode pada gambar 23 merupakan bagian dari fungsi AWS Lambda yang menangani request GET untuk mengambil item dari tabel berdasarkan parameter nomor pendaftaran yang dikirim melalui API Gateway.

```
elif event['routeKey'] == "GET /items":
    body = table.scan()
    body = body["Items"]
    print("ITEMS----")
    print(body)
    responseBody = []
    for items in body:
        responseBody.append(items)
    body = responseBody
```

Gambar 24. Operasi READ untuk semua item pada tabel status kelulusan

Kode pada gambar 24 merupakan bagian dari fungsi AWS Lambda yang menangani request GET untuk mengambil semua item dari tabel yang terdapat pada *database*.

```
    body = responseBody
elif event['routeKey'] == "PUT /items":
    requestJSON = json.loads(event['body'])
    table.put_item(Item=requestJSON)
    body = 'Put item ' + requestJSON['no_pendaftaran']
```

Gambar 25. Operasi CREATE/UPDATE pada tabel status kelulusan

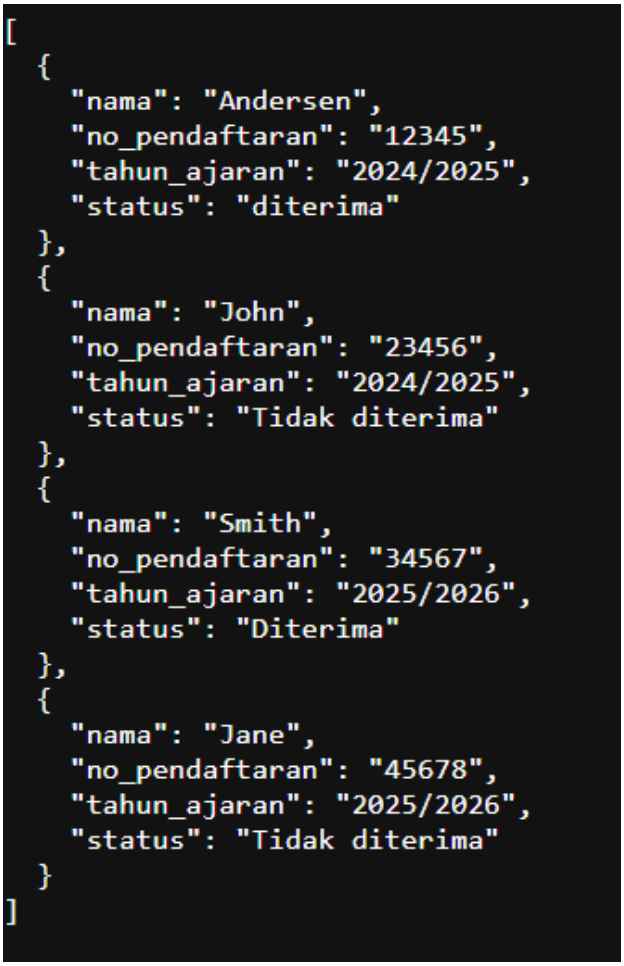
Kode pada gambar 25 merupakan bagian dari fungsi AWS Lambda yang menangani request PUT untuk menambahkan atau memperbarui item dalam tabel berdasarkan data yang dikirim melalui body request di API Gateway.

D. DynamoDB

DynamoDB sebagai basis data NoSQL dan menggunakan prinsip key pair menambahkan fleksibilatas pada perancangan basis data. Setiap *entry* data dapat memiliki jumlah kolom yang berbeda, sehingga memungkinkan untuk menambahkan atau mengurangi kolom sesuai dengan data yang ada.

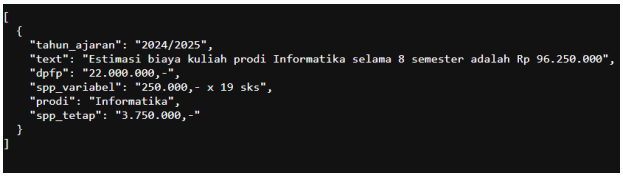
Gambar 26. Contoh perbedaan jumlah kolom pada DynamoDB

Gambar 26 menunjukkan jalur prestasi memiliki sebuah key berlebihan yaitu tgl_pengumuman. Adanya tanggal pengumuman ini merupakan akibat dari hasil pengumuman yang tidak spontan, tidak seperti jalur lainnya. Selain itu penggunaan *composite key* pada tabel jadwal memungkinkan *query* yang jauh lebih detil. Penulis dapat melakukan *query* berdasarkan jalurnya saja, periodenya saja ataupun keduanya sekaligus untuk mendapatkan data yang lebih spesifik.



Gambar 27. Tampilan data pada tabel status kelulusan

Gambar 27 menampilkan hasil dari operasi GET /items untuk tabel status kelulusan. Hasil dari operasi tersebut mengembalikan semua data yang terdapat pada tabel tanpa parameter tertentu.



Gambar 28. Tampilan data pada tabel biaya perkuliahan dengan parameter Informatika

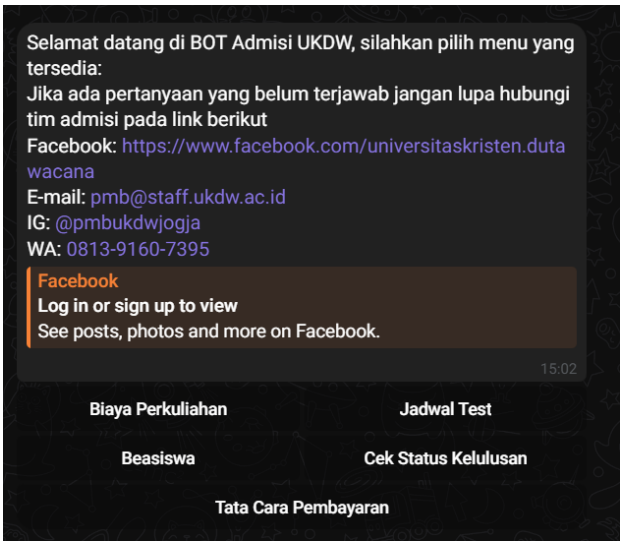
Gambar 28 menampilkan hasil dari operasi GET /items/Informatika untuk tabel biaya perkuliahan. Hasil dari operasi tersebut hanya mengembalikan data dengan *partition key* "Informatika".

E. Pengujian Sistem

Setelah implementasi sistem dilakukan, peneliti akan melanjutkan dengan melakukan pengujian sistem menggunakan metode *black-box testing*. Pengujian ini bertujuan untuk mengevaluasi fungsionalitas bot Telegram dengan menguji berbagai skenario interaksi pengguna tanpa melihat kode sumbernya. Pengujian dilakukan berdasarkan skenario yang telah disusun pada Bab 3, yang mencakup berbagai aspek seperti validasi perintah, respons terhadap

JUTEI Edisi Volume. 9 No.1 April 2025
ISSN 2579-3675, e-ISSN 2579-5538
DOI 10.21460/jutei.2025.91.397

input pengguna, serta keandalan dalam mengambil dan menampilkan data dari DynamoDB. Hasil dari pengujian ini akan dianalisis untuk memastikan bahwa sistem berjalan sesuai dengan spesifikasi yang telah ditentukan.



Gambar 29. Tampilan setelah menjalankan perintah */start*

Pengujian ini dilakukan dengan melibatkan 8 mahasiswa dan 2 relawan dari pihak admisi sebagai responden. Setiap responden bertindak sebagai pengguna dan menguji berbagai fitur bot, termasuk navigasi menu, pencarian informasi biaya perkuliahan, jadwal tes, beasiswa, serta pengecekan status kelulusan. Pengujian ini bertujuan untuk memastikan bahwa bot dapat menangani berbagai skenario interaksi dengan akurat dan memberikan respons yang sesuai terhadap input pengguna. Rincian dari pengujian akan ditampilkan pada tabel XI.

TABEL XI
TABEL HASIL PENGUJIAN

No	Test Scenario	Expected Result	Actual Result	Rate of Success
1	Pengguna mengirimkan perintah /start	Bot menampilkan pesan selamat datang beserta menu utama dalam bentuk inline keyboard	Sesuai	100%
2	Pengguna memilih opsi "Biaya Perkuliahan" pada inline keyboard	Bot menampilkan daftar program studi yang tersedia	Sesuai	100%
3	Pengguna memilih program studi tertentu untuk melihat biaya perkuliahan	Bot menampilkan rincian biaya perkuliahan sesuai dengan program studi yang dipilih	Sesuai	100%
4	Pengguna memilih opsi "Jadwal Tes" pada inline keyboard	Bot menampilkan daftar jadwal tes berdasarkan kategori seleksi	Sesuai	100%
5	Pengguna memilih jadwal tes tertentu	Bot menampilkan informasi detail mengenai jadwal tes yang dipilih	Sesuai	100%
6	Pengguna memilih opsi "Beasiswa" pada inline keyboard	Bot menampilkan informasi mengenai program beasiswa yang tersedia	Sesuai	100%
7	Pengguna memilih opsi "Cek Status Kelulusan" pada inline keyboard	Bot meminta pengguna untuk memasukkan nomor pendaftaran untuk mengecek status kelulusan	Sesuai	100%
8	Pengguna memasukkan nomor pendaftaran yang valid saat mengecek status kelulusan	Bot menampilkan pesan calon mahasiswa dengan nomor pendaftaran 12345 dinyatakan lulus	Sesuai	100%
9	Pengguna memasukkan nomor pendaftaran yang tidak valid saat mengecek status kelulusan	Bot menampilkan pesan calon mahasiswa dengan nomor pendaftaran 54321 tidak ditemukan	Sesuai	100%
10	Pengguna memilih opsi "Kembali" pada inline keyboard	Bot mengembalikan pengguna ke menu sebelumnya	Sesuai	100%
11	Pengguna memilih opsi "Menu Utama" setelah melihat informasi tertentu	Bot mengembalikan pengguna ke menu utama dengan daftar opsi yang tersedia	Sesuai	100%

V. KESIMPULAN

Berdasarkan pengujian black-box terhadap bot Telegram admisi UKDW yang bertujuan untuk mengevaluasi fungsionalitas, akurasi respons, dan keandalan sistem. Semua kasus uji berhasil dijalankan, dengan bot mampu menangani berbagai interaksi pengguna. Penerapan bot Telegram layanan AWS seperti *Lambda*, *DynamoDB* dan *API Gateway* memungkinan pengguna untuk selalu mendapatkan informasi terkini. Beberapa hal kunci yang ditarik dari pengujian adalah sebagai berikut:

1. Bot dapat menangani perintah dari pengguna sesuai dengan ekspektasi
2. Bot berhasil mengambil dan menampilkan informasi relevan dan terkini dari DynamoDB
3. Pengguna dapat berinteraksi dengana bot menggunakan *inline keyboard* sehingga mempermudah navigasi

Berdasarkan pengembangan yang telah dilakukan, ada beberapa saran yang dapat disampaikan peneliti dalam pengembangan selanjutnya, yaitu:

1. Integrasi dengan lingkungan *e-class* maupun *moodle* dapat memperluas *use case* dari bot yang mencakup kepentingan akademis
2. Optimisasi pada struktur *database* DynamoDB dengan menggunakan Global Secondary Index (GSI) ataupun penerapan *single-table design*

VI. DAFTAR PUSTAKA

[1] A. Lubis and I. Sumartono, "Implementasi Layanan Akademik Berbasis Chatbot untuk Meningkatkan Interaksi Mahasiswa," Resolusi : Rekayasa Teknik Informatika dan Informasi, 2023.

[2] D. M. Alfiansyah, W. L. Setiyani, D. F. Wati and D. , "Pengembangan Chatbot Berbasis Web untuk Layanan Informasi di Horizon University," Jurnal Teknologi dan Sistem Komputer, vol. 7, no. 3, pp. 1068-1077, 2025.

[3] F. Z. Alfaiz and M. , "IMPLEMENTATION TELEGRAM CHAT BOT ON STUDENT ORIENTATION PERIOD REGISTRATION SYSTEM FOR EFFICIENCY OF DATA MANAGEMENT," J. Tek. Inform. (JUTIF), vol. 2, no. 2, pp. 85-93, 2021.

[4] D. Lestari, F. Ramdhani, M. J. Ruliansyah, R. B. Richardviki Beay and D. I. Mulyana, "Implementasi Chatbot Telegram Dalam Meningkatkan Partisipasi Kegiatan Warga," Vol. 4 No. 2 (2023): Jurnal Pengabdian kepada Masyarakat Nusantara (JPkMN), 2023.

[5] G. C. Lenardo, H. and Y. Irawan, "Pemanfaatan Bot Telegram sebagai Media Informasi Akademik di STMIK Hang Tuah Pekanbaru," Jurnal Teknologi Informasi dan Multimedia (JTIM) vol. 1, no. 4, pp. 351-357, 2020.

[6] M. Qalimaturrahmah and D. B. Santoso, "Aplikasi Layanan dan Informasi Akademik Berbasis Chatbot Telegram Menggunakan Natural Language Processing," JTIK (Jurnal Teknologi Informasi dan Komunikasi), vol. 8, no. 2, pp. 434-443, 2024.

[7] H. Mulyo and G. Mohammad, "INTEGRASI SISTEM INFORMASI AKADEMIK MAHASISWA DENGAN BOT TELEGRAM SEBAGAI MESIN PENJAWAB OTOMATIS," Jurnal DISPROTEK, vol. 13, no. 1, pp. 11-20, 2022.

[8] A. H. Wibowo, "ARSITEKTUR EVEN DRIVEN UNTUK MENDUKUNG IMPLEMENTASI CRM PADA APLIKASI E-COMMERCE," semanTIK, vol. 9, no. 1, pp. 33-38, 2023.

[9] S. K. Choudhary, "Implementing Event-Driven Architecture for Real-Time Data Integration in Cloud Environments," INTERNATIONAL JOURNAL OF COMPUTER ENGINEERING & TECHNOLOGY 16(1):1535-1552, vol. 16, no. 1, pp. 1535-1552, 2025.

[10] A. R. Kommera, "The Power of Event-Driven Architecture: Enabling RealTime Systems and Scalable Solutions," Turkish Journal of Computer and Mathematics Education (TURCOMAT), vol. 11, no. 1, pp. 1740-1751, 2020.

[11] V. Lannurien, L. d'Orazio, O. Barais and J. Boukhobza, "Serverless Cloud Computing: State of the Art and Challenges," in Serverless Computing: Principles and Paradigms, Springer, 2023, pp. 275-316.

[12] W. Khan, T. Kumar, C. Zhang, K. Raj, A. M. Roy and B. Luo, "SQL and NoSQL Database Software Architecture Performance Analysis and Assessments—A Systematic Literature Review," Big Data Cogn. Comput., vol. 7, no. 2, p. 97, 2023.

JUTEI Edisi Volume. 9 No.1 April 2025
ISSN 2579-3675, e-ISSN 2579-5538
DOI 10.21460/jutei.2025.91.397