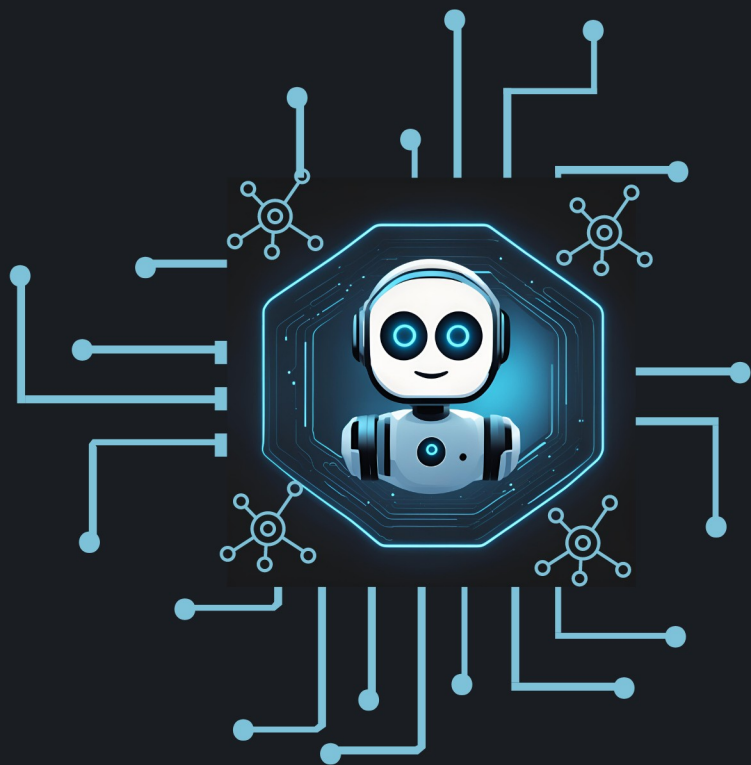


MEMBANGUN PLATFORM BOT DENGAN DJANGO



**PANDUAN LANGKAH DEMI LANGKAH MEMBANGUN
PLATFORM PEMBUATAN BOT DARI AWAL**



Membangun Platform Bot dengan Django

Panduan langkah demi langkah untuk membangun platform pembuatan bot dengan Django, termasuk terintegrasi dengan bot Telegram dan WhatsApp.

Zaenal Arifin

Hak Cipta © 2024 oleh Zaenal Arifin. Semua hak dilindungi undang-undang

UU No 28 tahun 2014 tentang Hak Cipta

Fungsi dan sifat hak cipta Pasal 4

Hak Cipta sebagaimana dimaksud dalam Pasal 3 huruf a merupakan hak eksklusif yang terdiri atas hak moral dan hak ekonomi.

Pembatasan Pelindungan Pasal 26

Ketentuan sebagaimana dimaksud dalam Pasal 23, Pasal 24, dan Pasal 25 tidak berlaku terhadap:

- i. penggunaan kutipan singkat Ciptaan dan/atau produk Hak Terkait untuk pelaporan peristiwa aktual yang ditujukan hanya untuk keperluan penyediaan informasi aktual;
- ii. Penggandaan Ciptaan dan/atau produk Hak Terkait hanya untuk kepentingan penelitian ilmu pengetahuan;
- iii. Penggandaan Ciptaan dan/atau produk Hak Terkait hanya untuk keperluan pengajaran, kecuali pertunjukan dan Fonogram yang telah dilakukan Pengumuman sebagai bahan ajar; dan
- iv. penggunaan untuk kepentingan pendidikan dan pengembangan ilmu pengetahuan yang memungkinkan suatu Ciptaan dan/atau produk Hak Terkait dapat digunakan tanpa izin Pelaku Pertunjukan, Produser Fonogram, atau Lembaga Penyiaran.

Sanksi Pelanggaran Pasal 113

1. Setiap Orang yang dengan tanpa hak melakukan pelanggaran hak ekonomi sebagaimana dimaksud dalam Pasal 9 ayat (1) huruf i untuk Penggunaan Secara Komersial dipidana dengan pidana penjara paling lama 1 (satu) tahun dan/atau pidana denda paling banyak Rp100.000.000 (seratus juta rupiah).
2. Setiap Orang yang dengan tanpa hak dan/atau tanpa izin Pencipta atau pemegang Hak Cipta melakukan pelanggaran hak ekonomi Pencipta sebagaimana dimaksud dalam Pasal 9 ayat (1) huruf c, huruf d, huruf f, dan/atau huruf h untuk Penggunaan Secara Komersial dipidana dengan pidana penjara paling lama 3 (tiga) tahun dan/atau pidana denda paling banyak Rp500.000.000,00 (lima ratus juta rupiah).

Membangun Platform Bot dengan Django

Zaenal Arifin

Editor :

Zaenal Arifin

Desain Cover :

Zaenal Arifin

Ukuran :

109033 Kata 836 Halaman. Uk: 14x20 cm

ISBN :

No ISBN

Cetakan Pertama :

12/09/24

Hak Cipta 2024, Zaenal Arifin

Isi diluar tanggung jawab percetakan

Copyright © 2024 by Zaenal Arifin

All Right Reserved

Hak cipta dilindungi undang-undang
Dilarang keras menerjemahkan, memfotokopi, atau
memperbanyak sebagian atau seluruh isi buku ini
tanpa izin tertulis dari Penerbit.

PENERBIT AigoreTech

Indonesia, Ciamis, Desa Rawa, Kecamatan Lumbung, Dusun Bojong
Sukamulya, 46258

Website: www.aigoretech.my.id

E-mail: admin@aigoretech.my.id

Kata Pengantar

Selamat datang di eBook ini!

Kami senang Anda memilih membaca karya ini. eBook ini dirancang untuk memberikan pemahaman yang jelas dan praktis tentang topik yang kami bahas, tanpa memerlukan detail teknis yang rumit.

Dalam eBook ini, kami telah menyusun materi dengan cara yang jelas dan terstruktur untuk membantu Anda memahami konsep-konsep kunci serta aplikasi praktisnya. Baik Anda seorang pemula yang baru memulai perjalanan belajar atau seseorang yang ingin memperdalam pemahaman dalam bidang ini, kami harap eBook ini dapat memenuhi kebutuhan Anda.

Kami berharap materi ini bermanfaat dan membantu Anda dalam perjalanan belajar Anda. Jika Anda memiliki umpan balik atau pertanyaan, jangan ragu untuk menghubungi kami.

Zaenal Arifin

Daftar Isi

Kata Pengantar.....	vii
Daftar Isi.....	ix
PENDAHULUAN.....	xxi
BAB 1 - Pengenalan Django.....	37
1.1 Apa itu Django?.....	38
1.2 Sejarah Singkat Django.....	40
1.3 Fitur Utama Django.....	44
1.3.1 ORM (Object-Relational Mapping):.....	44
1.3.2 Admin Panel Otomatis:.....	45
1.3.3 Keamanan:.....	46
1.3.4 Templating System:.....	48
1.3.5 Routing URL yang Fleksibel:.....	49
1.3.6 Komunitas dan Dokumentasi yang Kuat:.....	51
1.3.7 Skalabilitas dan Performa.....	51
1.4 Keunggulan Django.....	53
1.4.1 Pengembangan Cepat.....	53
1.4.2 Keamanan Terintegrasi.....	55
1.4.3 Skalabilitas.....	55
1.4.4 Arsitektur yang Terstruktur.....	57
1.4.5 Admin Interface yang Otomatis.....	58
1.4.6 Dukungan untuk Berbagai Database.....	59
1.4.7 Fitur Built-in untuk Pengembangan Web.....	60
1.4.8 Kemudahan Pengujian.....	60
1.4.9 Dukungan untuk Pengembangan Aplikasi Besar.....	62
1.5 Arsitektur MVC dalam Django.....	63
1.5.1 Model.....	64
1.5.2 View.....	64

1.5.3 Template.....	65
1.5.4 Bagaimana MVT Bekerja Bersama.....	66
1.6 Komponen-Komponen Utama Django.....	67
1.6.1 App.....	68
1.6.2 Template.....	73
1.6.3 URLs.....	78
1.6.4 Views.....	83
1.6.5 Models.....	88
1.6.6 Forms.....	93
1.6.7 Static Files dan Media Files.....	99
1.6.8 Middleware.....	105
1.6.9 Signals.....	109
1.7 Mengapa Memilih Django?.....	115
1.7.1 Keunggulan Django dalam Pengembangan Web...120	
1.7.2 Django vs Framework Lainnya.....	125
1.7.3 Django untuk Proyek Kecil hingga Skala Besar....	129
1.7.4 Kasus Penggunaan Django dalam Dunia Nyata....	132
Kesimpulan Bab.....	137
BAB 2 - Persiapan Dan Instalasi Lingkungan.....	139
2.1 Tujuan bab.....	139
2.2 Persiapan Lingkungan.....	141
2.2.1 Memilih Sistem Operasi.....	141
2.2.2 Memastikan Kebutuhan Sistem.....	143
2.3 Instalasi Python dan Virtual Environment.....	146
2.3.1 Instalasi Python.....	146
2.3.2 Mengatur Virtual Environment (Opsional tetapi Disarankan):.....	148
2.4 Instalasi Django.....	151
2.4.1 Menggunakan pip untuk Instalasi Django.....	151
2.4.2 Membuat Proyek Django Pertama.....	153
2.5 Instalasi dan Konfigurasi Ngrok.....	155
2.5.1 Mengunduh dan Menginstal Ngrok.....	156
2.5.2 Konfigurasi dan Menjalankan Ngrok.....	157
2.5.3 Mengatur Ngrok untuk Webhook.....	159
2.6 Instalasi dan Konfigurasi Bot Framework.....	162
2.6.1 Memilih Framework Bot.....	163
2.6.2 Instalasi dan Konfigurasi Bot Framework.....	166

2.7 Instalasi dan Penggunaan Code Editor.....	167
2.7.1 Memilih Code Editor.....	168
2.7.2 Vim.....	168
2.7.3 Instalasi Visual Studio Code.....	172
2.8 Menyiapkan Browser dan Terminal/Command Line.....	175
2.8.1 Browser.....	175
2.8.2 Terminal/Command Line.....	177
2.9 Verifikasi Lingkungan.....	179
2.9.1 Menguji Instalasi Django.....	179
2.9.2 Menguji Ngrok dan Webhook.....	182
Kesimpulan Bab.....	186
BAB 3 - Memahami dan Menggunakan Django Admin	
.....	189
3.1 Apa itu Django Admin?.....	190
3.2 Fitur Utama Django Admin.....	191
3.3 Membuat Superuser.....	194
3.3.1 Apa Itu Superuser?.....	194
3.3.2 Langkah-Langkah Membuat Superuser.....	196
3.3.3 Login ke Django Admin dengan Superuser.....	198
3.4 Menambahkan Model ke Django Admin.....	202
3.4.1 Mendaftarkan Model ke Django Admin.....	202
3.4.2 Personalisasi Tampilan Model di Admin.....	204
3.5 Personalisasi Django Admin.....	206
3.5.1 Mengubah Tampilan Django Admin.....	207
3.5.2 Menambahkan Aksi Khusus di Django Admin....	210
3.6 Fitur-Fitur Tambahan Django Admin.....	213
3.6.1 Inline Forms di Django Admin.....	213
3.6.2 Membuat Filter Kustom.....	216
3.6.3 Export Data dari Django Admin.....	219
3.7 Keamanan dalam Django Admin.....	222
3.7.1 Mengelola Izin Pengguna (User Permissions).....	223
3.7.2 Membatasi Akses ke Django Admin.....	225
3.8 Mengatasi Masalah Umum di Django Admin.....	227
3.8.1 Masalah Umum saat Mengakses Django Admin..	228
3.8.2 Debugging dan Solusi.....	230
Kesimpulan Bab.....	234
BAB 4 - Memulai Awal proyek.....	237

4.1	Pembuatan Proyek Django.....	237
4.1.1	Membuat Folder `apps`.....	238
4.1.2	Memperbaiki apps.py.....	242
4.1.3	Memahami Struktur Dasar Aplikasi.....	245
4.1.4	Mengelola Static Files.....	249
4.1.5	Menyusun File Templates di Folder apps.....	253
4.1.6	Membuat Context Processor.....	256
4.1.7	Pengaturan Media di settings.py.....	260
4.2	Membuat Halaman Home (Landing Page).....	261
4.2.1	Tujuan Halaman Home.....	261
4.2.2	Membuat View untuk Halaman Home.....	262
4.2.3	Membuat Template untuk Halaman Home.....	264
4.2.4	Menyusun Layout Halaman Home.....	266
4.2.5	Menambahkan Navigasi dan Elemen Interaktif... ..	272
4.2.6	Mengonfigurasi URL untuk Halaman Home.....	277
4.2.7	Menguji Halaman Home.....	279
	Kesimpulan Bab.....	281
BAB 5 - Fitur Registrasi, Login, dan Logout.....		283
5.1	Membuat Fitur Registrasi.....	283
5.1.1	Membuat Model User.....	283
5.1.2	Membuat Form Registrasi.....	289
5.1.3	Membuat View untuk Registrasi.....	292
5.1.4	Menyusun Template Registrasi.....	295
5.1.5	Menambahkan Routing URLS Registrasi.....	300
5.1.6	Testing Fitur Registrasi (opsional).....	301
5.2	Membuat Fitur Login.....	306
5.2.1	Membuat Form Login.....	306
5.2.2	Membuat View untuk Login.....	308
5.2.3	Membuat Template Login.....	311
5.2.4	Menambahkan Routing URLS Login.....	314
5.2.5	Testing Fitur Login (opsional).....	315
5.3	Membuat Fitur Logout.....	320
5.3.1	Mengatur View untuk Logout.....	320
5.3.2	Mengatur URL Routing Logout.....	322
5.3.3	Mengatur Redirect Setelah Logout.....	324
5.3.4	Testing Fitur Logout (Opsional).....	326
5.4	Kode Lengkap.....	329

5.4.1 Models.py.....	330
5.4.2 Forms.py.....	330
5.4.3 Views.py.....	333
5.4.4 URLS.py.....	336
5.5 Menghubungkan Halaman Home dengan App Authentication.....	337
Kesimpulan Bab.....	342
BAB 6 - Membuat Halaman Dashboard User.....	345
6.1 Memahami Tujuan dan Manfaat Dashboard.....	345
6.2 Membuat Model Tambahan untuk Informasi Dashboard	346
6.3 Membuat View Dashboard.....	350
6.4 Membuat Template Halaman Dashboard.....	353
6.4.1 Membuat Template base.html.....	354
6.4.2 Membuat halaman dashboard.html.....	357
6.4.3 Membuat Sidebar dengan Ikon dan Responsif Menggunakan Bootstrap.....	359
6.5 Menyiapkan URL Routing untuk Dashboard.....	367
6.6 Mengatur Redirect Otomatis ke Dashboard Setelah Login	368
Kesimpulan Bab.....	372
BAB 7 - Manajemen Profil Pengguna.....	375
7.1 Membuat halaman Profile pengguna.....	375
7.1.1 Membuat View untuk Halaman Profil Pengguna.	375
7.1.2 Membuat Template Halaman Profil.....	377
7.1.3 Menambahkan URL Routing.....	379
7.2 Membuat Fitur Pengeditan Profile.....	380
7.2.1 Membuat Form untuk mengedit Profil Pengguna.	380
7.2.2 Memperbarui Views Profile.....	384
7.2.3 Memperbarui Template Halaman Profil.....	385
7.2.4 Menguji Halaman Profil Pengguna.....	391
7.3 Integrasi dengan model log aktivitas.....	392
7.3.1 Memperbaiki Template untuk Menampilkan Log Aktivitas di Dashboard.....	394
7.4 Kode Lengkap.....	397
7.4.1 Forms.py.....	397
7.4.2 Views.py.....	400

7.4.3 Profile.html.....	401
7.4.4 URLS.py.....	406
Kesimpulan Bab.....	407
BAB 8 - Pengenalan Bot dan Menyiapkan Halaman	
Dasar.....	409
8.1 Menyiapkan Model dan Formulir untuk Bot.....	409
8.1.1 Menyiapkan Model Bot.....	410
8.1.2 Menyiapkan Formulir untuk Bot.....	416
8.2 Menyiapkan Handle Bot.....	421
8.2.1 Handle Telegram.....	422
8.2.2 Handle Whatsapp.....	428
8.2.3 Fungsi Penanganan Platform.....	432
8.3 Menyiapkan Halaman untuk membuat Bot.....	433
8.3.1 Membuat View untuk Membuat Bot.....	434
8.3.2 Membuat Template untuk Membuat Bot.....	438
8.3.3 Mengatur URL Routing untuk Membuat Bot.....	445
8.4 Membuat Halaman untuk manage Bot.....	447
8.4.1 Membuat View untuk Manage Bot.....	447
8.4.2 Membuat Template untuk Manage Bot.....	451
8.4.3 Mengatur URL Routing untuk Manage Bot.....	456
8.5 Kode lengkap.....	457
8.5.1 Models.py.....	457
8.5.2 Forms.py.....	459
8.5.3 Views.py.....	460
8.5.4 URLS.py.....	462
Kesimpulan Bab.....	463
BAB 9 - Validasi Token untuk Bot.....	465
9.1 Pendahuluan.....	465
9.1.1 Pentingnya Validasi Token.....	467
9.1.2 Jenis-jenis Token dan Penggunaannya.....	471
9.2 Memahami Format Token.....	477
9.2.1 Format Token Telegram.....	478
9.2.2 Format Token WhatsApp.....	483
9.2.3 Mengapa Validasi Ini Penting?.....	485
9.2.4 Format Token untuk Platform Lain.....	489
9.2.5 Mengapa Format Token Itu Penting?.....	496
9.3 Implementasi Validasi Token di Proyek.....	498

9.3.1 Menerapkan Validasi di Model.....	498
9.3.2 Mengaitkan Token dengan Bot.....	499
9.4 Menerapkan Validasi di Form.....	505
9.4.1 Validasi di Level Form.....	505
9.5 Menggunakan Validators Kustom.....	509
9.6 Menangani Kesalahan Input Token.....	514
9.6.1 Menampilkan Pesan Kesalahan yang Jelas.....	514
9.6.2 Menggunakan Bootstrap untuk Notifikasi Kesalahan	516
9.7 Kode lengkap setelah di Validasi.....	519
9.7.1 Models.py.....	522
9.7.2 Forms.py.....	525
9.7.3 Create_bot.html.....	527
Kesimpulan Bab.....	534
BAB 10 - Menyiapkan Webhook untuk Bot.....	537
10.1 Pengantar Webhook.....	537
10.1.1 Apa Itu Webhook?.....	538
10.1.2 Keuntungan Menggunakan Webhook.....	539
10.1.3 Cara Kerja Webhook.....	539
10.1.4 Menyiapkan Webhook dalam Django.....	540
10.2 Menyusun dan Mengelola Webhook.....	541
10.2.1 Menangani Data dari Webhook.....	541
10.3 Menyiapkan Webhook Telegram.....	542
10.3.1 Mengonfigurasi Webhook untuk Telegram.....	543
10.3.2 Mengatur Webhook Telegram.....	545
10.4 Menyiapkan Webhook Whatsapp.....	546
10.4.1 Mengonfigurasi Webhook untuk WhatsApp.....	547
10.4.2 Mengatur Webhook WhatsApp melalui Twilio. .	549
10.4.3 Menghubungkan URL Webhook ke Twilio.....	550
10.4.4 Menjalankan dan Menguji Webhook.....	551
10.5 Kode lengkap views.py.....	551
10.6 Mengatur Routing URL.....	554
10.7 Konfigurasi settings.py.....	555
Kesimpulan Bab.....	559
BAB 11 - Integrasi Bot dengan Webhook.....	561
11.1 Pengantar Integrasi Webhook dengan Bot.....	561
11.1.1 Apa Itu Integrasi Webhook?.....	562

11.1.2 Pentingnya Pengujian dalam Integrasi Webhook	564
11.2 Menyiapkan Integrasi Webhook untuk Bot	568
11.3 Implementasi Integrasi Webhook dan Bot	570
11.3.1 Integrasi di View apps/bots	571
11.3.2 Memperbaiki Halaman Manage Bot	575
11.4 Pengujian Membuat Bot	577
11.4.1 Pengujian Membuat Bot Telegram	578
11.4.2 Pengujian Membuat Bot Whatsapp	581
Kesimpulan Bab	585
BAB 12 - Membuat Fitur untuk Bot	587
12.1 Pengantar Fitur Command pada Bot	588
12.1.1 Apa Itu Command pada Bot?	588
12.1.2 Pentingnya Command dalam Interaksi Bot	590
12.1.3 Proses Menambahkan dan Mengedit Command	592
12.2 Menyiapkan Model dan Forms	595
12.2.1 Membuat Model untuk Command	596
12.2.2 Membuat forms untuk Command	599
12.3 Menyiapkan Fitur untuk Edit Bot	602
12.3.1 Membuat View untuk Edit Bot	602
12.3.2 Membuat Template untuk Edit Bot	604
12.3.3 Mengatur URL Routing untuk Fitur Edit Bot	607
12.4 Menyiapkan Fitur untuk Tambah Command Bot	608
12.4.1 Membuat View untuk Menambah Command	608
12.4.2 Membuat Template untuk Menambah Command	611
12.4.3 Mengatur URL Routing untuk Fitur Tambah Command	614
12.4.4 Menampilkan Daftar Command	616
12.4.5 Memperbaiki Handle update	619
12.5 Menyiapkan Fitur untuk Mengedit Command	623
12.5.1 Membuat View untuk Mengedit Command	624
12.5.2 Membuat Template untuk Mengedit Command	626
12.5.3 Mengatur URL Routing untuk Fitur Edit Command	629
12.6 Integrasi dengan Model Log Aktivitas	630
12.6.1 Integrasi Untuk Create Bot	631
12.6.2 Integrasi Untuk Edit Bot	633

12.6.3 Integrasi Untuk Hapus Bot.....	635
12.6.4 Integrasi Untuk Add Command.....	638
12.6.5 Integrasi Untuk Edit Command.....	641
12.6.6 Integrasi Untuk Hapus Command.....	644
12.7 Kode Lengkap untuk Bots.....	645
12.7.1 Models.py.....	645
12.7.2 Forms.py.....	647
12.7.3 Views.py.....	650
12.7.4 Services/telegram.py.....	656
12.7.5 Services/whatsapp.py.....	658
12.7.6 URLS.py.....	660
Kesimpulan Bab.....	661
BAB 13 - Penyempurnaan Halaman Home.....	665
13.1 Pendahuluan.....	665
13.1.1 Tujuan dan Manfaat Penggunaan Bootstrap.....	665
13.1.2 Pengenalan Dasar Bootstrap.....	666
13.1.3 Instalasi dan Konfigurasi Bootstrap di Proyek Django.....	667
13.2 Penyempurnaan Halaman Home.....	669
13.2.1 Perbaiki base.html.....	670
13.2.2 Menambahkan Komponen Bootstrap.....	675
13.2.3 Menyempurnakan Tampilan Responsif.....	677
13.2.4 Menambahkan Konten di Halaman Home.....	679
13.2.5 Menguji Tampilan Halaman Home.....	685
13.3 Penyempurnaan Halaman register.....	686
13.3.1 Mengintegrasikan Bootstrap Grid System.....	686
13.3.2 Menambahkan Komponen Bootstrap.....	687
13.3.3 Menguji Tampilan Halaman Register.....	695
13.4 Penyempurnaan Halaman Login.....	695
13.4.1 Memahami Struktur Halaman Login Saat Ini.....	696
13.4.2 Mengintegrasikan Bootstrap Grid System.....	696
Kesimpulan Bab.....	702
BAB 14 - Penyempurnaan Halaman Dashboard.....	705
14.1 Memisahkan Struktur Direktori Template.....	706
14.1.1 Memisahkan base.....	707
14.1.2 Memisahkan navbar.....	709
14.1.3 Memisahkan sidebar.....	709

14.1.4	Memisahkan dashboard.....	711
14.2	Memperbaiki Halaman.....	713
14.2.1	Menyiapkan css dan js untuk Dashboard.....	714
14.2.2	Memperbaiki Template base.html untuk Dashboard	723
14.2.3	Memperbaiki Sidebar untuk Dashboard.....	728
14.2.4	Memperbaiki Navbar untuk Dashboard.....	732
14.2.5	Memperbaiki Halaman Dashboard.....	736
14.3	Penyempurnaan Halaman Profile.....	746
	Kesimpulan Bab.....	753
BAB 15	- Penyempurnaan Halaman Bots.....	757
15.1	Memperbaiki Halaman Create Bot.....	758
15.2	Memperbaiki Halaman Manage Bots.....	766
15.3	Memperbaiki Halaman Edit Bot.....	773
15.4	Memperbaiki Halaman Add command.....	778
15.5	Memperbaiki halaman Edit command.....	782
	Kesimpulan Bab.....	787
BAB 16	- Menginstall Project ke GitHub dan Deploy Django.....	791
16.1	Pendahuluan.....	791
16.2	Mengupload Project ke GitHub.....	792
16.2.1	Membuat Repository di GitHub.....	793
16.2.2	Mengatur Personal Access Token (PAT) di GitHub	794
16.2.3	Menghubungkan Local Repository dengan GitHub	798
16.3	Menyiapkan Server untuk Deployment.....	802
16.3.1	Memilih Platform untuk Deployment.....	802
16.3.2	Memilih Platform Berdasarkan Kebutuhan.....	805
16.3.3	Rekomendasi Platform.....	806
16.4	Deploy Ke PythonAnywhere.....	806
16.4.1	Meng-clone Repository dari GitHub.....	807
16.4.2	Mengatur Aplikasi Web di PythonAnywhere....	807
16.4.3	Mengonfigurasi Static Files dan Templates.....	808
16.4.4	Mengatur File WSGI.....	809
16.4.5	Reload Aplikasi.....	809
16.5	Deploy ke Platform Heroku.....	810

16.5.1 Instalasi dan Setup Heroku CLI.....	810
16.5.2 Mengatur Procfile dan requirements.txt.....	812
16.5.3 Deploy ke Heroku.....	814
16.5.4 Menangani Environment Variables.....	815
16.6 Mendapatkan Webhook URL.....	818
16.6.1 Cara Mendapatkan Webhook di Heroku.....	818
16.6.2 Mengaktifkan Webhook untuk Bot.....	820
Kesimpulan Bab.....	823
PENUTUP.....	dcccxxv
Harapan untuk Pembaca.....	dcccxxvi
Struktur Akhir Proyek.....	dcccxxvii
Penjelasan Setiap Bagian.....	dcccxxx
Link Proyek GitHub.....	dcccxxxi
UCAPAN TERIMA KASIH.....	dcccxxxiii

PENDAHULUAN

Selamat Datang di Dunia Pengembangan Bot

Selamat datang di eBook yang akan membawa Anda melalui perjalanan yang mendalam dan memuaskan dalam dunia pengembangan bot. Dalam beberapa tahun terakhir, bot telah menjadi alat yang semakin penting dalam berbagai bidang, mulai dari layanan pelanggan hingga otomatisasi bisnis. Mereka tidak hanya mempermudah tugas sehari-hari tetapi juga membuka peluang baru bagi individu dan perusahaan untuk berinteraksi dengan pengguna mereka dengan cara yang lebih efisien dan personal.

Namun, membangun sebuah platform bot yang kuat bukanlah tugas yang mudah. Banyak pengembang pemula yang merasa kewalahan dengan kerumitan teknologi yang terlibat. Di sinilah Django, sebuah framework web yang powerful dan fleksibel, masuk untuk memberikan solusi yang elegan dan efisien. Django memungkinkan Anda untuk membangun aplikasi web yang kompleks dan dapat diskalakan dengan cepat, tanpa mengorbankan kualitas atau performa.

Mengapa Django?

Django bukan hanya salah satu framework web yang paling populer, tetapi juga salah satu yang paling andal. Dengan arsitektur

yang jelas dan dokumentasi yang mendalam, Django dirancang untuk membantu pengembang dari berbagai tingkat keahlian—dari pemula hingga ahli—untuk membangun aplikasi web yang aman, scalable, dan maintainable. Django digunakan oleh beberapa perusahaan teknologi terbesar di dunia, termasuk Instagram dan Pinterest, untuk alasan yang sangat baik: ini adalah alat yang memungkinkan ide besar diwujudkan dengan cepat dan efektif.

Dalam buku ini, Anda akan belajar cara membangun platform bot dari awal hingga akhir menggunakan Django. Panduan langkah demi langkah ini akan membantu Anda memahami tidak hanya bagaimana Django bekerja, tetapi juga bagaimana Anda dapat memanfaatkan kekuatannya untuk membuat solusi yang benar-benar berguna dan bernilai.

Apa itu Bot?

Bot, atau robot perangkat lunak, adalah program yang dirancang untuk mengotomatisasi tugas-tugas tertentu dalam sebuah aplikasi atau di internet. Bot sering digunakan untuk menjalankan tugas-tugas rutin yang dapat diprogram sebelumnya, seperti merespons pertanyaan, mengelola interaksi pengguna, atau memproses data secara otomatis. Bot ini sangat bermanfaat dalam berbagai skenario, terutama untuk interaksi cepat dan efisien dengan pengguna tanpa memerlukan intervensi manusia secara langsung.

Dalam konteks aplikasi web, bot sering kali berfungsi sebagai **chatbot** yang berinteraksi secara otomatis dengan pengguna melalui antarmuka chat, seperti di Telegram atau WhatsApp. Selain itu, bot juga bisa beroperasi di belakang layar untuk mengelola dan memproses data secara otomatis tanpa interaksi langsung dengan pengguna.

Contoh Bot: Telegram dan WhatsApp

Bot Telegram

Bot Telegram adalah salah satu jenis bot yang paling populer saat ini. Telegram menyediakan platform yang fleksibel bagi pengembang untuk membuat bot yang dapat berinteraksi dengan pengguna melalui percakapan chat. Bot Telegram bisa digunakan untuk berbagai keperluan, seperti:

- ***Chatbot Pelayanan:*** Bot yang merespons pertanyaan pelanggan secara otomatis, memberikan informasi produk, atau membantu pengguna menyelesaikan masalah.
- ***Pengingat dan Notifikasi:*** Bot yang mengirimkan pengingat kepada pengguna tentang suatu acara, tugas, atau peristiwa penting, serta mengirimkan notifikasi otomatis.
- ***Automatisasi Tugas:*** Bot yang menjalankan tugas tertentu berdasarkan instruksi dari pengguna, seperti memantau harga produk, memberikan laporan cuaca, atau menjalankan proses terjadwal.

Contoh nyata adalah bot yang dapat mengelola grup Telegram, memoderasi percakapan, menghapus pesan spam, atau bahkan memberikan kuis kepada pengguna.

Bot WhatsApp

Bot WhatsApp, di sisi lain, juga sangat bermanfaat bagi bisnis dan organisasi untuk berinteraksi dengan pengguna melalui platform WhatsApp. Salah satu penyedia layanan populer untuk membangun bot di WhatsApp adalah **Twilio**, yang memungkinkan pengembang untuk membuat bot yang terhubung langsung dengan nomor WhatsApp mereka. Fungsionalitas dari bot WhatsApp mencakup:

- ***Dukungan Pelanggan:*** Bot yang merespons pertanyaan umum pelanggan, memberikan status pesanan, atau memandu pengguna melalui berbagai proses langsung di WhatsApp.
- ***Pemberitahuan dan Konfirmasi:*** Bot yang mengirimkan pemberitahuan seperti konfirmasi pesanan, status pengiriman, atau pengingat janji temu.
- ***Pemesanan dan Pembayaran:*** Beberapa bisnis menggunakan bot WhatsApp untuk menerima pesanan dan memproses pembayaran secara otomatis, memberikan pengalaman belanja yang lebih cepat dan mudah bagi pengguna.

Misalnya, bot pada WhatsApp dapat digunakan oleh toko online untuk menerima pesanan dan memberikan update otomatis kepada pelanggan tentang status pengiriman mereka.

Kegunaan Bot dalam Aplikasi Web

1. ***Interaksi Pengguna:*** Bot dapat memberikan respons otomatis kepada pengguna, seperti menjawab pertanyaan yang sering diajukan (FAQ), memberikan panduan, atau menyampaikan informasi penting secara cepat dan efisien.
2. ***Automatisasi Tugas:*** Bot dapat menjalankan tugas-tugas berulang, seperti mengirim email, memberikan pengingat, atau memproses transaksi, tanpa memerlukan campur tangan manusia. Ini meningkatkan efisiensi operasional.
3. ***Pengolahan Data:*** Bot dapat mengumpulkan, menganalisis, dan menyajikan data dari berbagai sumber, membantu pengambilan keputusan yang lebih baik. Misalnya,

bot dapat memantau tren penjualan dan memberikan laporan berkala kepada pengguna.

Gambaran Umum Proyek: Platform Bot

Proyek ini bertujuan untuk membangun platform yang memungkinkan pengguna membuat dan mengelola bot mereka secara mandiri melalui antarmuka web yang sederhana namun kuat. Platform ini akan mencakup beberapa fitur utama sebagai berikut:

1. **Halaman Home (Landing Page):** Halaman utama yang menyambut pengguna baru dan menyediakan opsi untuk login atau registrasi. Halaman ini juga memberikan gambaran singkat tentang platform dan manfaatnya.
2. **Fitur Registrasi dan Login:** Sistem autentikasi yang memungkinkan pengguna untuk membuat akun, login, dan mengakses dashboard pribadi mereka. Proses ini dilengkapi dengan validasi keamanan untuk memastikan hanya pengguna yang sah dapat mengakses platform.
3. **Dashboard Pengelolaan Bot:** Setelah login, pengguna akan diarahkan ke dashboard yang berfungsi sebagai pusat kendali. Di sini, mereka dapat melihat informasi umum tentang bot mereka, statistik, dan aktivitas terbaru.
4. **Create Bot:** Fitur yang memungkinkan pengguna untuk membuat bot baru. Pengguna dapat memilih platform (misalnya, Telegram atau WhatsApp), lalu memasukkan token atau informasi spesifik platform lainnya. Proses pembuatan bot dibuat mudah dengan antarmuka yang interaktif dan instruksi yang jelas.
5. **Manage Bot:** Fitur ini memungkinkan pengguna untuk melihat daftar semua bot yang telah mereka buat. Mereka bisa melihat detail bot, status bot (aktif/tidak aktif), serta

melakukan tindakan seperti mengedit atau menghapus bot.

6. **Edit Bot:** Pengguna dapat mengedit informasi bot yang telah dibuat, seperti nama bot, token, atau pengaturan lainnya yang berkaitan dengan platform yang dipilih. Setiap perubahan akan disimpan dan diperbarui secara real-time.
7. **Add Command:** Fitur ini memungkinkan pengguna untuk menambahkan perintah (command) baru ke bot mereka. Perintah ini bisa berupa respons otomatis yang akan dijalankan ketika pengguna tertentu mengirimkan pesan atau instruksi tertentu.
8. **Edit Command:** Selain menambahkan perintah, pengguna juga dapat mengedit perintah yang sudah ada. Ini memungkinkan mereka untuk memperbarui respons atau logika di balik setiap perintah sesuai kebutuhan.
9. **Edit Profile:** Pengguna juga memiliki opsi untuk memperbarui informasi profil mereka, seperti nama, alamat email, dan kata sandi. Fitur ini memastikan pengguna memiliki kendali penuh atas informasi pribadi mereka.
10. **Webhook dan Integrasi Bot:** Platform ini juga memungkinkan integrasi dengan platform eksternal melalui webhook. Fitur ini memungkinkan bot untuk menerima dan merespons data secara real-time, memastikan interaksi yang mulus antara bot dan aplikasi eksternal.

Dengan memanfaatkan Django sebagai framework inti, proyek ini dirancang untuk menyediakan solusi yang fleksibel, aman, dan mudah digunakan. Django akan menangani manajemen data dan sistem autentikasi, sementara kekuatan Python digunakan untuk mengelola logika bot dan integrasi webhook. Platform ini

menawarkan skalabilitas tinggi, memungkinkan pengembang untuk menambahkan lebih banyak fitur dan platform bot di masa depan.

Tentang Buku Ini

Buku ini dirancang khusus untuk membantu Anda membangun **platform pembuatan bot** yang canggih menggunakan framework Django. Dalam era digital saat ini, bot telah menjadi alat yang sangat penting dalam berbagai industri, mulai dari layanan pelanggan, pemasaran, hingga otomatisasi proses bisnis. Dengan berkembangnya kebutuhan akan solusi yang lebih efisien dan terukur, membangun platform bot yang fleksibel dan skalabel menjadi semakin relevan. Buku ini akan memandu Anda melalui setiap langkah proses pengembangan platform tersebut, mulai dari tahap perencanaan hingga implementasi akhir.

Tujuan Buku Ini

Tujuan utama dari buku ini adalah untuk memberikan **panduan praktis** dan **mudah dipahami**, bahkan bagi mereka yang baru mengenal Django atau pengembangan web secara umum. Anda akan dibimbing langkah demi langkah dalam membangun aplikasi web berbasis Django yang dapat digunakan untuk membuat, mengelola, dan menjalankan bot dengan menggunakan **webhook**. Selain itu, buku ini juga dirancang agar Anda dapat mengembangkan pemahaman mendalam tentang berbagai aspek pengembangan aplikasi berbasis Django, mulai dari manajemen database hingga antarmuka pengguna yang interaktif.

Setiap bab dalam buku ini difokuskan pada bagian penting dari pengembangan platform bot, dengan instruksi yang dirancang untuk membawa Anda dari tingkat dasar hingga proyek yang da-

pat digunakan dalam produksi. Anda tidak hanya akan mempelajari teknik-teknik koding, tetapi juga prinsip-prinsip pengembangan perangkat lunak yang baik, seperti manajemen versi, pengujian, dan implementasi keamanan.

Siapa yang Harus Membaca Buku Ini?

Buku ini cocok untuk **pemula** yang ingin belajar Django sambil membangun aplikasi nyata. Tidak hanya membahas dasar-dasar Django, buku ini juga menyediakan ***kasus penggunaan praktis*** yang akan membantu Anda memahami bagaimana Django bisa digunakan untuk membangun aplikasi yang kompleks dan fungsional. Meskipun fokus utama buku ini adalah pada pengembangan platform bot, konsep dan keterampilan yang dipelajari dapat dengan mudah diterapkan ke proyek Django lainnya.

Buku ini dirancang untuk pengembang pemula yang ingin memperdalam pengetahuan mereka tentang Django dan bagaimana framework ini dapat digunakan untuk membangun platform bot. Namun, ini juga cocok untuk pengembang yang sudah berpengalaman tetapi ingin mempelajari cara mengintegrasikan bot ke dalam aplikasi web mereka.

Buku ini ditargetkan untuk:

- ***Pemula Django***: Orang-orang yang baru memulai dengan Django dan ingin memahami cara kerja framework ini melalui proyek nyata.
- ***Pengembang yang ingin belajar secara praktis***: Mereka yang lebih suka belajar dengan langsung terjun ke dalam proyek, bukan hanya membaca teori.
- ***Orang yang tertarik membuat aplikasi berbasis bot***: Baik untuk kebutuhan pribadi maupun profesional.

Anda tidak perlu menjadi ahli sebelum memulai perjalanan ini. Yang Anda butuhkan hanyalah pengetahuan dasar tentang Python dan semangat untuk belajar. Langkah demi langkah, kita akan melalui setiap aspek pengembangan platform bot, mulai dari pengaturan lingkungan hingga deployment di server produksi.

Jika Anda seorang pengembang web pemula atau seseorang yang ingin memperluas keterampilan Anda dalam **pengembangan bot** dan **integrasi webhook**, buku ini adalah pilihan yang tepat. Anda tidak memerlukan pengalaman mendalam dengan Django atau pengembangan bot sebelumnya, karena buku ini akan membahas semua hal dari dasar hingga tahap lanjutan dengan pendekatan yang ramah pengguna.

Pendekatan Buku Ini

Buku ini menggunakan pendekatan **langsung** dan **berbasis proyek**. Setiap bab mencakup komponen penting dari proses pengembangan, dimulai dari penyiapan lingkungan kerja, struktur proyek Django, hingga membangun fitur-fitur utama seperti halaman pembuatan bot, manajemen bot, hingga integrasi webhook. Anda akan belajar bagaimana cara menggunakan **Bootstrap** untuk mempercantik tampilan aplikasi, serta bagaimana mengelola **model database** untuk menyimpan informasi bot dan pengguna.

Selain itu, buku ini juga mengajarkan prinsip-prinsip pengembangan perangkat lunak yang baik, termasuk manajemen dependensi, pengujian, dan **keamanan aplikasi web**. Anda akan dibimbing untuk memahami bagaimana menjaga keamanan token bot, bagaimana menggunakan Django untuk mengelola autentikasi pengguna, serta cara menghindari kesalahan umum dalam pe-

ngembangan bot yang berinteraksi dengan platform eksternal seperti Telegram dan WhatsApp.

Setiap fitur yang dibangun akan dijelaskan secara rinci, dengan kode yang disertai penjelasan agar Anda memahami setiap langkah yang diambil. Dengan pendekatan ini, Anda tidak hanya akan belajar cara membuat aplikasi yang berfungsi, tetapi juga memahami ***konsep di baliknya*** sehingga Anda bisa menerapkan pengetahuan ini pada proyek-proyek lain di masa depan.

Konten yang Disajikan dalam Buku Ini

Buku ini terdiri dari beberapa bab yang berfokus pada topik-topik kunci, di antaranya:

- ***Pengenalan Django dan Persiapan Proyek:*** Membahas dasar-dasar Django, penyiapan lingkungan kerja, dan struktur proyek.
- ***Membangun Halaman Pembuatan Bot:*** Mengajarkan cara membuat antarmuka pengguna yang interaktif untuk membuat dan mengelola bot.
- ***Manajemen Bot dan Komando:*** Memperkenalkan fitur-fitur manajemen bot seperti edit, hapus, dan tambahkan perintah ke dalam bot yang dibuat.
- ***Integrasi Webhook:*** Membahas cara menghubungkan platform bot dengan platform eksternal seperti Telegram dan WhatsApp menggunakan webhook.
- ***Keamanan dan Validasi Token:*** Membahas aspek-aspek keamanan dalam pengembangan bot, termasuk cara melindungi token dan memastikan hanya token valid yang dapat digunakan.

- **Testing dan Deployment:** Panduan tentang cara menguji aplikasi secara efektif dan mengimplementasikannya ke dalam server produksi.

Apa yang Akan Anda Dapatkan Setelah Membaca Buku Ini?

Setelah menyelesaikan buku ini, Anda akan memiliki **pemahaman yang mendalam** tentang cara membangun aplikasi Django yang fungsional dan kompleks. Tidak hanya terbatas pada pembuatan bot, keterampilan yang Anda pelajari akan mencakup berbagai aspek pengembangan web, seperti pengelolaan database, desain antarmuka pengguna yang interaktif, integrasi API, dan penerapan keamanan dalam aplikasi web.

Selain itu, Anda akan memiliki platform bot yang sepenuhnya berfungsi, yang bisa Anda kembangkan lebih lanjut atau gunakan sebagai dasar untuk proyek-proyek masa depan. Anda juga akan terbiasa dengan praktik terbaik dalam pengembangan perangkat lunak, sehingga meningkatkan kemampuan Anda sebagai pengembang yang lebih profesional dan berpengalaman.

Apa yang Akan Kita Bangun

Dalam proyek akhir ini, kita akan membangun **sebuah platform pembuatan dan pengelolaan bot** yang lengkap dan canggih, memungkinkan pengguna untuk dengan mudah membuat, mengelola, dan memantau bot mereka melalui antarmuka yang ramah pengguna. Platform ini tidak hanya menyediakan alat untuk pembuatan bot, tetapi juga menawarkan kontrol penuh melalui **panel admin** yang kuat dan **dashboard pengguna** yang interaktif. Dengan memanfaatkan Django sebagai framework backend, proyek ini akan mengintegrasikan beberapa layanan bot seperti

Telegram dan **WhatsApp**, serta memungkinkan pengembangan platform di masa mendatang.

Platform ini akan berfokus pada keamanan, skalabilitas, dan kemudahan penggunaan, sehingga memungkinkan pengguna untuk berinteraksi dengan bot dan mengelolanya tanpa memerlukan pengetahuan teknis mendalam. Berikut adalah rincian fitur utama dari proyek ini:

1. Halaman Registrasi dan Login

Pengguna perlu mendaftar dan masuk ke platform menggunakan **halaman registrasi dan login** yang aman dan intuitif. Fitur-fitur ini akan dilengkapi dengan lapisan keamanan tambahan untuk memastikan hanya pengguna yang sah yang dapat mengakses platform. Fitur yang disertakan meliputi:

- **Formulir pendaftaran** untuk pengguna baru, dengan validasi data yang ketat untuk memastikan informasi yang dimasukkan akurat dan aman.
- **Formulir login** untuk pengguna yang telah terdaftar, dengan mekanisme autentikasi yang aman untuk mencegah akses tidak sah.
- **Fitur pemulihan kata sandi**, memungkinkan pengguna yang lupa kata sandi mereka untuk mereset melalui email verifikasi.
- **Verifikasi email** untuk memastikan akun yang dibuat sah, dan memastikan keamanan tambahan saat pengguna masuk.

2. Halaman Dashboard Pengguna

Setelah berhasil masuk, pengguna akan diarahkan ke **halaman dashboard pribadi** mereka. Dashboard ini berfungsi sebagai pu-

sat kendali, di mana pengguna dapat mengelola semua aspek bot mereka dengan mudah. Fitur-fitur utama di halaman dashboard meliputi:

- **Pengelolaan Bot:** Pengguna dapat melihat daftar bot yang telah mereka buat, menambah bot baru, atau mengedit dan menghapus bot yang ada. Setiap bot akan ditampilkan dengan informasi dasar seperti nama, platform, dan statusnya (aktif atau non-aktif).
- **Pilihan Template Bot:** Platform menyediakan berbagai **template bot siap pakai**, yang memungkinkan pengguna untuk memulai dengan cepat tanpa perlu menulis kode dari awal. Pengguna dapat memilih dan menyesuaikan template bot sesuai dengan kebutuhan mereka.
- **Statistik Bot:** Dashboard akan menampilkan statistik penting terkait performa bot, termasuk jumlah pengguna yang berinteraksi, pesan yang dikirim, dan laporan error. Informasi ini ditampilkan dalam bentuk grafik dan tabel untuk memudahkan analisis.
- **Fitur Tambahan:** Pengguna juga bisa mengelola **komando bot**, seperti menambahkan perintah baru, mengedit perintah yang sudah ada, atau menghapusnya. Semua ini dapat dilakukan melalui antarmuka yang intuitif.

3. Panel Admin

Platform ini akan dilengkapi dengan **panel admin** yang kuat untuk memudahkan administrator dalam mengelola pengguna, bot, dan fitur lainnya. Panel admin berfungsi sebagai pusat pengelolaan platform secara keseluruhan, dengan akses eksklusif bagi administrator untuk mengatur berbagai elemen penting. Fitur yang disediakan di panel admin meliputi:

- **Manajemen Pengguna:** Administrator dapat menambah, mengedit, atau menghapus pengguna yang terdaftar di

platform. Selain itu, admin dapat melihat **aktivitas pengguna**, seperti kapan terakhir kali mereka login, atau aktivitas apa saja yang dilakukan.

- **Manajemen Hak Akses:** Admin memiliki kendali penuh atas hak akses pengguna. Mereka dapat memberikan hak istimewa kepada pengguna tertentu, seperti akses admin atau moderator.
- **Manajemen Bot:** Admin bisa memantau semua bot yang dibuat oleh pengguna, serta menambahkan fitur tambahan ke bot tertentu. Admin juga memiliki kemampuan untuk menghapus bot yang melanggar kebijakan platform.
- **Manajemen Template Bot:** Admin dapat menambah, mengedit, atau menghapus template bot yang tersedia di platform. Ini memungkinkan admin untuk menambah opsi baru yang dapat dipilih oleh pengguna ketika membuat bot.

4. Panel Pengguna

Panel pengguna dirancang untuk memberikan **pengalaman pengguna yang optimal** dan intuitif, memungkinkan mereka berinteraksi dengan platform dengan mudah. Panel ini memuat beberapa fitur utama yang mendukung aktivitas pengguna sehari-hari di platform:

- **Pembuatan dan Pengelolaan Bot:** Pengguna dapat membuat bot baru dengan cepat melalui wizard yang mudah dipahami. Setelah bot dibuat, pengguna dapat mengelola setiap aspek bot tersebut, seperti mengubah pengaturan, menambah komando baru, atau bahkan menghapus bot yang sudah tidak dibutuhkan.
- **Akses ke Template Bot:** Pengguna dapat memilih dari berbagai template bot yang disediakan oleh platform. Setiap template dapat disesuaikan berdasarkan preferensi

pengguna, baik dari segi fungsionalitas maupun tampilan.

- ***Dashboard Personal***: Setiap pengguna akan memiliki ***dashboard personal*** yang menampilkan semua informasi terkait bot yang mereka kelola, seperti jumlah interaksi pengguna, jumlah pesan yang dikirimkan bot, dan aktivitas terbaru. Ini memberikan pengguna wawasan yang jelas tentang kinerja bot mereka dan langkah-langkah yang dapat diambil untuk meningkatkan performanya.

Dengan mengikuti panduan dalam buku ini, Anda akan mempelajari cara membangun platform bot yang ***fungsional, aman, dan skalabel*** menggunakan Django. Selain membangun antarmuka pengguna yang ramah, Anda juga akan diajarkan untuk mengelola ***aspek teknis*** di balik layar, seperti autentikasi, validasi token, manajemen bot, serta penerapan webhook untuk integrasi bot yang mulus dengan platform eksternal. Setelah menyelesaikan proyek ini, Anda akan memiliki keterampilan dan pemahaman yang diperlukan untuk mengembangkan platform serupa atau bahkan lebih kompleks di masa mendatang.

Mari Kita Mulai!

Dunia bot menanti Anda. Dengan Django sebagai alat utama Anda, mari kita mulai petualangan ini dan lihat seberapa jauh Anda bisa melangkah. Tidak hanya akan ada tantangan di sepanjang jalan, tetapi juga banyak peluang untuk belajar dan berkembang. Saya yakin bahwa pada akhir buku ini, Anda akan melihat Django bukan hanya sebagai sebuah framework, tetapi sebagai sekutu yang kuat dalam perjalanan pengembangan Anda.

BAB 1 - Pengenalan Django

Dalam bab ini, kita akan menjelajahi dasar-dasar Django, sebuah framework web yang powerful dan sangat populer di kalangan pengembang. Kita akan memulai dengan memahami apa itu Django, termasuk sejarah singkatnya dan mengapa ia menjadi pilihan utama bagi banyak developer di seluruh dunia. Bab ini akan memperkenalkan berbagai fitur utama Django yang membedakannya dari framework lain, seperti Object-Relational Mapping (ORM), Admin Panel otomatis, sistem templating yang fleksibel, hingga dukungannya terhadap keamanan dan performa yang tinggi.

Selain itu, kita juga akan mendalami keunggulan Django, seperti kemampuannya untuk mempercepat proses pengembangan, kemudahan integrasi dengan berbagai database, dan arsitektur yang terstruktur. Untuk mempermudah pemahaman, kita juga akan membahas konsep arsitektur Model-View-Template (MVT) yang menjadi dasar Django dalam membangun aplikasi web.

Setelah memahami fondasi Django, kita akan menelusuri komponen-komponen utama yang membentuk sebuah proyek Django, seperti app, models, views, templates, dan URL routing. Pada akhirnya, bab ini akan memberikan gambaran yang jelas tentang mengapa Django adalah pilihan tepat untuk pengembangan aplikasi web dari skala kecil hingga besar.

Pengenalan Django

Dengan demikian, bab ini akan menjadi landasan penting untuk semua pembahasan teknis selanjutnya, dan kita akan siap untuk mulai membangun aplikasi berbasis Django yang efisien dan scalable.

1.1 Apa Itu Django?

Django adalah framework web berbasis Python yang dirancang untuk memudahkan pengembangan aplikasi web yang kompleks dan dinamis. Dikembangkan oleh Django Software Foundation, framework ini menawarkan berbagai fitur canggih dan praktis untuk mempercepat proses pengembangan.

Secara umum, Django menggunakan konsep arsitektur Model-View-Template (MVT), yang memungkinkan pemisahan antara logika bisnis, data, dan presentasi. Hal ini memudahkan pengembangan proyek yang lebih besar karena setiap bagian dapat dipecah dan ditangani secara independen.

Keunggulan Django terletak pada "batteries included," di mana Django menyediakan banyak fitur bawaan, seperti sistem autentikasi, manajemen URL, dan ORM (Object-Relational Mapping) untuk mengelola database. Dengan Django, kita dapat dengan mudah membangun aplikasi web yang kompleks tanpa harus memikirkan banyak detail teknis seperti keamanan, validasi input, dan manajemen sesi, karena semuanya sudah diurus oleh framework ini.

Berikut adalah beberapa poin penting tentang Django:

Pengenalan Django

Framework Web Full-Stack: Django adalah framework full-stack, artinya ia mencakup berbagai komponen yang diperlukan untuk membangun aplikasi web, seperti sistem manajemen database, routing URL, pengolahan formulir, dan sistem template untuk pembuatan antarmuka pengguna.

Berbasis Python: Django dibangun dengan bahasa pemrograman Python, yang terkenal dengan sintaks yang bersih dan kemudahan penggunaannya. Python memungkinkan pengembang untuk menulis kode yang efisien dan mudah dibaca, menjadikannya pilihan populer dalam pengembangan web.

Fokus pada Keamanan: Django memiliki fitur keamanan bawaan yang membantu melindungi aplikasi dari berbagai ancaman umum, seperti serangan SQL injection, cross-site scripting (XSS), dan cross-site request forgery (CSRF). Ini memberikan perlindungan ekstra dan mengurangi risiko keamanan.

Prinsip DRY (Don't Repeat Yourself): Django mengikuti prinsip DRY, yang berarti bahwa kode yang sama tidak perlu ditulis berulang kali. Ini membantu mengurangi duplikasi kode dan meningkatkan efisiensi pengembangan.

Sistem Admin Otomatis: Salah satu fitur menonjol dari Django adalah sistem admin otomatisnya. Setelah mendefinisikan model data, Django dapat secara otomatis menghasilkan antarmuka admin yang memungkinkan Anda untuk mengelola data aplikasi dengan mudah.

Pengenalan Django

Arsitektur MVC: Django menggunakan arsitektur Model-View-Controller (MVC), meskipun istilah yang digunakan dalam Django adalah Model-View-Template (MVT). Dalam arsitektur ini:

- **Model:** Mendefinisikan struktur data aplikasi dan berinteraksi dengan database.
- **View:** Menangani logika aplikasi dan memproses permintaan dari pengguna.
- **Template:** Menyediakan cara untuk menampilkan data dalam format HTML yang bisa dilihat oleh pengguna.

Komunitas dan Dokumentasi: Django memiliki komunitas pengembang yang aktif dan dokumentasi yang sangat baik. Ini memudahkan untuk mencari dukungan, menemukan solusi untuk masalah, dan mempelajari lebih lanjut tentang framework.

Dengan fitur-fitur tersebut, Django menjadi pilihan yang sangat baik untuk membangun berbagai jenis aplikasi web, dari situs web sederhana hingga aplikasi yang kompleks dan berfitur lengkap. Dalam buku ini, Anda akan mempelajari bagaimana memanfaatkan Django untuk membuat platform pembuatan bot dari awal, dengan pendekatan praktis yang mudah diikuti.

1.2 Sejarah Singkat Django

Django tidak muncul begitu saja sebagai framework web yang kita kenal hari ini. Sejarah Django dimulai pada tahun 2003, ketika tim pengembang di perusahaan media Lawrence Journal-World, yang berbasis di Kansas, Amerika Serikat, dihadapkan pada tantangan untuk mengelola beberapa situs web berita. Situs-

Pengenalan Django

situs ini memerlukan pembaruan cepat dan sering, dan tim pengembangan membutuhkan cara untuk mengelola konten secara efisien. Framework yang ada pada saat itu tidak memenuhi kebutuhan mereka, sehingga mereka mulai membangun framework mereka sendiri yang dapat mempercepat proses pengembangan.

Awalnya, framework ini tidak memiliki nama resmi dan hanya digunakan secara internal. Namun, berkat efisiensi yang ditawarkannya—terutama dalam hal manajemen konten dan pembuatan situs web dinamis—framework ini segera menjadi lebih penting bagi tim. Pada tahun 2005, framework ini secara resmi dirilis sebagai open source dan diberi nama ***Django***, yang diambil dari nama gitaris jazz legendaris, Django Reinhardt. Nama ini mencerminkan sifat framework yang cepat, gesit, dan fleksibel, layaknya gaya bermain gitar Django Reinhardt yang terkenal.

Sejak dirilis, Django terus berkembang dan menarik perhatian banyak pengembang di seluruh dunia. Dengan komunitas yang semakin besar, Django mengalami peningkatan fitur dan perbaikan keamanan secara berkelanjutan. Salah satu perubahan terbesar dalam evolusi Django adalah kemampuannya untuk menangani proyek-proyek yang semakin kompleks, baik dalam hal skalabilitas maupun fleksibilitas. Setiap rilis utama dari Django tidak hanya membawa peningkatan performa tetapi juga fitur baru yang menjawab kebutuhan pengembangan web modern.

Salah satu momen penting dalam sejarah Django adalah saat dirilisnya Django 1.0 pada September 2008. Ini adalah tonggak sejarah utama karena menandakan stabilitas framework dengan API

Pengenalan Django

yang matang, dokumentasi yang lengkap, dan komitmen jangka panjang untuk dukungan backward-compatibility. Seiring berlalunya waktu, Django juga mulai mendukung teknologi terbaru, seperti Python 3, setelah Python 2 mulai ditinggalkan oleh komunitas.

Versi-versi Django selanjutnya semakin menekankan pada kemudahan penggunaan, keamanan, dan skalabilitas. Salah satu keunggulan utama Django adalah fokusnya pada keamanan. Django secara konsisten menjaga pengembang dari kesalahan-kesalahan umum yang sering terjadi dalam pengembangan web, seperti serangan SQL injection, cross-site scripting (XSS), dan cross-site request forgery (CSRF). Fitur-fitur ini membuat Django sangat andal untuk membangun aplikasi web yang aman sejak awal.

Selain itu, Django juga diadopsi oleh berbagai perusahaan teknologi besar dan digunakan dalam skala produksi. Beberapa perusahaan ternama yang menggunakan Django dalam pengembangan aplikasi mereka antara lain Instagram, Pinterest, dan Mozilla. Penggunaan Django oleh perusahaan-perusahaan ini menunjukkan bahwa framework ini mampu menangani skala besar dan volume lalu lintas yang tinggi.

Salah satu alasan utama kesuksesan Django adalah dokumentasinya yang sangat baik. Sejak awal, pengembang Django menyadari bahwa dokumentasi adalah kunci bagi adopsi framework oleh komunitas. Dokumentasi yang tersedia tidak hanya mencakup tutorial dasar, tetapi juga panduan lanjutan untuk fitur-fitur yang lebih kompleks. Ini memungkinkan baik pemula maupun

Pengenalan Django

pengembang berpengalaman untuk menggunakan Django dengan efektif.

Hari ini, Django tetap menjadi salah satu framework web terpopuler di dunia. Dengan rilis terbaru yang terus menghadirkan fitur modern, Django berkomitmen untuk tetap relevan dalam dunia pengembangan web yang terus berkembang. Misalnya, integrasi dengan asynchrony (pengolahan asynchronous) di Django 3.x memungkinkan pengembang membangun aplikasi yang lebih efisien dalam menangani permintaan secara bersamaan.

Dengan fondasi yang kuat, komunitas yang besar, dan dukungan dari perusahaan besar, Django terus menjadi pilihan utama untuk pengembangan aplikasi web di berbagai industri, dari media, e-commerce, hingga teknologi finansial (fintech).

Jika kita bekerja pada proyek seperti *platform_bot*, kita bisa memanfaatkan semua fitur yang telah dikembangkan selama bertahun-tahun ini, seperti manajemen URL yang fleksibel, penggunaan ORM untuk berinteraksi dengan database, dan sistem template yang kuat.

Django dirancang dengan filosofi utama yaitu "Don't Repeat Yourself" (DRY) dan "Explicit is Better Than Implicit." Filosofi ini memungkinkan pengembang untuk menulis kode yang bersih, efisien, dan mudah dipelihara.

1.3 Fitur Utama Django

Salah satu alasan utama Django menjadi pilihan banyak pengembang adalah karena berbagai fitur bawaannya yang memudahkan proses pengembangan aplikasi web. Fitur-fitur ini dirancang untuk mempercepat pengembangan tanpa mengorbankan kualitas dan keamanan. Di bagian ini, kita akan membahas beberapa fitur inti yang membuat Django menonjol, seperti ORM (Object-Relational Mapping), admin panel otomatis, dan sistem routing.

1.3.1 ORM (Object-Relational Mapping):

Django hadir dengan ORM bawaan yang sangat kuat dan efisien, memungkinkan kita berinteraksi dengan database tanpa harus menulis kueri SQL secara manual. ORM ini menghubungkan model-model Python dengan tabel-tabel database, sehingga kita dapat mengelola data dalam aplikasi kita menggunakan objek Python. ORM juga memungkinkan kita untuk melakukan operasi database yang kompleks dengan sintaksis yang sederhana dan intuitif.

Sebagai contoh, kita dapat mendefinisikan model untuk mengelola data bot dalam proyek ***platform_bot***. Model ini akan diterjemahkan ke dalam tabel database yang sesuai.

```
# platform_bot/models.py
from django.db import models

class Bot(models.Model):
    # Mendefinisikan field model
    name = models.CharField(max_length=100) #
    Nama bot
```


Pengenalan Django

```
platform = models.CharField(max_length=50)
# Platform bot (misalnya, Telegram, WhatsApp)
created_at =
models.DateTimeField(auto_now_add=True) #
Tanggal pembuatan bot

def __str__(self):
    return self.name # Mengembalikan nama
bot saat dipanggil
```

Setelah mendefinisikan model, kita perlu membuat dan menerapkan migrasi untuk membuat tabel-tabel yang sesuai di database:

```
$ python manage.py makemigrations
$ python manage.py migrate
```

Dengan Django ORM, kita bisa melakukan operasi database dengan cara yang sederhana. Sebagai contoh, untuk membuat entri baru di tabel **Bot**, kita cukup menulis:

```
# Membuat entri baru untuk Bot
new_bot = Bot(name="ChatBot",
platform="Telegram")
new_bot.save() # Menyimpan entri baru ke
database
```

Kita juga bisa dengan mudah melakukan operasi lain seperti mengambil data, mengubah, atau menghapus entri tanpa perlu menulis kueri SQL.

1.3.2 Admin Panel Otomatis:

Salah satu fitur unik yang membuat Django sangat menarik adalah admin panel otomatis yang dihasilkan dari model yang kita buat. Dengan beberapa konfigurasi sederhana, Django dapat

Pengenalan Django

menghasilkan antarmuka administrasi untuk mengelola data dalam aplikasi kita, yang sangat membantu dalam proses pengembangan dan pengelolaan aplikasi.

Untuk mengaktifkan admin panel untuk model Bot yang kita buat sebelumnya, kita hanya perlu menambahkannya ke dalam file `admin.py`:

```
# platform_bot/admin.py
from django.contrib import admin
from .models import Bot

# Mendaftarkan model Bot ke admin panel
admin.site.register(Bot)
```

Setelah itu, kita bisa mengakses admin panel dengan membuka `http://127.0.0.1:8000/admin/` di browser dan login menggunakan akun superuser yang telah kita buat sebelumnya. Di sana, kita akan melihat antarmuka yang memungkinkan kita untuk membuat, mengedit, atau menghapus entri dalam tabel Bot, tanpa harus menulis kode tambahan untuk CRUD.

Admin panel ini sangat fleksibel dan dapat dikustomisasi lebih lanjut jika diperlukan. Kita bisa menambahkan fitur pencarian, filter, atau mengubah tampilan data dalam admin panel dengan mudah.

1.3.3 Keamanan:

Keamanan merupakan salah satu aspek paling penting dalam pengembangan aplikasi web, dan Django secara konsisten menempatkan keamanan sebagai prioritas utama. Django dilengkapi de-

Pengenalan Django

ngan berbagai fitur keamanan bawaan yang melindungi aplikasi dari ancaman umum di dunia web. Salah satu fitur utama adalah perlindungan terhadap **SQL Injection**, di mana Django ORM memastikan bahwa semua kueri ke database aman dan terlindungi dari manipulasi eksternal.

Selain itu, Django juga melindungi dari **Cross-Site Scripting (XSS)** dan **Cross-Site Request Forgery (CSRF)**. Setiap form yang kita buat di Django secara otomatis dilengkapi dengan token CSRF, yang mencegah serangan yang mencoba memanfaatkan sesi pengguna untuk melakukan tindakan tanpa sepengetahuan mereka. Untuk menjaga keamanan data pengguna, Django juga mendukung **Hashing Password** menggunakan algoritma yang aman.

Misalnya, untuk mengaktifkan keamanan CSRF pada form yang kita buat, Django akan secara otomatis menambahkan token ini dalam file template:

```
<form method="post">
    {% csrf_token %} <!-- Token ini mencegah
serangan CSRF -->
    <!-- Isi form -->
</form>
```

Django juga memastikan bahwa pengembang tidak perlu mengkhawatirkan hal-hal seperti **Clickjacking** dan **Session Hijacking**, karena fitur-fitur ini sudah dikelola secara otomatis.

1.3.4 Templating System:

Django memiliki sistem templating yang sangat kuat, yang memungkinkan kita membangun tampilan halaman web secara dinamis. Sistem ini mendukung penggunaan template untuk memisahkan logika pemrograman dari tampilan. Dengan demikian, pengembangan frontend dan backend bisa dilakukan secara terpisah, dan template dapat digunakan kembali di banyak tempat.

Sistem templating Django mendukung penggunaan **template inheritance**, yang memungkinkan kita membuat satu template dasar yang dapat di-*extend* oleh template-template lain. Misalnya, kita bisa membuat sebuah file `base.html` sebagai kerangka utama, dan template lain seperti halaman bot atau halaman login bisa memperluasnya. Berikut adalah contoh sederhana dari template dasar:

```
<!-- templates/base.html -->
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Platform Bot</title>
</head>
<body>
  <header>
    <!-- Header konten -->
  </header>
  <main>
    {% block content %} <!-- Block yang
akan di-*extend* oleh template lain -->
    {% endblock %}
  </main>
  <footer>
```

```
        <!-- Footer konten -->
    </footer>
</body>
</html>
```

Kemudian, untuk halaman lain, kita bisa menggunakan template inheritance:

```
<!-- templates/bot_list.html -->
{% extends 'base.html' %}

{% block content %}
<h1>Daftar Bot</h1>
<ul>
    {% for bot in bots %}
        <li>{{ bot.name }} - Platform:
    {{ bot.platform }}</li>
    {% endfor %}
</ul>
{% endblock %}
```

Dengan sistem ini, kita bisa menjaga konsistensi antarhalaman dan menghemat waktu dalam penulisan kode template.

1.3.5 Routing URL yang Fleksibel:

Fitur penting lain dari Django adalah sistem routing yang sangat fleksibel. Django menggunakan pola URL yang memungkinkan kita mendefinisikan URL untuk setiap view atau halaman di aplikasi kita dengan cara yang sangat terstruktur. Setiap URL yang dimasukkan pengguna akan dipetakan ke view yang sesuai.

Untuk mendefinisikan pola URL, kita perlu membuat entri di file `urls.py`. Sebagai contoh, jika kita ingin menampilkan daftar

Pengenalan Django

semua bot yang ada, kita bisa menambahkan view dan URL untuk keperluan tersebut.

```
# platform_bot/views.py
from django.shortcuts import render
from .models import Bot

# Menampilkan daftar bot
def bot_list(request):
    bots = Bot.objects.all() # Mengambil semua
    bot dari database
    return render(request, 'bot_list.html',
    {'bots': bots}) # Me-render halaman dengan data
    bot
```

Kemudian, kita tambahkan URL pattern yang menghubungkan view tersebut dengan URL yang sesuai:

```
# platform_bot/urls.py
from django.urls import path
from .views import bot_list

urlpatterns = [
    path('bots/', bot_list, name='bot_list'), #
    URL untuk menampilkan daftar bot
]
```

Dengan sistem routing ini, kita bisa dengan mudah membuat URL yang bersih dan mudah dimengerti untuk setiap halaman di aplikasi kita. Django juga mendukung pola URL yang lebih kompleks, seperti URL dinamis yang menerima parameter, misalnya `bots/<int:id>/` untuk menampilkan detail dari bot tertentu.

Pengenalan Django

Sistem routing ini bekerja dengan baik bersama sistem template Django, yang memungkinkan kita untuk membangun antarmuka pengguna yang dinamis berdasarkan data yang dikirim dari view.

1.3.6 Komunitas dan Dokumentasi yang Kuat:

Salah satu faktor utama yang membuat Django terus berkembang adalah komunitas global yang sangat aktif. Django memiliki komunitas yang solid, di mana pengembang dari seluruh dunia berkontribusi dalam pengembangan fitur baru, memperbaiki bug, dan memberikan dukungan bagi pengguna lain. Django juga terkenal dengan dokumentasinya yang sangat lengkap dan jelas. Dokumentasi ini mencakup segala hal mulai dari pengenalan dasar hingga konsep-konsep yang lebih kompleks.

Django mengadakan konferensi tahunan seperti ***DjangoCon*** yang menjadi tempat berkumpulnya para pengembang Django untuk saling berbagi ilmu, pengalaman, dan praktik terbaik. Komunitas ini juga sangat responsif dalam menangani masalah keamanan, sehingga jika ada kerentanan ditemukan, patch keamanan biasanya dirilis dengan cepat.

Bagi pengembang pemula, dokumentasi resmi Django adalah salah satu yang paling ramah pengguna. Tutorial bawaan Django memungkinkan kita membangun aplikasi dari nol, mulai dari instalasi hingga deployment ke server produksi.

1.3.7 Skalabilitas dan Performa

Django didesain untuk mendukung aplikasi web berskala besar dan tetap dapat dioptimalkan untuk kinerja tinggi. Dengan sistem

Pengenalan Django

caching yang mudah diterapkan dan dukungan terhadap penggunaan database skala besar, Django memungkinkan kita menangani jutaan pengguna dan permintaan secara efisien. Django mendukung berbagai mekanisme caching, seperti caching berbasis memori atau caching berbasis file, untuk mempercepat waktu respon aplikasi.

Sebagai contoh, kita dapat mengaktifkan caching pada level view menggunakan dekorator:

```
# platform_bot/views.py
from django.views.decorators.cache import
cache_page

@cache_page(60 * 15) # Cache selama 15 menit
def bot_list(request):
    bots = Bot.objects.all()
    return render(request, 'bot_list.html',
{'bots': bots})
```

Dengan fitur caching ini, Django akan menyimpan hasil dari request tertentu untuk mengurangi beban di server dan mempercepat waktu respon bagi pengguna.

Skalabilitas Django juga terlihat dari penggunaannya oleh perusahaan besar seperti Instagram dan Pinterest, yang berhasil menjalankan aplikasi berskala besar dengan basis Django.

Django tidak hanya menyediakan alat-alat yang Anda butuhkan untuk membangun aplikasi web, tetapi juga panduan dan best practices untuk memastikan aplikasi Anda aman, scalable, dan

Pengenalan Django

mudah di-maintain. Seiring dengan kemajuan teknologi, Django terus berkembang dan menawarkan fitur-fitur baru untuk mendukung kebutuhan pengembang modern.

Fitur-fitur utama Django, seperti ORM, admin panel otomatis, dan sistem routing yang fleksibel, menjadikannya salah satu framework web paling kuat dan efisien untuk pengembangan aplikasi web. Django tidak hanya membantu kita menulis kode yang bersih dan terstruktur, tetapi juga memungkinkan pengembangan yang cepat dan aman.

1.4 Keunggulan Django

Django adalah framework yang sangat populer di kalangan pengembang karena banyak keunggulan yang ditawarkannya. Keunggulan-keunggulan ini menjadikan Django pilihan utama untuk berbagai jenis proyek, dari aplikasi skala kecil hingga aplikasi besar yang digunakan oleh jutaan pengguna. Dalam bagian ini, kita akan membahas beberapa alasan utama mengapa Django begitu diminati, termasuk kecepatan, keamanan, dan skalabilitasnya.

1.4.1 Pengembangan Cepat

Salah satu alasan utama mengapa banyak pengembang memilih Django adalah kecepatannya dalam membantu membangun aplikasi. Django didesain untuk mempercepat proses pengembangan dengan menyediakan banyak fitur bawaan yang membantu memotong waktu penulisan kode. Misalnya, Django ORM yang kuat memungkinkan kita bekerja dengan database tanpa harus me-

Pengenalan Django

nulis kueri SQL yang rumit, sementara admin panel otomatis memudahkan pengelolaan data tanpa perlu membangun panel admin dari nol.

Selain itu, Django juga menyediakan **reusable components** yang bisa digunakan berkali-kali di berbagai bagian aplikasi, seperti template, form, dan view. Ini memudahkan kita untuk mengurangi redundansi kode dan mempercepat pengembangan fitur baru. Django mengikuti prinsip **DRY (Don't Repeat Yourself)**, yang berarti bahwa kita dapat menulis kode yang lebih efisien dan mudah dipelihara.

Sebagai contoh, saat mengembangkan aplikasi bot dalam proyek **platform_bot**, kita bisa membuat view yang berfungsi untuk menampilkan daftar bot, dan menggunakannya di banyak tempat dengan sedikit modifikasi:

```
# platform_bot/views.py
from django.shortcuts import render
from .models import Bot

# View untuk menampilkan daftar bot
def bot_list(request):
    bots = Bot.objects.all() # Mengambil semua
    bot dari database
    return render(request, 'bot_list.html',
    {'bots': bots}) # Me-render halaman dengan data
    bot
```

Dengan Django, kita dapat membangun aplikasi kompleks dalam waktu yang lebih singkat, tanpa harus mengorbankan fungsionalitas.

1.4.2 Keamanan Terintegrasi

Keamanan adalah prioritas utama bagi Django, dan ini adalah salah satu alasan utama mengapa framework ini sering dipilih untuk aplikasi web berskala besar. Django secara otomatis menangani banyak aspek keamanan, seperti ***CSRF protection***, ***XSS protection***, ***SQL Injection protection***, dan ***secure password hashing***. Semua fitur ini diaktifkan secara default, sehingga pengembang tidak perlu khawatir tentang keamanan aplikasi pada level dasar.

Misalnya, ketika kita membuat form dalam Django, token ***CSRF*** akan secara otomatis disertakan untuk mencegah serangan yang memanfaatkan sesi pengguna:

```
<form method="post">
    {% csrf_token %} <!-- Token CSRF secara
otomatis di-generate oleh Django -->
    <!-- Input form -->
</form>
```

Dengan Django, pengembang dapat fokus pada logika bisnis tanpa harus terlalu khawatir dengan ancaman keamanan umum yang sering kali menyerang aplikasi web.

1.4.3 Skalabilitas

Django telah terbukti sangat skalabel, yang berarti bahwa aplikasi yang dibangun menggunakan Django dapat dengan mudah tumbuh untuk menangani ratusan hingga jutaan pengguna. Ini adalah salah satu alasan mengapa Django dipilih oleh perusahaan besar seperti Instagram dan Pinterest. Dengan arsitektur yang

Pengenalan Django

modular dan dukungan untuk berbagai solusi caching dan database, Django dapat dioptimalkan untuk performa yang lebih tinggi seiring dengan meningkatnya jumlah pengguna.

Django mendukung integrasi dengan berbagai jenis database, dari SQLite yang ringan hingga PostgreSQL dan MySQL yang lebih kuat untuk aplikasi berskala besar. Django juga mendukung **horizontal scaling** dengan memungkinkan aplikasi berjalan di beberapa server untuk menangani lebih banyak lalu lintas.

Kita juga dapat menggunakan fitur **caching** bawaan Django untuk meningkatkan performa aplikasi dengan mengurangi beban server. Caching sangat berguna untuk halaman atau operasi yang sering kali menghasilkan data yang sama, sehingga tidak perlu diproses berulang kali.

Sebagai contoh, untuk mengaktifkan caching di level view, kita dapat menggunakan dekorator `cache_page` yang disediakan Django:

```
# platform_bot/views.py
from django.views.decorators.cache import
cache_page

@cache_page(60 * 15) # Cache view selama 15
menit
def bot_list(request):
    bots = Bot.objects.all() # Mengambil data
    bot dari database
    return render(request, 'bot_list.html',
{'bots': bots}) # Me-render halaman dengan data
bot
```

Pengenalan Django

Dengan cara ini, aplikasi yang dibangun dengan Django dapat menangani lebih banyak permintaan tanpa mengorbankan performa.

1.4.4 Arsitektur yang Terstruktur

Django didesain dengan struktur arsitektur yang sangat teratur, yang memisahkan antara **model (M)**, **view (V)**, dan **template (T)**. Struktur ini dikenal sebagai pola **Model-View-Template (MVT)**. Dengan menggunakan pola ini, kita dapat menjaga keteraturan dalam pengembangan aplikasi, membuat kode yang lebih mudah dipelihara dan diukur seiring dengan pertumbuhan proyek.

Dalam arsitektur MVT, **Model** bertanggung jawab atas pengelolaan data dan interaksi dengan database, **View** menangani logika aplikasi dan pemrosesan request, sementara **Template** mengatur bagaimana data ditampilkan ke pengguna. Pemisahan ini memungkinkan tim pengembang untuk bekerja secara paralel pada bagian yang berbeda dari aplikasi tanpa saling mengganggu.

Misalnya, dalam aplikasi **platform_bot**, kita bisa memiliki struktur MVT seperti ini:

- **Model** di `platform_bot/models.py`, yang mendefinisikan data bot.
- **View** di `platform_bot/views.py`, yang memproses permintaan untuk menampilkan daftar bot.
- **Template** di `platform_bot/templates/bot_list.html`, yang menampilkan data bot di halaman web.

Pengenalan Django

Dengan struktur ini, kita bisa memisahkan logika aplikasi dari tampilan, memudahkan pengelolaan kode, dan memastikan aplikasi tetap modular dan terukur seiring dengan pertumbuhan proyek.

1.4.5 Admin Interface yang Otomatis

Salah satu fitur paling menarik dari Django adalah ***admin interface*** yang dibangunnya secara otomatis. Saat kita membuat model dalam Django, framework ini secara otomatis menghasilkan panel admin yang memungkinkan kita untuk mengelola data dengan mudah tanpa perlu menulis kode tambahan untuk antarmuka admin. Fitur ini sangat bermanfaat, terutama untuk pengembangan cepat dan pengelolaan aplikasi internal.

Panel admin Django memungkinkan kita untuk ***menambah, mengedit, dan menghapus data*** langsung dari antarmuka yang disediakan. Kita juga bisa menyesuaikan tampilan panel admin sesuai kebutuhan, dengan menambahkan model tertentu atau mengatur bagaimana data ditampilkan.

Sebagai contoh, dalam proyek ***platform_bot***, kita bisa mendaftarkan model bot ke dalam admin panel hanya dengan menambahkan beberapa baris kode di file `admin.py`:

```
# platform_bot/admin.py
from django.contrib import admin
from .models import Bot

# Mendaftarkan model Bot ke dalam admin panel
admin.site.register(Bot)
```

Pengenalan Django

Setelah langkah ini, kita bisa mengakses admin panel melalui URL `/admin` dan langsung mengelola data bot tanpa perlu membangun antarmuka sendiri.

1.4.6 Dukungan untuk Berbagai Database

Django mendukung berbagai jenis *database*, termasuk *SQLite*, *PostgreSQL*, *MySQL*, dan *Oracle*. Kemampuan untuk bekerja dengan banyak jenis database ini menjadikan Django sangat fleksibel dan memungkinkan pengembang untuk memilih database yang sesuai dengan kebutuhan proyek mereka. Pada saat pengembangan, kita mungkin menggunakan *SQLite*, yang ringan dan mudah diatur, namun seiring pertumbuhan aplikasi, kita bisa dengan mudah berpindah ke database yang lebih kuat seperti *PostgreSQL* atau *MySQL*.

Untuk mengubah database, cukup dengan mengkonfigurasi pengaturan di file `settings.py`:

```
# platform_bot/settings.py
DATABASES = {
    'default': {
        'ENGINE':
'django.db.backends.postgresql', # Menggunakan
PostgreSQL
        'NAME': 'platform_bot_db',
        'USER': 'postgres_user',
        'PASSWORD': 'your_password',
        'HOST': 'localhost',
        'PORT': '5432',
    }
}
```

Pengenalan Django

Dengan beberapa perubahan pada konfigurasi ini, Django secara otomatis akan menyesuaikan dirinya untuk bekerja dengan database yang baru.

1.4.7 Fitur Built-in untuk Pengembangan Web

Django menyediakan banyak *fitur built-in* yang sangat memudahkan pengembangan web. Mulai dari sistem *routing* hingga pengelolaan *session*, Django telah menyiapkan berbagai alat yang diperlukan untuk membangun aplikasi web yang lengkap. Misalnya, Django memiliki sistem *form* bawaan yang memudahkan kita untuk membuat, memvalidasi, dan memproses data dari form HTML.

Django juga menyertakan berbagai fitur *middleware*, seperti pengelolaan *authentication* dan *authorization*, yang membuat pengelolaan sesi pengguna menjadi lebih mudah. Ini memungkinkan kita untuk fokus pada logika aplikasi, sementara

Django menangani detail teknis yang lebih kompleks.

Selain itu, Django juga dilengkapi dengan sistem *caching*, yang dapat digunakan untuk meningkatkan performa aplikasi dengan menyimpan data sementara. Semua fitur ini disediakan oleh Django tanpa memerlukan konfigurasi atau penambahan pustaka eksternal yang berlebihan.

1.4.8 Kemudahan Pengujian

Django menyediakan dukungan *built-in* untuk pengujian, sehingga pengembang dapat dengan mudah menulis dan menjalankan *unit test* untuk memeriksa apakah fitur-fitur aplikasi berfungsi

Pengenalan Django

dengan benar. Pengujian adalah bagian penting dari pengembangan perangkat lunak, dan Django membuatnya lebih mudah dengan menyediakan kerangka kerja pengujian yang terintegrasi.

Kita bisa menulis test case untuk memastikan bahwa view, model, dan fungsi lain bekerja sesuai harapan. Django juga menyediakan alat untuk melakukan ***pengujian basis data***, yang memungkinkan kita untuk menguji integritas data selama pengembangan.

Sebagai contoh, untuk menguji apakah view yang menampilkan daftar bot berfungsi dengan benar, kita dapat menulis test case seperti ini:

```
# platform_bot/tests.py
from django.test import TestCase
from .models import Bot

class BotViewTests(TestCase):
    def test_bot_list_view(self):
        response = self.client.get('/bots/') #
        # Mengirim request ke view daftar bot
        self.assertEqual(response.status_code,
        200) # Memastikan status response adalah 200 OK
        self.assertContains(response, 'Daftar
        Bot') # Memeriksa apakah teks "Daftar Bot" ada
        di halaman
```

Pengujian ini dapat dijalankan dengan perintah terminal berikut:

```
$ python manage.py test
```

Dengan Django, kita dapat mengotomatisasi pengujian dan menjaga kualitas aplikasi sepanjang siklus pengembangan.

1.4.9 Dukungan untuk Pengembangan Aplikasi Besar

Django adalah pilihan yang sangat tepat untuk pengembangan aplikasi berskala besar karena menyediakan struktur yang jelas dan terorganisir dengan baik. Django memaksa pengembang untuk memisahkan berbagai komponen seperti model, view, dan template, yang membantu dalam pengelolaan kode dan memudahkan kolaborasi antar anggota tim.

Selain itu, Django juga mendukung pengembangan aplikasi dengan pendekatan **modular**, di mana setiap bagian aplikasi dapat dikemas dalam bentuk **app**. Ini memudahkan untuk membagi-bagi fitur aplikasi yang besar ke dalam modul-modul yang lebih kecil dan lebih mudah dikelola. Misalnya, dalam proyek **platform_bot**, kita bisa membuat app terpisah untuk mengelola bot, komponen platform, dan sebagainya, tanpa mencampurkan semuanya dalam satu tempat.

Dengan menggunakan Django, tim pengembang dapat bekerja pada bagian-bagian yang berbeda dari aplikasi tanpa mengalami konflik, karena Django menyediakan struktur dan panduan yang jelas.

Keunggulan-keunggulan Django, mulai dari kecepatan pengembangannya yang luar biasa, hingga keamanannya yang kuat dan

Pengenalan Django

skalabilitasnya yang terbukti, menjadikannya pilihan yang sangat populer di kalangan pengembang. Django tidak hanya membantu pengembang membangun aplikasi dengan cepat, tetapi juga memastikan aplikasi tersebut aman dan siap untuk menangani penggunaan dalam jumlah besar.

Dengan keunggulan-keunggulan ini, Django menjadi pilihan yang solid untuk pengembangan aplikasi web, baik untuk proyek kecil maupun besar. Dalam buku ini, Anda akan memanfaatkan keunggulan Django untuk membangun platform pembuatan bot yang efektif dan efisien.

1.5 Arsitektur MVC Dalam Django

Salah satu konsep arsitektural yang menjadi fondasi banyak framework web adalah **Model-View-Controller (MVC)**. Django juga mengadopsi pendekatan ini, meskipun dengan sedikit modifikasi. Dalam Django, pendekatan MVC berubah menjadi **Model-View-Template (MVT)**, di mana komponen utama dari arsitektur ini tetap menjaga pemisahan yang jelas antara logika bisnis, data, dan presentasi.

Pada intinya, baik MVC maupun MVT bertujuan untuk memisahkan tanggung jawab dalam aplikasi sehingga kode tetap bersih, modular, dan mudah dipelihara. Perbedaannya, dalam Django, **Template** mengambil peran yang biasanya dipegang oleh **Controller** dalam MVC. Django menangani banyak tugas yang biasanya menjadi tanggung jawab controller di framework lain,

seperti pemetaan URL ke view dan pemrosesan permintaan HTTP, secara otomatis.

1.5.1 Model

Model dalam Django mewakili data dan logika bisnis dari aplikasi. Model berhubungan langsung dengan database dan bertanggung jawab untuk mengelola data, termasuk definisi kolom dan relasi antar entitas dalam database. Model inilah yang menentukan bagaimana data disimpan, diubah, dan diambil kembali dari database. Django menyediakan **ORM (Object-Relational Mapping)**, yang memungkinkan kita bekerja dengan database menggunakan objek Python alih-alih SQL.

Fungsi: Di Django, model didefinisikan sebagai kelas dalam file `models.py`. Setiap kelas model mewakili sebuah tabel dalam database, dan setiap atribut kelas mewakili kolom tabel. Django menyediakan berbagai alat untuk melakukan migrasi database berdasarkan model-model ini.

Contoh: Sebagai contoh, jika Anda memiliki model `User`, maka model ini akan mendefinisikan atribut seperti nama, email, dan kata sandi, serta metode untuk berinteraksi dengan data pengguna.

1.5.2 View

View adalah bagian dari aplikasi yang menangani logika aplikasi dan menentukan bagaimana data diproses dan dikirimkan ke pengguna. View bertanggung jawab untuk menerima permintaan HTTP, mengambil data yang diperlukan dari model, dan mengembalikan respons berupa halaman web (atau jenis respons la-

Pengenalan Django

in, seperti JSON). Dalam Django, view juga berfungsi sebagai penghubung antara *model* dan *template*.

Fungsi: Di Django, view didefinisikan sebagai fungsi atau kelas dalam file `views.py`. View mengambil data dari model, mengolahnya jika perlu, dan mengirimkan data tersebut ke template untuk ditampilkan kepada pengguna.

Contoh: Misalnya, view `profile_view` mungkin mengambil data pengguna dari model `User`, kemudian mengirimkan data tersebut ke template `profile.html` untuk ditampilkan.

1.5.3 Template

Template adalah bagian dari aplikasi yang menangani presentasi atau tampilan. Django menggunakan template untuk merender data dari view ke dalam format HTML yang akan ditampilkan kepada pengguna. Template dalam Django menggunakan bahasa template yang sederhana, memungkinkan kita untuk menulis HTML dengan elemen dinamis yang diambil dari view.

Fungsi: Di Django, template didefinisikan dalam file HTML yang terletak di direktori `templates`. Template menggunakan sintaks Django Template Language (DTL) untuk memasukkan data dari view ke dalam HTML.

Contoh: Sebagai contoh, template `profile.html` akan menentukan bagaimana data pengguna (seperti nama dan email) ditampilkan dalam bentuk HTML yang mudah dibaca.

1.5.4 Bagaimana MVT Bekerja Bersama

Arsitektur **MVT** bekerja dengan cara yang sangat terstruktur dan jelas dalam aplikasi Django. Proses alirannya dapat dijelaskan seperti ini:

1. **Permintaan (Request)**: Ketika pengguna mengunjungi halaman tertentu di aplikasi Django, permintaan HTTP dikirim ke server. Django menggunakan sistem **URL routing** untuk memetakan URL yang diminta ke view yang sesuai.
2. **View**: Setelah permintaan diterima, Django memanggil **view** yang bertanggung jawab untuk menangani permintaan tersebut. View mungkin akan berinteraksi dengan **model** untuk mengambil data dari database.
3. **Model**: Jika view membutuhkan data dari database, view akan meminta data tersebut melalui **model**. Model mengelola logika bisnis dan memastikan bahwa data yang diminta sesuai dengan aturan aplikasi.
4. **Template**: Setelah view mendapatkan data yang dibutuhkan, view akan menggunakan **template** untuk merender tampilan data dalam format HTML. Template mengambil data yang diproses oleh view dan menampilkannya kepada pengguna dalam antarmuka yang telah dirancang.
5. **Respons**: Django mengirimkan **respons** dalam bentuk HTML atau format lain (seperti JSON) kembali ke pengguna melalui browser mereka.

Sebagai contoh, ketika pengguna mengunjungi URL `/bots/` untuk melihat daftar bot:

Pengenalan Django

- **URL routing** akan memetakan permintaan ini ke view `bot_list`.
- **View `bot_list`** akan mengambil data bot dari **model `Bot`**.
- Data tersebut akan dikirim ke **template `bot_list.html`**.
- Template akan merender data menjadi HTML, yang kemudian dikirim kembali sebagai respons ke browser pengguna.

Dengan cara ini, Django menjaga pemisahan yang jelas antara data (model), logika aplikasi (view), dan presentasi (template), memastikan aplikasi mudah dipelihara dan dikelola.

Dengan memisahkan aplikasi menjadi komponen Model, View, dan Template, Django memungkinkan pengembang untuk mengelola dan mengorganisir kode dengan lebih baik, serta mempermudah pemeliharaan dan pengembangan aplikasi web.

1.6 Komponen-Komponen Utama Django

Django terdiri dari berbagai komponen inti yang bekerja bersama untuk menciptakan aplikasi web yang fungsional dan terstruktur. Setiap komponen memiliki peran spesifik dan mendukung pengembangan aplikasi yang modular dan mudah dikelola. Dalam sub bab ini, kita akan membahas komponen-komponen penting yang menjadi fondasi framework Django.

1.6.1 App

Dalam Django, istilah **app** (aplikasi) merujuk pada bagian-bagian kecil dari sebuah proyek yang memiliki fungsi tertentu dan terpisah. Django dirancang untuk mendukung pengembangan aplikasi yang modular, yang berarti sebuah proyek Django dapat terdiri dari banyak aplikasi yang terpisah namun saling berhubungan. Setiap aplikasi dapat mengelola satu bagian dari fungsionalitas keseluruhan proyek, misalnya aplikasi untuk mengelola pengguna, aplikasi untuk blog, atau aplikasi untuk sistem pemesanan.

Dengan pendekatan ini, aplikasi Django sangat **mudah dipindahkan**, artinya kita bisa memindahkan aplikasi yang sudah kita buat ke proyek lain dengan sedikit perubahan konfigurasi. Hal ini mendukung **reusability** yang tinggi, sehingga kode yang sama bisa digunakan kembali dalam proyek yang berbeda.

Membuat Aplikasi Baru

Setiap proyek Django biasanya terdiri dari beberapa aplikasi. Untuk membuat aplikasi baru, kita bisa menggunakan perintah berikut di terminal:

```
$ python manage.py startapp myapp
```

Perintah ini akan menghasilkan struktur direktori dan file yang diperlukan untuk aplikasi tersebut:

```
myapp/  
  migrations/  
  __init__.py  
  admin.py
```


Pengenalan Django

```
apps.py
models.py
tests.py
views.py
```

Setiap file dalam direktori ini memiliki peran spesifik:

- ***admin.py***: Digunakan untuk mengonfigurasi panel admin Django.
- ***apps.py***: Berisi konfigurasi untuk aplikasi tersebut.
- ***models.py***: Tempat kita mendefinisikan model, yaitu representasi dari data dalam database.
- ***views.py***: Berisi view yang menghubungkan logika aplikasi dengan presentasi.
- ***tests.py***: Digunakan untuk menulis unit test untuk aplikasi ini.

Setelah aplikasi dibuat, kita perlu menambahkannya ke proyek Django dengan menambahkannya ke dalam daftar `INSTALLED_APPS` di file `settings.py`:

```
# platform_bot/settings.py
INSTALLED_APPS = [
    # Aplikasi-aplikasi bawaan Django
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    # Aplikasi yang kita buat
    'myapp', # Aplikasi yang baru saja kita
    buat
]
```

Pengenalan Django

Dengan menambahkannya ke `INSTALLED_APPS`, Django akan mengenali aplikasi tersebut dan memasukkannya ke dalam siklus pengembangan.

Struktur Modular untuk Proyek Besar

Pada proyek besar, arsitektur aplikasi modular ini sangat berguna. Kita bisa membagi setiap bagian aplikasi menjadi modul-modul kecil, di mana setiap modul menangani satu aspek spesifik dari proyek. Sebagai contoh, dalam proyek *platform_bot*, kita mungkin memiliki beberapa aplikasi seperti:

- ***auth_app***: Untuk menangani otentikasi dan otorisasi pengguna.
- ***bot_app***: Untuk mengelola bot dan platform yang terhubung.
- ***command_app***: Untuk mengelola perintah bot yang dapat dikustomisasi oleh pengguna.

Dengan memisahkan aplikasi ke dalam modul-modul terpisah, setiap tim pengembang bisa bekerja pada satu bagian tanpa saling bertabrakan dengan bagian lain. Django juga memungkinkan pengembang untuk menggunakan aplikasi pihak ketiga, yang bisa langsung diinstal ke dalam proyek tanpa harus menulis ulang fungsionalitas yang sudah ada.

Pengelolaan Aplikasi Django

Pengelolaan aplikasi dalam Django dilakukan dengan perintah ***manage.py***. File `manage.py` adalah alat yang digunakan untuk berinteraksi dengan proyek Django, seperti membuat aplikasi, menjalankan migrasi, dan mengelola database.

Pengenalan Django

Beberapa perintah umum untuk mengelola aplikasi adalah:

- Membuat aplikasi baru:

```
$ python manage.py startapp app_name
```

- Melakukan migrasi database untuk model yang baru dibuat:

```
$ python manage.py makemigrations
```

- Menerapkan perubahan migrasi ke database:

```
$ python manage.py migrate
```

Sebagai contoh, jika kita menambahkan model baru ke aplikasi *myapp*, kita harus membuat migrasi dan menerapkannya agar perubahan tersebut tercermin dalam database.

Contoh Penggunaan Aplikasi dalam Proyek

Sebagai contoh, mari kita buat aplikasi baru bernama *bot_app* dalam proyek *platform_bot*. Pertama, kita akan membuat aplikasi menggunakan perintah terminal:

```
$ python manage.py startapp bot_app
```

Setelah aplikasi dibuat, kita akan menambahkannya ke dalam file *settings.py*:

```
# platform_bot/settings.py
INSTALLED_APPS = [
    # Aplikasi bawaan Django
```

Pengenalan Django

```
'django.contrib.admin',
'django.contrib.auth',
'django.contrib.contenttypes',
'django.contrib.sessions',
'django.contrib.messages',
'django.contrib.staticfiles',

# Aplikasi yang kita buat
'bot_app', # Aplikasi baru yang mengelola
bot
]
```

Dalam `bot_app/models.py`, kita bisa mendefinisikan model yang mewakili data bot:

```
# bot_app/models.py
from django.db import models

class Bot(models.Model):
    name = models.CharField(max_length=100)
    platform = models.CharField(max_length=50)
    created_at =
models.DateTimeField(auto_now_add=True)
```

Model ini mengatur data bot yang akan dikelola dalam aplikasi. Setelah model dibuat, kita bisa menjalankan perintah migrasi untuk menerapkannya ke database:

```
$ python manage.py makemigrations bot_app
$ python manage.py migrate
```

Aplikasi ini sekarang menjadi bagian dari proyek, siap untuk dikembangkan lebih lanjut dengan penambahan view, form, template, dan fitur lainnya.

Pengenalan Django

Dengan pendekatan aplikasi yang modular seperti ini, Django mendukung pengembangan proyek besar dengan skala yang mudah dikelola. Setiap aplikasi bisa dikembangkan dan diuji secara terpisah, namun tetap bekerja bersama dalam satu proyek yang utuh.

1.6.2 Template

Dalam Django, **Template** adalah komponen yang menangani aspek tampilan atau presentasi dari aplikasi web. Template bertugas untuk mengonversi data dari view menjadi format yang dapat ditampilkan di browser, seperti HTML. Dengan menggunakan sistem template Django, kita dapat membangun antarmuka pengguna yang dinamis, memungkinkan data yang ditampilkan berubah-ubah tergantung pada input dan status aplikasi.

Sistem Template Django

Django menggunakan sistem template berbasis bahasa template yang sederhana namun kuat. Sistem ini memungkinkan kita untuk menggabungkan HTML statis dengan elemen dinamis yang diambil dari view, sehingga menghasilkan halaman web yang interaktif dan responsif. Sistem template ini memungkinkan kita untuk memisahkan logika presentasi dari logika bisnis dan data, sehingga menjaga kode tetap bersih dan terorganisir.

Setiap template dalam Django berisi kombinasi HTML dan template tags yang digunakan untuk menyisipkan data dari view. Template tags dan filters memungkinkan kita untuk melakukan operasi seperti perulangan, kondisi, dan format data langsung di dalam template.

Pengenalan Django

Struktur Template

Template ditempatkan di dalam direktori `templates` yang berada di dalam aplikasi Django atau di dalam folder proyek. Struktur direktori template bisa disesuaikan, namun umumnya template dikelompokkan dalam subdirektori berdasarkan aplikasi atau jenis tampilan.

Sebagai contoh, jika kita memiliki aplikasi bernama ***bot_app***, struktur template mungkin terlihat seperti ini:

```
bot_app/  
  templates/  
    bot_app/  
      bot_list.html  
      bot_detail.html
```

Dalam direktori `templates/bot_app/`, kita bisa menyimpan template yang digunakan untuk menampilkan data terkait bot, seperti daftar bot dan detail bot.

Membuat dan Menggunakan Template

Untuk membuat template, kita bisa membuat file HTML di dalam direktori `templates` dan menulis HTML biasa dengan menambahkan template tags yang diperlukan. Misalnya, kita dapat membuat template `bot_list.html` untuk menampilkan daftar bot seperti berikut:

```
<!-- bot_app/templates/bot_app/bot_list.html -->  
<!DOCTYPE html>  
<html lang="en">
```

```
<head>
  <meta charset="UTF-8">
  <title>Daftar Bot</title>
</head>
<body>
  <h1>Daftar Bot</h1>
  <ul>
    {% for bot in bots %}
    <li>{{ bot.name }} - {{ bot.platform
}}</li>
    {% empty %}
    <li>Tidak ada bot yang ditemukan.</li>
    {% endfor %}
  </ul>
</body>
</html>
```

Di sini, `{% for bot in bots %}` adalah template tag yang digunakan untuk melakukan iterasi melalui daftar `bots` yang dikirim dari view. Template tag `{% empty %}` memberikan alternatif jika daftar kosong, sedangkan `{{ bot.name }}` dan `{{ bot.platform }}` digunakan untuk menyisipkan data dinamis ke dalam template.

Menggunakan Template dalam View

Setelah template dibuat, kita harus menghubungkannya dengan view yang akan memproses data dan meneruskannya ke template. Misalnya, dalam aplikasi **bot_app**, kita mungkin memiliki view `bot_list` yang digunakan untuk merender template `bot_list.html` dengan data bot:

```
# bot_app/views.py
from django.shortcuts import render
from .models import Bot
```

```
def bot_list(request):  
    bots = Bot.objects.all() # Mengambil semua  
    data bot dari database  
    return render(request,  
        'bot_app/bot_list.html', {'bots': bots})
```

Fungsi `render` dalam view ini menerima tiga argumen:

- `request`: Objek permintaan HTTP.
- `'bot_app/bot_list.html'`: Jalur ke template yang akan dirender.
- `{'bots': bots}`: Konteks yang berisi data yang akan digunakan dalam template.

Django akan mengambil template `bot_list.html`, menggantikan placeholder dengan data yang dikirim dari view, dan menghasilkan halaman HTML yang dikirim sebagai respons kepada pengguna.

Template Inheritance

Django juga mendukung *template inheritance*, yang memungkinkan kita untuk membuat template dasar dan mewarisi atau mengubah bagian tertentu dari template tersebut di template anak. Ini sangat berguna untuk konsistensi dan pemeliharaan antarmuka pengguna di seluruh aplikasi.

Sebagai contoh, kita bisa membuat template dasar `base.html` yang berisi struktur HTML umum dan kemudian membuat template spesifik yang mewarisi `base.html`:

Pengenalan Django

```
<!-- bot_app/templates/base.html -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>{% block title %}My Site{% endblock
%}</title>
</head>
<body>
    <header>
        <h1>Website Header</h1>
    </header>
    <main>
        {% block content %}{% endblock %}
    </main>
    <footer>
        <p>Website Footer</p>
    </footer>
</body>
</html>
```

Kemudian, kita bisa membuat template `bot_list.html` yang mewarisi `base.html` dan mengisi blok konten yang spesifik:

```
<!-- bot_app/templates/bot_app/bot_list.html -->
{% extends 'base.html' %}

{% block title %}Daftar Bot{% endblock %}

{% block content %}
<h2>Daftar Bot</h2>
<ul>
    {% for bot in bots %}
    <li>{{ bot.name }} - {{ bot.platform }}</li>
    {% empty %}
    <li>Tidak ada bot yang ditemukan.</li>
    {% endfor %}
</ul>
```

```
{% endblock %}
```

Dengan menggunakan inheritance, kita dapat menjaga konsistensi desain dan memudahkan pemeliharaan kode HTML.

Sistem template Django memungkinkan pengembang untuk memisahkan logika presentasi dari logika aplikasi dengan efisien. Dengan kemampuan untuk menggunakan template tags dan filters, serta mendukung template inheritance, Django memberikan fleksibilitas dalam membangun antarmuka pengguna yang dinamis dan terstruktur. Template yang terpisah dan modular mendukung pengembangan antarmuka yang bersih, konsisten, dan mudah dikelola.

1.6.3 URLs

Dalam Django, sistem URL routing adalah komponen utama yang menghubungkan permintaan dari pengguna ke logika pemrosesan aplikasi yang tepat. URL routing bekerja dengan mencocokkan permintaan URL yang diterima oleh server dengan pola URL yang telah didefinisikan di dalam aplikasi Django. Setiap URL dipetakan ke sebuah view yang akan menangani permintaan tersebut dan mengembalikan respons yang sesuai, baik itu berupa halaman HTML, data JSON, atau tipe respons lainnya.

Cara Kerja URL Routing

URL routing di Django diatur melalui file `urls.py`. Setiap aplikasi Django umumnya memiliki file `urls.py` yang berfungsi untuk mendefinisikan pola URL yang terkait dengan aplikasi ter-

Pengenalan Django

sebut. Django menggunakan regex atau path converter untuk mencocokkan pola URL dengan request yang datang.

Saat pengguna mengakses URL pada aplikasi Django, permintaan tersebut akan dikirimkan ke middleware Django, yang kemudian mencocokkannya dengan pola URL yang telah didefinisikan. Jika ditemukan kecocokan, permintaan akan diteruskan ke view yang bersangkutan. Jika tidak ada kecocokan, Django akan mengembalikan respons 404 (Page Not Found).

Contoh Struktur URL

Struktur file `urls.py` di sebuah aplikasi Django biasanya seperti ini:

```
# bot_app/urls.py
from django.urls import path
from . import views

urlpatterns = [
    path('bots/', views.bot_list,
        name='bot_list'),
    path('bots/<int:id>/', views.bot_detail,
        name='bot_detail'),
]
```

Dalam contoh di atas:

- URL `'bots/'` dipetakan ke view `bot_list`, yang mungkin digunakan untuk menampilkan daftar bot. Kita menggunakan `path()` untuk mendefinisikan URL, yang lebih bersahabat dibandingkan pendekatan regex pada versi Django sebelumnya.

Pengenalan Django

- URL `'bots/<int:id>/'` dipetakan ke view `bot_detail`. `<int:id>` adalah ***path converter*** yang menangkap integer dari URL dan meneruskannya sebagai argumen ke view. Ini memungkinkan kita untuk membuat URL dinamis yang bergantung pada data tertentu, seperti ID bot.

Penjelasan Pola URL

Fungsi `path()` menerima beberapa argumen:

1. ***route***: Ini adalah pola URL yang akan dicocokkan. Dalam contoh di atas, `bots/` dan `bots/<int:id>/` adalah rute yang akan dicocokkan oleh Django.
2. ***view***: Fungsi view yang akan dipanggil jika rute cocok. Misalnya, `views.bot_list` dan `views.bot_detail`.
3. ***name***: Nama unik untuk pola URL ini. Nama ini dapat digunakan untuk membuat tautan secara dinamis di template dan view menggunakan fungsi `reverse()` atau tag `{% url %}`.

Menyambungkan URL Aplikasi ke Proyek

Setelah kita mendefinisikan URL di dalam aplikasi, kita perlu menyambungkannya ke proyek utama Django melalui file `urls.py` di direktori proyek. File `urls.py` di tingkat proyek berfungsi sebagai penghubung antara URL root dari aplikasi dan URL yang lebih spesifik dari masing-masing aplikasi.

Sebagai contoh, berikut adalah file `urls.py` di proyek utama:

Pengenalan Django

```
# platform_bot/urls.py
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('bot_app/', include('bot_app.urls')),
]
```

Dalam contoh ini, kita menggunakan `include()` untuk memasukkan URL dari aplikasi **bot_app** ke dalam URL proyek. Jadi, setiap permintaan ke `http://localhost:8000/bot_app/` akan diteruskan ke file `urls.py` milik **bot_app** dan dicocokkan dengan pola URL yang ada di sana.

Pola URL yang Dinamis

Salah satu keunggulan URL routing Django adalah fleksibilitas untuk mencocokkan pola URL yang dinamis. Misalnya, kita dapat menangkap parameter dari URL menggunakan converters, seperti integer, slug, dan lain-lain. Django juga memungkinkan kita untuk menambahkan konverter kustom jika dibutuhkan.

Contoh penggunaan path converter dalam URL:

```
# bot_app/urls.py
from django.urls import path
from . import views

urlpatterns = [
    path('bots/<slug:slug>/',
        views.bot_detail_by_slug,
        name='bot_detail_by_slug'),
]
```

]

Dalam contoh ini, kita menggunakan converter `slug`, yang cocok untuk URL yang lebih ramah SEO. View `bot_detail_by_slug` akan menerima nilai `slug` sebagai argumen.

Menggunakan Nama URL di Template

Dengan menambahkan nama pada pola URL, kita bisa memanfaatkannya untuk membuat tautan di template secara dinamis. Sebagai contoh, jika kita ingin membuat tautan ke halaman detail bot, kita dapat menggunakan tag template `{% url %}`:

```
<!-- bot_app/templates/bot_app/bot_list.html -->
<ul>
    {% for bot in bots %}
        <li>
            <a href="{% url 'bot_detail' bot.id
%}">{{ bot.name }}</a>
        </li>
    {% endfor %}
</ul>
```

Tag `{% url 'bot_detail' bot.id %}` akan menghasilkan tautan yang sesuai dengan URL yang telah didefinisikan dalam file `urls.py` dengan nama `bot_detail`.

Sistem URL routing di Django menyediakan cara yang sangat fleksibel dan mudah untuk mengelola rute dalam aplikasi web. Dengan menggunakan pola URL yang dinamis, pengembang dapat dengan mudah menangani berbagai permintaan dan menerus-

Pengenalan Django

kannya ke view yang tepat. Django juga memungkinkan penggunaan nama URL untuk membuat tautan yang lebih terkelola dan terstruktur, sehingga memudahkan pemeliharaan dan pengembangan aplikasi di masa mendatang.

1.6.4 Views

Views adalah salah satu komponen terpenting dalam arsitektur Django. Pada dasarnya, views bertindak sebagai penghubung antara logika aplikasi dan apa yang ditampilkan kepada pengguna di browser. Saat pengguna melakukan permintaan HTTP ke aplikasi, Django akan memprosesnya melalui URL routing, lalu permintaan tersebut akan diteruskan ke view yang relevan. Fungsi view ini kemudian bertanggung jawab untuk menangani logika bisnis dan menghasilkan respons yang sesuai, yang bisa berupa halaman HTML, file JSON, file teks, atau bahkan file gambar.

Cara Kerja Views

View pada Django umumnya ditulis dalam bentuk fungsi atau kelas. Sebuah fungsi view menerima permintaan HTTP, melakukan berbagai operasi (misalnya, mengambil data dari database atau memproses form), dan kemudian menghasilkan respons yang ditampilkan kepada pengguna. Views juga bisa diatur untuk memproses berbagai tipe request, seperti GET dan POST, untuk menangani tindakan berbeda pada URL yang sama.

Contoh sederhana fungsi view bisa dilihat seperti berikut:

```
# bot_app/views.py
from django.shortcuts import render
from .models import Bot
```

```
def bot_list(request):  
    bots = Bot.objects.all() # Mengambil semua  
    data bot dari database  
    return render(request,  
    'bot_app/bot_list.html', {'bots': bots})
```

Dalam contoh di atas, view `bot_list` menerima request dari pengguna. Pertama, kita mengambil semua objek bot dari database menggunakan `Bot.objects.all()`. Kemudian, kita menggunakan fungsi `render()` untuk merender template `bot_list.html` sambil mengirimkan data bot tersebut ke template sebagai konteks.

Komentar:

- ***from django.shortcuts import render:*** Kita menggunakan `render()` untuk menggabungkan template dengan data yang akan dikirimkan ke halaman HTML.
- ***Bot.objects.all():*** Ini adalah query ke model `Bot` yang mengambil semua objek bot dari database.

Jenis-Jenis Views

Django mendukung dua jenis views: ***function-based views (FBVs)*** dan ***class-based views (CBVs)***. Keduanya memiliki tujuan yang sama tetapi dengan pendekatan yang berbeda.

Function-Based Views (FBVs)

FBV adalah pendekatan yang lebih sederhana dan langsung, di mana setiap view adalah sebuah fungsi Python. Contoh di atas

Pengenalan Django

adalah FBV, yang mengeksekusi logika bisnis di dalam satu fungsi.

Keuntungan utama FBV adalah kesederhanaannya. Namun, jika aplikasi semakin kompleks dan kita memiliki view yang membutuhkan perilaku yang mirip, menggunakan class-based views bisa lebih efektif karena mendukung penggunaan kembali logika umum.

Class-Based Views (CBVs)

CBV memungkinkan kita untuk mendefinisikan views sebagai kelas. Pendekatan ini lebih modular dan mendukung inheritance (pewarisan), sehingga cocok untuk mengurangi duplikasi kode di berbagai views. Django menyediakan beberapa CBV bawaan yang umum digunakan, seperti `ListView`, `DetailView`, `CreateView`, dan `UpdateView`.

Sebagai contoh, mari kita ubah `bot_list` menjadi class-based view:

```
# bot_app/views.py
from django.views.generic import ListView
from .models import Bot

class BotListView(ListView):
    model = Bot
    template_name = 'bot_app/bot_list.html'
    context_object_name = 'bots'
```

Dalam CBV ini, kita menggunakan kelas `ListView` yang disediakan oleh Django untuk menampilkan daftar objek dari model

Pengenalan Django

Bot. Kita hanya perlu menentukan model, template, dan nama konteks untuk template. Django secara otomatis menangani pengambilan data dan perenderan template tanpa memerlukan logika tambahan yang biasanya ditulis secara manual dalam FBV.

Menggunakan Views untuk Menangani Berbagai Tipe Request

Salah satu fitur menarik dari views di Django adalah kemampuan untuk menangani request GET dan POST di dalam view yang sama. Ini sangat berguna, misalnya, dalam kasus form submission, di mana GET digunakan untuk menampilkan form, sedangkan POST digunakan untuk memproses data yang dikirimkan oleh pengguna.

Contoh di bawah menunjukkan bagaimana sebuah view dapat menangani request GET dan POST secara bersamaan:

```
# bot_app/views.py
from django.shortcuts import render, redirect
from .forms import BotForm

def bot_create(request):
    if request.method == 'POST':
        form = BotForm(request.POST)
        if form.is_valid():
            form.save() # Menyimpan bot baru ke
            database
            return redirect('bot_list') #
            Redirect ke halaman daftar bot
        else:
            form = BotForm()
            return render(request,
                'bot_app/bot_create.html', {'form': form})
```

Pengenalan Django

Dalam view `bot_create` ini, jika metode request adalah POST, maka form yang dikirimkan oleh pengguna diproses. Jika valid, data akan disimpan ke database dan pengguna akan diarahkan kembali ke daftar bot. Jika request-nya GET, form kosong akan ditampilkan di halaman untuk diisi oleh pengguna.

Memisahkan Views untuk Tujuan yang Berbeda

Penting untuk memisahkan logika views berdasarkan tindakan yang berbeda. Misalnya, view yang digunakan untuk menampilkan daftar objek mungkin berbeda dari view yang digunakan untuk membuat atau mengedit objek. Django sangat fleksibel dalam hal ini, memungkinkan kita untuk membangun logika yang kompleks dengan cara yang terstruktur.

Contoh di bawah adalah struktur umum yang memisahkan views untuk operasi berbeda pada objek bot:

```
# bot_app/views.py
def bot_detail(request, id):
    bot = Bot.objects.get(id=id) # Mengambil
    data bot berdasarkan ID
    return render(request,
        'bot_app/bot_detail.html', {'bot': bot})

def bot_update(request, id):
    bot = Bot.objects.get(id=id)
    if request.method == 'POST':
        form = BotForm(request.POST,
            instance=bot)
        if form.is_valid():
            form.save()
            return redirect('bot_detail', id=id)
    else:
```

```
form = BotForm(instance=bot)
return render(request,
'bot_app/bot_update.html', {'form': form})
```

Di sini, `bot_detail` digunakan untuk menampilkan detail bot tertentu, sedangkan `bot_update` digunakan untuk mengedit bot. Keduanya memisahkan logika untuk menjaga kerapian dan modularitas kode.

Views di Django adalah jembatan antara permintaan pengguna dan hasil yang akan mereka lihat. Melalui views, kita dapat mengatur logika aplikasi dan menentukan respons yang dikirimkan kembali ke pengguna. Dengan kemampuan untuk menggunakan function-based views dan class-based views, Django menawarkan fleksibilitas dalam penulisan logika yang sesuai dengan kebutuhan proyek. Views juga bisa diatur untuk menangani berbagai tipe permintaan, memproses form, dan mengelola interaksi pengguna dengan aplikasi web secara efisien.

1.6.5 Models

Models adalah salah satu komponen kunci dalam Django yang digunakan untuk *menangani* interaksi dengan database. Django menggunakan pendekatan Object-Relational Mapping (ORM), yang memungkinkan pengembang untuk berinteraksi dengan database menggunakan objek Python, bukan harus menulis perintah SQL secara langsung. Model dalam Django berperan sebagai representasi dari tabel di dalam database, di mana setiap atribut model berhubungan dengan kolom pada tabel tersebut.

Pengenalan Django

Sebuah model dalam Django didefinisikan sebagai sebuah kelas yang mewarisi dari `django.db.models.Model`. Setiap atribut kelas tersebut akan menjadi sebuah kolom dalam tabel database. Django secara otomatis akan membuat tabel berdasarkan model yang kita definisikan ketika kita menjalankan migrasi database.

Sebagai contoh, mari kita lihat bagaimana kita dapat mendefinisikan model `Bot` untuk aplikasi `platform_bot`:

```
# bot_app/models.py
from django.db import models

class Bot(models.Model):
    name = models.CharField(max_length=100) # Kolom untuk nama bot
    description = models.TextField() # Kolom untuk deskripsi bot
    created_at = models.DateTimeField(auto_now_add=True) # Waktu dibuat
    updated_at = models.DateTimeField(auto_now=True) # Waktu diperbarui

    def __str__(self):
        return self.name # Mengembalikan nama bot sebagai representasi string
```

Komentar:

- **`models.CharField(max_length=100)`**: Mendefinisikan kolom untuk menyimpan teks dengan panjang maksimum 100 karakter.

Pengenalan Django

- **`models.TextField()`**: Digunakan untuk menyimpan teks yang lebih panjang seperti deskripsi.
- **`models.DateTimeField(auto_now_add=True)`**: Menyimpan waktu saat objek pertama kali dibuat.
- **`models.DateTimeField(auto_now=True)`**: Menyimpan waktu saat objek terakhir kali diperbarui.
- **`__str__`**: Metode ini mendefinisikan bagaimana objek Bot akan direpresentasikan sebagai string ketika diakses di antarmuka admin atau di shell.

Menggunakan ORM untuk Berinteraksi dengan Database

Dengan model yang sudah didefinisikan, Django ORM memungkinkan kita untuk melakukan operasi database seperti menyimpan, memperbarui, menghapus, dan mengambil data dengan menggunakan objek Python. Django akan menerjemahkan operasi ini menjadi query SQL yang tepat di belakang layar.

Sebagai contoh, untuk menambahkan entri baru ke dalam tabel Bot, kita bisa menggunakan model Bot seperti berikut:

```
# Menambahkan data bot baru
bot = Bot(name="Bot Baru",
description="Deskripsi bot baru")
bot.save() # Menyimpan ke database
```

Dalam kode di atas, kita membuat objek Bot dengan nama dan deskripsi tertentu. Setelah itu, kita memanggil metode `save()` untuk menyimpan objek ini ke dalam database.

Query dengan Django ORM

ORM Django tidak hanya memungkinkan kita untuk menambah data, tetapi juga untuk mengambil dan memanipulasi data dari database dengan cara yang sangat efisien. Misalnya, kita dapat mengambil semua bot yang ada di database dengan cara berikut:

```
# Mengambil semua bot
all_bots = Bot.objects.all()
```

Untuk query yang lebih spesifik, Django ORM menyediakan berbagai metode filter yang dapat digunakan untuk mengambil data berdasarkan kriteria tertentu:

```
# Mengambil bot yang namanya 'Bot Baru'
bot = Bot.objects.get(name="Bot Baru")
```

Metode `get()` akan mengembalikan satu objek yang memenuhi kriteria yang ditentukan, atau akan memunculkan kesalahan jika tidak ada objek yang ditemukan. Jika kita menginginkan banyak objek yang memenuhi kriteria tertentu, kita dapat menggunakan `filter()`:

```
# Mengambil semua bot yang namanya mengandung
kata 'Bot'
bots = Bot.objects.filter(name__contains="Bot")
```

Dalam contoh di atas, `filter()` akan mengembalikan QuerySet yang berisi semua bot yang namanya mengandung kata 'Bot'. Django ORM mendukung berbagai jenis query yang bisa digunakan untuk membuat filter yang kompleks dan efisien.

Pengenalan Django

Migrasi Database

Setiap kali kita membuat atau mengubah model, kita perlu menjalankan migrasi database untuk mencerminkan perubahan tersebut dalam struktur tabel di database. Django menyediakan manajemen migrasi yang sangat mudah. Setelah kita mendefinisikan model seperti `Bot`, kita dapat menjalankan perintah migrasi dengan menggunakan terminal:

```
$ python manage.py makemigrations bot_app
```

Perintah ini akan menghasilkan file migrasi berdasarkan perubahan yang dilakukan pada model di dalam aplikasi `bot_app`. Selanjutnya, kita dapat menerapkan migrasi tersebut ke database dengan perintah:

```
$ python manage.py migrate
```

Dengan perintah ini, Django akan membuat tabel `Bot` di database sesuai dengan definisi model.

Hubungan Antar Model

Django ORM juga mendukung definisi hubungan antar model, seperti hubungan satu-ke-banyak, banyak-ke-banyak, dan satu-ke-satu. Ini memungkinkan kita untuk membuat struktur data yang lebih kompleks. Sebagai contoh, jika kita ingin setiap `Bot` memiliki satu pemilik (user), kita bisa menambahkan hubungan `ForeignKey` ke dalam model `Bot`:

```
# bot_app/models.py
from django.contrib.auth.models import User
```



```
class Bot(models.Model):
    name = models.CharField(max_length=100)
    description = models.TextField()
    owner = models.ForeignKey(User,
on_delete=models.CASCADE) # Relasi ke model
User
    created_at =
models.DateTimeField(auto_now_add=True)
    updated_at =
models.DateTimeField(auto_now=True)

    def __str__(self):
        return self.name
```

Dengan menambahkan `ForeignKey` ke model `User`, setiap bot sekarang akan terkait dengan satu pemilik. Relasi ini memungkinkan kita untuk dengan mudah mendapatkan informasi terkait antara dua model.

Models dalam Django merupakan komponen yang sangat penting karena memungkinkan kita untuk berinteraksi dengan database dengan cara yang efisien dan intuitif. Dengan dukungan ORM yang kuat, pengembang dapat menulis kode yang mudah dipahami dan dirawat, tanpa perlu khawatir tentang detail teknis SQL. Models juga memungkinkan kita untuk mengelola migrasi database secara otomatis, menjaga sinkronisasi antara model aplikasi dan struktur tabel di database.

1.6.6 Forms

Forms dalam Django adalah komponen penting yang digunakan untuk menangani input dari pengguna. Dalam banyak aplikasi

Pengenalan Django

web, form digunakan untuk menerima data dari pengguna, baik itu untuk proses pendaftaran, login, mengirim pesan, atau melakukan pencarian. Django menyediakan sistem form yang terintegrasi dengan baik untuk menangani input ini, termasuk validasi data, pengelolaan error, dan penyajian form di antarmuka.

Django Forms adalah kelas Python yang digunakan untuk merepresentasikan form HTML dan menangani validasi data. Kita bisa membuat form secara manual atau menggunakan Model Forms yang secara otomatis menghasilkan form berdasarkan model yang sudah ada.

Membuat Form dengan Django

Ada dua cara utama untuk membuat form di Django: secara manual dengan mendefinisikan form dari awal, atau menggunakan `ModelForm` untuk form yang berdasarkan pada model. Pertama, mari kita lihat bagaimana membuat form secara manual.

Contoh, kita ingin membuat form untuk pengguna menambahkan bot baru di aplikasi `platform_bot`. Kita bisa mendefinisikan sebuah form dalam file `forms.py` sebagai berikut:

```
# bot_app/forms.py
from django import forms

class BotForm(forms.Form):
    name = forms.CharField(max_length=100) #
    # Input untuk nama bot
    description =
    forms.CharField(widget=forms.Textarea) # Input
    # Input untuk deskripsi bot
```

Pengenalan Django

Komentar:

- **`forms.CharField`**: Ini adalah field untuk menerima input berupa teks. `max_length=100` memastikan panjang maksimal input adalah 100 karakter.
- **`widget=forms.Textarea`**: Mengganti input teks biasa menjadi textarea, yang lebih cocok untuk menerima teks panjang seperti deskripsi.

Form ini bisa kita tampilkan dalam template dan dihubungkan dengan view untuk menangani input dari pengguna.

Menggunakan ModelForm

Jika form kita berbasis model, lebih efisien untuk menggunakan `ModelForm`, di mana Django akan secara otomatis menghasilkan form berdasarkan model yang kita buat. Contoh, kita bisa membuat form untuk model `Bot` seperti ini:

```
# bot_app/forms.py
from django import forms
from .models import Bot

class BotForm(forms.ModelForm):
    class Meta:
        model = Bot # Model yang digunakan
        # sebagai basis form
        fields = ['name', 'description'] #
        # Field yang ingin kita tampilkan dalam form
```

Komentar:

- **`model = Bot`**: Menentukan bahwa form ini berdasarkan model `Bot`.

Pengenalan Django

- **`fields = ['name', 'description']`**: Menunjukkan field yang ingin kita masukkan dalam form. Dalam hal ini, kita hanya menggunakan `name` dan `description`.

Dengan `ModelForm`, Django secara otomatis akan menghasilkan form HTML yang sesuai dengan model dan field yang kita tentukan, serta menangani validasi dasar seperti batas panjang teks.

Menampilkan Form dalam Template

Setelah form didefinisikan, langkah berikutnya adalah menampilkannya dalam template. Kita bisa menyertakan form ini dalam template HTML dengan memanfaatkan template tags Django. Misalnya, kita ingin menampilkan form untuk menambahkan bot di halaman `add_bot.html`:

```
<!-- templates/add_bot.html -->
<form method="post">
    {% csrf_token %}
    {{ form.as_p }} <!-- Menampilkan form dalam
elemen <p> -->
    <button type="submit">Simpan</button>
</form>
```

Komentar:

- **`{% csrf_token %}`**: Digunakan untuk melindungi form dari serangan CSRF (Cross-Site Request Forgery).
- **`{{ form.as_p }}`**: Template tag ini merender form sebagai paragraf HTML (`<p>`), yang berarti setiap field

Pengenalan Django

akan dikelilingi oleh tag `<p>`. Django juga menyediakan metode lain seperti `{{ form.as_table }}` dan `{{ form.as_ul }}` untuk merender form dalam bentuk tabel atau daftar tak berurut.

Menangani Input Form di Views

Setelah form di-submit, Django memerlukan logika untuk menangani data yang masuk. Logika ini biasanya didefinisikan di views. Berikut adalah contoh bagaimana kita bisa memproses form `BotForm` di dalam view:

```
# bot_app/views.py
from django.shortcuts import render, redirect
from .forms import BotForm

def add_bot(request):
    if request.method == 'POST': # Jika form
di-submit
        form = BotForm(request.POST)
        if form.is_valid(): # Memeriksa
validasi data
            # Menangani data form yang sudah
divalidasi
            # Dalam kasus ini, kita hanya
mencetak data ke terminal
            print(form.cleaned_data)
            return redirect('manage_bots') #
Mengarahkan pengguna setelah sukses
        else:
            form = BotForm() # Jika request adalah
GET, tampilkan form kosong

            return render(request, 'add_bot.html',
{'form': form}) # Render template dengan form
```

Pengenalan Django

Komentar:

- **`request.method == 'POST'`**: Mengecek apakah form telah di-submit.
- **`form.is_valid()`**: Memeriksa apakah data yang di-submit sesuai dengan semua aturan validasi form.
- **`form.cleaned_data`**: Berisi data yang sudah divalidasi dan siap digunakan.
- **`redirect('manage_bots')`**: Mengarahkan pengguna ke halaman lain setelah form berhasil diproses, dalam hal ini, ke halaman `manage_bots`.

Validasi Form

Django Forms menyediakan mekanisme validasi otomatis. Misalnya, jika kita menetapkan `max_length=100` pada field `name`, Django akan secara otomatis memunculkan error jika pengguna memasukkan lebih dari 100 karakter. Namun, kita juga bisa menambahkan validasi khusus sesuai kebutuhan kita dengan menulis metode khusus dalam form.

Sebagai contoh, jika kita ingin memastikan bahwa nama bot harus unik, kita bisa menambahkan validasi seperti ini:

```
# bot_app/forms.py
from django import forms
from .models import Bot

class BotForm(forms.ModelForm):
    class Meta:
        model = Bot
        fields = ['name', 'description']
```

```
def clean_name(self):
    name = self.cleaned_data.get('name')
    if
Bot.objects.filter(name=name).exists():
        raise forms.ValidationError("Nama
bot sudah digunakan.")
    return name
```

Dalam kode di atas, metode `clean_name` akan dipanggil selama proses validasi. Jika nama bot sudah ada di database, form akan memunculkan error, mencegah pengguna untuk memasukkan nama yang sama.

Django Forms adalah alat yang sangat kuat dan fleksibel untuk menangani input dari pengguna. Dengan sistem validasi yang kuat dan integrasi erat dengan model melalui `ModelForm`, form di Django menjadi lebih mudah dikelola dan diproses. Ini membuat proses penanganan input pengguna lebih aman dan efisien, serta menghemat waktu pengembangan dengan banyaknya fitur bawaan yang dapat langsung digunakan tanpa perlu menulis logika manual yang rumit.

1.6.7 Static Files dan Media Files

Dalam pengembangan aplikasi web, kita sering kali perlu menangani file statis seperti CSS, JavaScript, atau gambar, serta file media yang diunggah oleh pengguna, seperti foto profil, video, atau dokumen lainnya. Django menyediakan mekanisme yang sangat fleksibel dan terstruktur untuk mengelola kedua jenis file ini: **Static Files** dan **Media Files**.

Static Files

Static files adalah file yang tidak berubah selama aplikasi berjalan, seperti file CSS, JavaScript, dan gambar yang digunakan untuk antarmuka pengguna. Django memiliki kerangka kerja bawaan untuk menangani static files, yang memudahkan kita untuk menyimpan, mengatur, dan menyajikan file statis di aplikasi.

Untuk menangani static files, pertama-tama kita perlu menentukan direktori tempat file-file ini akan disimpan. Pada umumnya, kita mendefinisikan folder `static/` di dalam setiap aplikasi. Kita juga perlu menambahkan konfigurasi dalam file `settings.py` untuk memberitahu Django di mana mencari file statis.

Konfigurasi dalam `settings.py` untuk static files adalah sebagai berikut:

```
# settings.py
STATIC_URL = '/static/' # URL dasar untuk file
statis
STATICFILES_DIRS = [
    BASE_DIR / 'static', # Direktori tempat
    kita menyimpan file statis
]
```

Komentar:

- **STATIC_URL**: URL dasar tempat file statis dapat diakses. Biasanya `/static/`, artinya jika ada file CSS di `static/css/style.css`, file tersebut akan diakses melalui URL `/static/css/style.css`.

Pengenalan Django

- ***STATICFILES_DIRS***: Daftar direktori tambahan tempat Django akan mencari file statis selain dari setiap aplikasi.

Setelah ini diatur, kita dapat menyimpan file CSS, JavaScript, atau gambar dalam folder `static/` di dalam proyek atau aplikasi tertentu. Misalnya, dalam aplikasi `bot_app`, kita dapat memiliki struktur direktori sebagai berikut:

```
bot_app/  
  static/  
    bot_app/  
      css/  
      js/  
      img/
```

File CSS atau JavaScript akan di-load di template menggunakan template tag `{% static %}`. Misalnya, untuk memuat file `style.css` dalam template, kita bisa menulis:

```
<!-- templates/base.html -->  
<link rel="stylesheet" type="text/css" href="{% static 'bot_app/css/style.css' %}">
```

Media Files

Media files adalah file yang diunggah oleh pengguna, seperti foto profil, dokumen, video, atau file lainnya. Django juga menyediakan cara mudah untuk menangani file media ini dengan menggunakan ***MEDIA_URL*** dan ***MEDIA_ROOT***.

Dalam `settings.py`, kita perlu mendefinisikan di mana file media akan disimpan di sistem file (dalam variabel ***MEDIA_ROOT***).

Pengenalan Django

OT) dan bagaimana mereka akan diakses melalui URL (dalam variabel `MEDIA_URL`).

```
# settings.py
MEDIA_URL = '/media/' # URL dasar untuk file
media
MEDIA_ROOT = BASE_DIR / 'media' # Direktori
tempat file media akan disimpan
```

Komentar:

- ***MEDIA_URL***: URL dasar untuk mengakses file media yang diunggah pengguna. Misalnya, jika ada file gambar yang diunggah dan disimpan di `media/images/avatar.jpg`, file tersebut dapat diakses melalui URL `/media/images/avatar.jpg`.
- ***MEDIA_ROOT***: Path di mana semua file media akan disimpan di sistem file. Kita biasanya menyimpan file ini dalam folder `media/` di dalam proyek.

Untuk menyimpan file media di aplikasi, kita perlu mengonfigurasi model kita agar mendukung file upload. Misalnya, kita ingin pengguna mengunggah gambar avatar ketika mereka mendaftarkan bot di aplikasi kita. Kita bisa menambahkan field gambar dalam model `Bot` seperti ini:

```
# bot_app/models.py
from django.db import models

class Bot(models.Model):
    name = models.CharField(max_length=100) #
    Nama bot
```

Pengenalan Django

```
description = models.TextField() #  
Deskripsi bot  
avatar =  
models.ImageField(upload_to='avatars/') # Field  
untuk menyimpan gambar avatar
```

Komentar:

- ***ImageField***: Field ini digunakan untuk menyimpan file gambar di database. Argumen `upload_to='avatars/'` menunjukkan bahwa semua file gambar yang diunggah akan disimpan di folder `media/avatars/`.

Untuk menampilkan dan mengakses file media dalam template, kita dapat menggunakan URL yang dihasilkan oleh `MEDIA_URL`:

```
<!-- templates/bot_detail.html -->  

```

Komentar:

- ***bot.avatar.url***: Ini menghasilkan URL lengkap dari file gambar yang diunggah untuk ditampilkan di halaman.

Serving File Statis dan Media di Development

Dalam lingkungan pengembangan, Django tidak secara otomatis melayani file statis dan media. Kita perlu menambahkan sedikit konfigurasi dalam `urls.py` agar file-file ini bisa diakses selama pengembangan.

Pengenalan Django

Dalam `urls.py`, tambahkan berikut ini:

```
# project/urls.py
from django.conf import settings
from django.conf.urls.static import static

urlpatterns = [
    # URL patterns lainnya
]

if settings.DEBUG:
    urlpatterns += static(settings.MEDIA_URL,
        document_root=settings.MEDIA_ROOT) # Serving
    file media
    urlpatterns += static(settings.STATIC_URL,
        document_root=settings.STATICFILES_DIRS[0]) #
    Serving file statis
```

Komentar:

- **`static()`**: Fungsi ini digunakan untuk melayani file statis dan media di environment development. Dalam produksi, kita akan menggunakan server web seperti Nginx atau Apache untuk melayani file-file ini.

Dengan adanya dukungan bawaan untuk static files dan media files, Django membuat pengelolaan file di aplikasi web menjadi jauh lebih mudah. Semua file yang bersifat statis bisa dikelola dengan baik di dalam direktori `static/`, dan file media pengguna dapat disimpan serta diakses dengan mekanisme `MEDIA_URL` dan `MEDIA_ROOT`. Ini memberikan fleksibilitas besar dalam pengembangan antarmuka yang interaktif, sambil memastikan file-file pengguna dapat diakses dan diolah dengan aman.

1.6.8 Middleware

Dalam arsitektur Django, **middleware** adalah komponen penting yang bekerja di antara request dan response. Secara sederhana, middleware adalah lapisan yang memungkinkan kita untuk memproses data dari request sebelum mencapai view, atau memodifikasi response sebelum dikirim kembali ke klien.

Middleware memberikan kontrol tambahan terhadap bagaimana permintaan HTTP diproses, memberikan kita kesempatan untuk menerapkan berbagai fungsi seperti otentikasi, logging, modifikasi response, atau bahkan menghalangi permintaan tertentu.

Fungsi Dasar Middleware:

Middleware dalam Django berfungsi sebagai hook yang dieksekusi secara berurutan setiap kali permintaan HTTP diterima oleh server. Setiap middleware dapat mengakses request sebelum mencapai view dan juga response setelah view diproses. Ini memungkinkan kita untuk menyisipkan logika yang berjalan baik pada tahap awal (misalnya, otentikasi) atau tahap akhir (misalnya, pengaturan header tertentu).

Dalam proses alur kerja biasa, Django melakukan hal-hal berikut:

1. Menerima request dari klien.
2. Melewati request tersebut melalui berbagai middleware yang telah kita tentukan.
3. Mengirimkan request yang telah diproses ke view yang relevan.

Pengenalan Django

4. Setelah view menghasilkan response, response tersebut kembali diproses melalui middleware sebelum dikirimkan ke klien.

Alur ini membuat middleware sangat fleksibel dan bermanfaat dalam berbagai skenario.

Contoh Penggunaan Middleware:

Misalnya, kita ingin membuat middleware sederhana yang mencatat setiap permintaan HTTP yang masuk ke aplikasi kita. Kita bisa membuat middleware khusus seperti ini:

```
# project/middleware.py
import logging

logger = logging.getLogger(__name__)

class LogRequestMiddleware:
    def __init__(self, get_response):
        self.get_response = get_response

    def __call__(self, request):
        # Log request method dan path
        logger.info(f"Request Method:
{request.method}, Path: {request.path}")

        # Melanjutkan ke view berikutnya
        response = self.get_response(request)

        # Mengembalikan response yang sudah
        diproses
        return response
```

Komentar:

Pengenalan Django

- Middleware ini dibuat di file `middleware.py` yang ada di dalam folder proyek.
- `__call__()`: Ini adalah metode utama yang dieksekusi setiap kali ada request HTTP. Di sini kita bisa menambahkan logika tambahan, seperti mencatat informasi tentang request yang masuk.
- `self.get_response(request)` melanjutkan request ke middleware berikutnya atau langsung ke view.

Setelah middleware ini dibuat, kita perlu mendaftarkannya di file `settings.py` agar Django mengenali dan menjalankannya:

```
# settings.py
MIDDLEWARE = [

    'django.middleware.security.SecurityMiddleware',

    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',

    'django.contrib.auth.middleware.AuthenticationMiddleware',

    'django.contrib.messages.middleware.MessageMiddleware',

    'django.middleware.clickjacking.XFrameOptionsMiddleware',

    # Middleware custom kita
    'project.middleware.LogRequestMiddleware',
]
```

Pengenalan Django

Komentar:

- Dalam `settings.py`, middleware kita ditambahkan ke urutan `MIDDLEWARE`, sehingga Django akan menjalankannya bersamaan dengan middleware bawaan lainnya.

Jenis-Jenis Middleware Bawaan Django:

Django juga menyediakan sejumlah middleware bawaan yang sangat berguna dalam mengelola berbagai aspek dari request dan response. Beberapa middleware yang sering digunakan meliputi:

1. ***SecurityMiddleware***: Menerapkan praktik keamanan seperti Strict-Transport-Security untuk memastikan semua request menggunakan HTTPS.
2. ***SessionMiddleware***: Menyediakan dukungan untuk session management, yang memungkinkan kita melacak pengguna selama mereka berinteraksi dengan aplikasi.
3. ***CsrfViewMiddleware***: Mengaktifkan perlindungan terhadap CSRF (Cross-Site Request Forgery) untuk setiap request yang melibatkan form.
4. ***AuthenticationMiddleware***: Mengelola autentikasi pengguna dengan mengaitkan pengguna yang terautentikasi dengan setiap request.

Dengan middleware bawaan ini, Django memberikan banyak fitur keamanan dan kemudahan dalam menangani request tanpa kita perlu menulis kode tambahan.

Keuntungan Menggunakan Middleware:

Pengenalan Django

Beberapa keuntungan menggunakan middleware di Django adalah:

- **Modularitas:** Middleware memungkinkan kita memisahkan logika yang berbeda ke dalam komponen-komponen yang mudah dikelola. Misalnya, kita dapat menempatkan semua logika terkait autentikasi di satu middleware, dan semua logika terkait logging di middleware lain.
- **Fleksibilitas:** Dengan middleware, kita dapat melakukan berbagai modifikasi terhadap request dan response tanpa perlu mengganggu view atau logika utama aplikasi. Hal ini sangat berguna untuk menambahkan fitur keamanan, logging, caching, atau manipulasi header.
- **Pengelolaan Permintaan:** Middleware memberi kita kontrol penuh terhadap alur permintaan, memungkinkan kita memblokir atau memodifikasi permintaan sebelum sampai ke view, atau memanipulasi response yang dihasilkan.

Middleware adalah alat yang sangat penting dalam arsitektur Django, memungkinkan kita untuk menangani berbagai kebutuhan aplikasi tanpa mengganggu logika inti. Baik kita ingin menambahkan otentikasi, melakukan logging, atau melindungi aplikasi dari serangan CSRF, middleware menyediakan solusi yang elegan dan terintegrasi dengan baik di dalam framework Django.

1.6.9 Signals

Signals dalam Django adalah fitur yang memungkinkan kita untuk menerapkan pemrograman berbasis event. Dengan menggunakan signals, kita dapat membuat komponen aplikasi bereaksi

Pengenalan Django

secara otomatis terhadap event tertentu yang terjadi di dalam sistem. Ini sangat berguna ketika kita ingin mengeksekusi kode tambahan sebagai respon terhadap perubahan dalam model, user action, atau kejadian lain di aplikasi tanpa mengganggu logika utama.

Konsep Signals

Signals di Django berfungsi untuk menghubungkan event dengan respons. Ketika suatu event terjadi, signal akan "mengirim" pemberitahuan, dan fungsi yang terhubung ke signal tersebut akan dijalankan secara otomatis. Hal ini memungkinkan kita untuk memisahkan logika event-driven dari kode utama aplikasi, menciptakan sistem yang lebih modular dan mudah dikelola.

Beberapa signal yang sering digunakan dalam Django adalah:

- ***pre_save*** dan ***post_save***: Dipicu sebelum dan sesudah model disimpan ke database.
- ***pre_delete*** dan ***post_delete***: Dipicu sebelum dan sesudah model dihapus dari database.
- ***request_started*** dan ***request_finished***: Dipicu pada awal dan akhir siklus request.
- ***user_logged_in*** dan ***user_logged_out***: Dipicu saat pengguna login atau logout.

Membuat dan Menghubungkan Signals:

Untuk memahami bagaimana signals bekerja, mari kita buat contoh signal sederhana yang akan mengirim notifikasi setiap kali objek User baru dibuat. Kita akan menggunakan signal

Pengenalan Django

`post_save` untuk menjalankan fungsi setelah model User berhasil disimpan.

```
# apps/users/signals.py
from django.db.models.signals import post_save
from django.contrib.auth.models import User
from django.dispatch import receiver

# Fungsi ini akan dipanggil setelah user
# berhasil disimpan
@receiver(post_save, sender=User)
def send_welcome_email(sender, instance,
                        created, **kwargs):
    if created:
        # Kirim email selamat datang ke pengguna
        # baru
        print(f"Selamat datang
        {instance.username}, terima kasih telah
        mendaftar.")
```

Komentar:

- File ini diletakkan di dalam folder `signals.py` di dalam aplikasi `users`.
- Fungsi `send_welcome_email` akan dijalankan setiap kali objek `User` baru disimpan ke database.
- **`@receiver(post_save, sender=User)`** menunjukkan bahwa signal ini hanya berlaku untuk model `User` dan akan dipicu setelah proses penyimpanan.

Untuk memastikan signal ini aktif, kita perlu mendaftarkannya. Biasanya, ini dilakukan di file `apps.py` aplikasi:

```
# apps/users/apps.py
```

Pengenalan Django

```
from django.apps import AppConfig

class UsersConfig(AppConfig):
    name = 'users'

    def ready(self):
        # Mendaftarkan signal saat aplikasi siap
        import users.signals
```

Komentar:

- File `apps.py` diubah untuk memastikan bahwa signal dimuat saat aplikasi `users` diinisialisasi oleh Django.
- **`ready()`** adalah metode khusus yang dipanggil saat aplikasi di-load, dan di sinilah kita mengimpor file `signals.py`.

Signals Bawaan Django

Django menyediakan berbagai signal bawaan yang berguna untuk banyak situasi. Beberapa contoh signal yang sering digunakan dalam pengembangan aplikasi adalah:

1. **`pre_save`**: Dipanggil sebelum objek disimpan. Signal ini berguna ketika kita ingin memodifikasi data sebelum ditulis ke database.
2. **`post_save`**: Dipanggil setelah objek disimpan. Signal ini cocok untuk mengeksekusi tugas-tugas tambahan, seperti mengirim notifikasi atau memperbarui data terkait.
3. **`pre_delete`**: Dipanggil sebelum objek dihapus. Kita dapat menggunakan signal ini untuk membersihkan data yang terhubung sebelum penghapusan terjadi.

Pengenalan Django

4. ***post_delete***: Dipanggil setelah objek dihapus, memungkinkan kita untuk menindaklanjuti event penghapusan, seperti menghapus file terkait.
5. ***request_started*** dan ***request_finished***: Berguna untuk mencatat atau memantau siklus hidup request HTTP.

Contoh Penggunaan Signal untuk Pembersihan Data:

Misalnya, kita memiliki model **Profile** yang terkait dengan **User**. Jika pengguna dihapus, kita ingin otomatis menghapus profile terkait. Kita bisa memanfaatkan signal ***pre_delete*** untuk tujuan ini.

```
# apps/users/signals.py
from django.db.models.signals import pre_delete
from django.contrib.auth.models import User
from django.dispatch import receiver
from .models import Profile

@receiver(pre_delete, sender=User)
def delete_user_profile(sender, instance,
    **kwargs):
    # Menghapus profil pengguna sebelum user
    dihapus
    Profile.objects.filter(user=instance).delete()
```

Komentar:

- Di sini, kita menggunakan signal ***pre_delete*** untuk memastikan bahwa ketika objek **User** dihapus, profile terkait juga dihapus dari database.

Pengenalan Django

- ***Profile.objects.filter(user=instance).delete()*** melakukan penghapusan semua objek *Profile* yang terkait dengan pengguna yang sedang dihapus.

Keuntungan Menggunakan Signals:

Signals memberikan beberapa keuntungan penting dalam arsitektur Django:

1. ***Pemrograman yang Modular***: Signals memungkinkan kita memisahkan logika event-driven dari logika utama aplikasi. Ini menciptakan sistem yang lebih bersih dan modular, sehingga lebih mudah dikelola dan di-debug.
2. ***Reduksi Kode Boilerplate***: Dengan menggunakan signals, kita dapat mengurangi kode yang berulang. Misalnya, jika kita perlu mengeksekusi tindakan tertentu setiap kali model disimpan, kita cukup mendefinisikan signal yang relevan daripada menulis kode tersebut di berbagai tempat.
3. ***Pengaturan Otomatis***: Signals sangat cocok untuk tugas-tugas otomatis, seperti mengirim email selamat datang, memperbarui data yang terhubung, atau melakukan logging event.

Signals adalah alat yang sangat kuat dalam Django, memungkinkan kita untuk menambahkan logika event-driven tanpa mengganggu struktur utama aplikasi. Dengan signals, kita dapat menangani berbagai kejadian, seperti penyimpanan atau penghapusan model, secara otomatis dan efisien. Penggunaan signals

memperkuat sifat modular dan fleksibel dari Django, memberikan pengembang lebih banyak kendali atas bagaimana event-event diproses dalam aplikasi.

1.7 Mengapa Memilih Django?

Django adalah salah satu framework web yang paling populer dan banyak digunakan di seluruh dunia. Tentu saja, ada beberapa alasan kuat mengapa banyak pengembang memilih Django sebagai fondasi untuk membangun aplikasi web mereka. Alasan-alasan ini berkaitan erat dengan berbagai fitur yang ditawarkan Django, serta kemampuannya untuk menyederhanakan pengembangan aplikasi web yang kompleks. Di bagian ini, kita akan menjelaskan keunggulan-keunggulan tersebut dan mengapa Django menjadi pilihan yang tepat bagi banyak pengembang web.

Cepat dan Efisien

Salah satu keunggulan utama Django adalah kecepatannya dalam pengembangan. Django dirancang untuk membantu pengembang mengubah ide menjadi produk jadi dengan cepat dan efisien. Ini karena Django menawarkan banyak alat dan fitur yang memudahkan pembuatan aplikasi web. Dari admin interface yang otomatis hingga ORM yang canggih, Django menyediakan segala yang dibutuhkan pengembang untuk membangun aplikasi web dengan waktu yang jauh lebih singkat dibandingkan membangun aplikasi dari awal.

Pengenalan Django

Di terminal, kita dapat membuat proyek Django baru dengan sangat cepat. Misalnya, untuk membuat proyek baru, kita cukup menjalankan perintah berikut:

```
$ django-admin startproject platform_bot
```

Komentar:

- Perintah ini membuat kerangka dasar proyek Django baru di direktori yang ditentukan. Django secara otomatis menyiapkan struktur folder, file konfigurasi, dan komponen penting lainnya.

Dengan struktur proyek yang sudah siap, kita bisa langsung memulai pengembangan tanpa perlu memikirkan setup awal yang rumit.

Keamanan Terintegrasi

Keamanan adalah salah satu prioritas utama dalam pengembangan aplikasi web. Django memiliki fitur keamanan yang terintegrasi langsung ke dalam framework, termasuk perlindungan terhadap berbagai jenis serangan seperti **SQL injection**, **cross-site scripting (XSS)**, **cross-site request forgery (CSRF)**, dan **click-jacking**. Fitur-fitur ini memungkinkan pengembang untuk fokus pada pengembangan fungsionalitas aplikasi tanpa harus khawatir tentang keamanan dasar.

Sebagai contoh, Django secara otomatis melindungi aplikasi dari serangan CSRF melalui penggunaan token CSRF dalam form yang digunakan untuk menerima input dari pengguna. Django se-

Pengenalan Django

cara otomatis menyisipkan token ini dalam form, sehingga kita tidak perlu menambahkannya secara manual.

```
<form method="POST">
    {% csrf_token %}
    <!-- Field form lainnya -->
</form>
```

Komentar:

- Di dalam template HTML, kita hanya perlu menambahkan **{% csrf_token %}** di dalam form untuk memastikan bahwa form dilindungi dari serangan CSRF.

Skala yang Mudah

Django dirancang dengan mempertimbangkan skalabilitas. Artinya, aplikasi yang dibangun dengan Django dapat dengan mudah dioptimalkan untuk menangani peningkatan beban lalu lintas pengguna. Baik untuk startup kecil maupun aplikasi enterprise yang besar, Django dapat menangani berbagai skala proyek dengan mudah. Django memiliki kemampuan untuk mendukung berbagai database, dan dapat dengan mudah dikonfigurasi untuk mendukung caching, load balancing, dan sistem distribusi lainnya yang diperlukan saat aplikasi berkembang.

Komunitas dan Dokumentasi yang Kuat

Salah satu aspek paling menguntungkan dari Django adalah komunitas penggunanya yang besar dan aktif. Komunitas ini menyediakan banyak sekali sumber daya, seperti tutorial, forum diskusi, dan library open-source yang bisa digunakan untuk menambah fungsionalitas aplikasi dengan mudah. Selain itu, doku-

Pengenalan Django

mentasi Django juga sangat lengkap dan mudah dipahami, bahkan untuk pengembang yang baru pertama kali menggunakan framework ini.

Jika kita menemui masalah selama pengembangan, kemungkinan besar solusinya sudah tersedia dalam dokumentasi resmi Django atau melalui komunitas yang aktif di berbagai platform seperti Stack Overflow dan GitHub.

Ekosistem yang Kaya dan Fitur Built-in

Django tidak hanya menyediakan alat-alat untuk pengembangan dasar, tetapi juga mencakup berbagai fitur built-in yang sangat membantu pengembang. Fitur-fitur seperti authentication, URL routing, form handling, serta admin interface yang otomatis membuat Django menjadi framework yang sangat lengkap.

Selain itu, Django juga memiliki library-library tambahan yang dapat diintegrasikan dengan mudah untuk menambah fungsionalitas tertentu, seperti Django REST Framework untuk membangun API.

Untuk mengatur URL routing, misalnya, kita cukup mendefinisikan pola URL dalam file ***urls.py***:

```
# apps/platform_bot/urls.py
from django.urls import path
from . import views

urlpatterns = [
    path('', views.home, name='home'),
```

```
path('about/', views.about, name='about'),  
]
```

Komentar:

- Di file **urls.py** ini, kita mendefinisikan URL yang akan memetakan request ke fungsi **home** dan **about** dalam **views.py**. Dengan cara ini, Django mempermudah proses routing dan menghubungkan URL dengan views.

Django Menawarkan "Batteries Included"

Salah satu filosofi inti dari Django adalah "batteries included", yang berarti Django menyertakan banyak alat dan fitur langsung di dalam frameworknya, sehingga kita tidak perlu mencari solusi eksternal untuk sebagian besar kebutuhan pengembangan. Misalnya, Django menawarkan ORM bawaan untuk mengelola interaksi dengan database, form handling untuk mempermudah validasi dan proses input pengguna, serta admin interface otomatis yang sangat berguna untuk mengelola data aplikasi tanpa menulis kode tambahan.

Dengan Django, banyak aspek pengembangan yang sudah terotomatiskan atau disediakan secara default, sehingga menghemat waktu dan tenaga pengembang dalam menyusun berbagai fitur dasar aplikasi web.

Django adalah framework yang sangat kuat, fleksibel, dan efisien untuk pengembangan aplikasi web. Kecepatan, keamanan, skalabilitas, serta komunitas yang besar menjadikan Django pilihan

Pengenalan Django

yang sangat menarik bagi banyak pengembang. Dengan banyaknya fitur built-in yang menyederhanakan proses pengembangan, Django memungkinkan pengembang untuk fokus pada pengembangan logika bisnis dan fungsionalitas unik dari aplikasi mereka, daripada terjebak pada hal-hal teknis dasar.

1.7.1 Keunggulan Django dalam Pengembangan Web

Django, sebagai framework web berbasis Python, dikenal karena kemampuannya dalam mempermudah pengembangan aplikasi web yang kompleks. Keunggulan Django dalam hal ini dapat dilihat dari berbagai fitur dan fungsionalitas yang ditawarkannya. Dalam sub-bab ini, kita akan menjelaskan lebih lanjut bagaimana Django mempermudah proses pengembangan aplikasi web dan mengapa framework ini menjadi pilihan populer di kalangan pengembang.

Struktur Proyek yang Terorganisir

Salah satu kekuatan utama Django adalah struktur proyeknya yang terorganisir dengan baik. Ketika kita memulai proyek Django baru, framework ini secara otomatis menghasilkan struktur folder dan file yang diperlukan. Struktur ini termasuk folder untuk aplikasi, file konfigurasi, dan file penting lainnya yang membuat pengembangan menjadi lebih terstruktur.

Sebagai contoh, saat kita membuat proyek baru menggunakan perintah:

```
$ django-admin startproject platform_bot
```

Pengenalan Django

Komentar:

- Perintah ini menghasilkan folder proyek utama **platform_bot** dengan sub-folder yang menyertakan **manage.py**, **platform_bot/settings.py**, **platform_bot/urls.py**, dan **platform_bot/wsgi.py**. Struktur ini memudahkan pengelolaan dan organisasi kode.

Dengan adanya struktur ini, pengembang dapat dengan cepat memahami dan bekerja pada bagian-bagian aplikasi tanpa perlu menghabiskan waktu menyiapkan kerangka dasar proyek.

Admin Interface Otomatis

Django menyediakan sebuah fitur luar biasa yang disebut admin interface otomatis. Fitur ini memungkinkan pengembang untuk mengelola data aplikasi secara langsung melalui antarmuka web yang dihasilkan secara otomatis. Admin interface ini sangat berguna untuk keperluan administratif seperti menambah, mengubah, dan menghapus data dari model aplikasi.

Untuk menggunakan admin interface, kita hanya perlu mendaftarkan model kita di **admin.py**. Sebagai contoh, jika kita memiliki model **Bot**, kita bisa mendaftarkannya seperti ini:

```
# apps/platform_bot/admin.py
from django.contrib import admin
from .models import Bot

admin.site.register(Bot)
```

Pengenalan Django

Komentar:

- Dengan menambahkan model **Bot** ke dalam admin site, Django secara otomatis menghasilkan antarmuka administratif yang memungkinkan pengguna untuk mengelola entitas **Bot** dari web browser.

Fitur ini sangat menghemat waktu pengembang, karena kita tidak perlu membuat antarmuka administratif dari awal. Cukup dengan sedikit konfigurasi, kita sudah mendapatkan alat yang sangat berguna.

Penggunaan ORM (Object-Relational Mapping)

Django menyediakan sistem ORM yang kuat untuk menangani interaksi dengan database. ORM memungkinkan kita untuk berinteraksi dengan database menggunakan objek Python daripada menulis query SQL secara langsung. Ini tidak hanya mempermudah penulisan dan pemeliharaan kode, tetapi juga meningkatkan keamanan aplikasi dengan mencegah serangan SQL injection.

Sebagai contoh, jika kita ingin mengambil semua entitas **Bot** dari database, kita bisa menggunakan kode berikut:

```
# apps/platform_bot/views.py
from django.shortcuts import render
from .models import Bot

def bot_list(request):
    bots = Bot.objects.all() # Mengambil semua
    # entitas Bot dari database
    return render(request, 'bot_list.html',
    {'bots': bots})
```

Pengenalan Django

Komentar:

- Dengan menggunakan **`Bot.objects.all()`**, kita dapat mengambil semua entitas **`Bot`** tanpa menulis query SQL secara manual. Ini mempermudah proses pengambilan data dan membuat kode lebih bersih.

Form Handling yang Mudah

Django juga menawarkan alat yang sangat membantu untuk menangani form dan validasi input pengguna. Dengan menggunakan **`forms.py`**, kita dapat mendefinisikan form, validasi, dan logika pemrosesan input dengan cara yang sangat terstruktur.

Sebagai contoh, berikut adalah form untuk membuat entitas **`Bot`**:

```
# apps/platform_bot/forms.py
from django import forms
from .models import Bot

class BotForm(forms.ModelForm):
    class Meta:
        model = Bot
        fields = ['name', 'description'] #
# Field yang akan ditampilkan di form
```

Komentar:

- Dengan menggunakan **`ModelForm`**, kita dapat secara otomatis menghasilkan form dari model **`Bot`**, termasuk validasi yang sesuai untuk field yang didefinisikan.

Sistem Routing yang Fleksibel

Pengenalan Django

Django memiliki sistem routing yang fleksibel dan kuat untuk menentukan pola URL dan menghubungkannya dengan views. Sistem ini memungkinkan kita untuk mengatur bagaimana aplikasi menangani berbagai permintaan URL, dan mempermudah pengembangan aplikasi dengan banyak endpoint.

Misalnya, untuk mengatur pola URL di ***urls.py***:

```
# apps/platform_bot/urls.py
from django.urls import path
from . import views

urlpatterns = [
    path('', views.home, name='home'),
    path('bot/<int:id>/', views.bot_detail,
name='bot_detail'),
]
```

Komentar:

- Dengan mendefinisikan pola URL di ***urls.py***, kita dapat menghubungkan URL tertentu dengan fungsi ***home*** dan ***bot_detail*** di ***views.py***, mempermudah penanganan request dan pengaturan URL.

Django mempermudah pengembangan aplikasi web yang kompleks melalui struktur proyek yang terorganisir, admin interface otomatis, sistem ORM yang kuat, form handling yang efisien, dan sistem routing yang fleksibel. Dengan fitur-fitur ini, Django memungkinkan pengembang untuk fokus pada pengembangan logika aplikasi dan fungsionalitas, sementara Django menangani aspek-aspek teknis yang kompleks. Keuntungan-keuntungan ini

menjadikan Django pilihan yang sangat menarik untuk membangun aplikasi web yang skalabel dan terkelola dengan baik.

1.7.2 Django vs Framework Lainnya

Dalam memilih framework untuk pengembangan web, banyak faktor yang harus dipertimbangkan, termasuk fitur, kemudahan penggunaan, dan ekosistem yang tersedia. Django adalah salah satu framework web yang populer, tetapi banyak pengembang juga mempertimbangkan framework lain seperti Flask, Ruby on Rails, dan Laravel. Dalam sub-bab ini, kita akan membandingkan Django dengan framework-framework ini, menyoroti kelebihan dan kekurangan masing-masing.

Django vs Flask

Flask adalah framework web berbasis Python yang sering dibandingkan dengan Django.

Berikut adalah beberapa perbedaan utama antara keduanya:

Pendekatan dan Fitur:

- ***Django:*** Django adalah framework "batteries-included," artinya ia menyediakan banyak fitur bawaan seperti admin interface, ORM, dan sistem templating. Ini memungkinkan pengembang untuk memulai dengan cepat tanpa memerlukan banyak konfigurasi tambahan.
- ***Flask:*** Flask, di sisi lain, adalah framework micro. Ini berarti Flask memberikan dasar yang sangat ringan dan membiarkan pengembang memilih dan mengintegrasikan berbagai komponen tambahan sesuai kebutuhan. Flask memberikan fleksibilitas lebih besar dalam memilih alat

Pengenalan Django

yang digunakan tetapi memerlukan lebih banyak konfigurasi manual.

Kelebihan Django:

- ***Fitur Bawaan:*** Dengan fitur seperti admin interface otomatis dan ORM, Django mempercepat pengembangan aplikasi dengan menyediakan alat-alat yang sudah siap pakai.
- ***Struktur Proyek:*** Django menyediakan struktur proyek yang terorganisir dengan baik, memudahkan pengelolaan kode dalam aplikasi besar.

Kelebihan Flask:

- ***Kebebasan dalam Pilihan:*** Flask memberikan kebebasan untuk memilih pustaka dan alat yang mereka inginkan, cocok untuk proyek dengan kebutuhan khusus.
- ***Ringan dan Sederhana:*** Flask lebih sederhana dan lebih ringan, membuatnya ideal untuk aplikasi kecil atau layanan mikro yang tidak memerlukan banyak fitur bawaan.

Django vs Ruby on Rails

Ruby on Rails (RoR) adalah framework web berbasis Ruby yang juga populer.

Berikut adalah beberapa perbandingan antara Django dan RoR:

Pendekatan dan Fitur:

- ***Django:*** Django menggunakan pendekatan MVT (Model-View-Template) dan menawarkan banyak fitur bawa-

Pengenalan Django

an untuk mempercepat pengembangan. Django juga dikenal dengan prinsip "Don't Repeat Yourself" (DRY), yang mendorong pengkodean yang efisien dan terstruktur.

- **Ruby on Rails:** RoR juga menggunakan prinsip DRY dan "Convention over Configuration," yang mengurangi kebutuhan konfigurasi dengan mengikuti konvensi tertentu. RoR menyediakan banyak fitur otomatis yang mempermudah pengembangan aplikasi web.

Kelebihan Django:

- **Pengelolaan Proyek:** Struktur proyek Django mempermudah pengelolaan dan organisasi kode.
- **Keamanan:** Django dikenal karena fitur keamanannya yang kuat, termasuk perlindungan terhadap serangan SQL injection, XSS, dan CSRF.

Kelebihan Ruby on Rails:

- **Kemudahan Penggunaan:** RoR terkenal karena kemudahan penggunaannya dan kemampuan untuk membuat aplikasi web dengan cepat, berkat konvensi yang sudah ditetapkan.
- **Ecosystem dan Komunitas:** RoR memiliki ekosistem gem yang kaya, menawarkan banyak pustaka dan alat untuk mempercepat pengembangan.

Django vs Laravel

Laravel adalah framework web berbasis PHP yang populer.

Berikut adalah beberapa perbandingan antara Django dan Laravel:

Pengenalan Django

Pendekatan dan Fitur:

- ***Django:*** Seperti disebutkan sebelumnya, Django menawarkan banyak fitur bawaan dan menggunakan Python, bahasa yang dikenal karena sintaksisnya yang bersih dan mudah dipahami.
- ***Laravel:*** Laravel juga menawarkan banyak fitur bawaan dan menggunakan PHP. Laravel memiliki fitur seperti Eloquent ORM, Blade templating, dan Laravel Mix untuk pengelolaan aset front-end.

Kelebihan Django:

- ***Kualitas Kode:*** Django mendorong kualitas kode dengan mengikuti prinsip-prinsip desain yang kuat dan menyediakan fitur untuk menjaga kode tetap bersih dan terstruktur.
- ***Python Ecosystem:*** Django memanfaatkan ekosistem Python yang luas, memungkinkan integrasi dengan banyak alat dan pustaka Python lainnya.

Kelebihan Laravel:

- ***Fitur Modern:*** Laravel menawarkan fitur modern seperti routing canggih, middleware, dan sistem migrasi database yang kuat.
- ***Blade Templating:*** Blade, sistem templating Laravel, menawarkan sintaksis yang sederhana dan mudah digunakan untuk membuat tampilan yang dinamis.

Memilih antara Django, Flask, Ruby on Rails, dan Laravel tergantung pada kebutuhan spesifik proyek dan preferensi pengembang. Django menawarkan banyak fitur bawaan dan struktur proyek yang terorganisir dengan baik, ideal untuk aplikasi web besar

Pengenalan Django

dengan kebutuhan kompleks. Flask memberikan fleksibilitas dan kesederhanaan, cocok untuk aplikasi kecil atau layanan mikro.

Ruby on Rails dikenal dengan kemudahan penggunaan dan konvensi yang mempermudah pengembangan aplikasi, sementara Laravel menawarkan fitur modern dan ekosistem PHP yang kuat. Setiap framework memiliki kelebihan dan kekurangan, dan keputusan akhir harus didasarkan pada kebutuhan proyek dan keahlian tim pengembang.

Django, dengan keunggulan-keunggulannya, menjadi pilihan yang sangat kuat bagi pengembang yang ingin membangun aplikasi web yang canggih, aman, dan scalable. Meskipun framework lain mungkin menawarkan keunggulan dalam situasi tertentu, Django memberikan keseimbangan yang ideal antara fitur, kemudahan penggunaan, dan dukungan komunitas yang luas.

1.7.3 Django untuk Proyek Kecil hingga Skala Besar

Django adalah framework yang sangat fleksibel dan dapat digunakan untuk berbagai jenis proyek, mulai dari aplikasi web kecil hingga sistem besar dengan kebutuhan kompleks. Kemampuan ini menjadikannya pilihan yang populer di kalangan pengembang, baik untuk proyek dengan skala kecil maupun besar.

Penggunaan Django dalam Proyek Kecil

Untuk proyek kecil, Django menawarkan banyak kemudahan. Dengan struktur proyek yang telah disiapkan secara otomatis dan fitur-fitur built-in seperti admin interface dan ORM, pengembang

Pengenalan Django

dapat dengan cepat membangun aplikasi web sederhana. Beberapa keunggulan Django untuk proyek kecil meliputi:

- ***Simplicity***: Django mempermudah pembuatan aplikasi web dengan menyediakan banyak fitur bawaan yang mempercepat pengembangan.
- ***Built-in Admin Interface***: Fitur admin Django memungkinkan pengelolaan data aplikasi secara mudah tanpa perlu menulis banyak kode.
- ***ORM (Object-Relational Mapping)***: Django ORM memudahkan interaksi dengan database, membuat query database lebih mudah tanpa memerlukan SQL langsung.

Sebagai contoh, jika kita ingin membangun blog atau aplikasi informasi sederhana, Django memungkinkan kita untuk dengan cepat mengatur model, views, dan templates dengan minimal konfigurasi. Misalnya, untuk membuat aplikasi blog sederhana, kita hanya perlu beberapa perintah untuk mengatur aplikasi dan model dasar.

```
$ django-admin startproject myblog  
$ cd myblog  
$ python manage.py startapp blog
```

Kemudian, kita dapat membuat model dan views dengan cepat di dalam file `models.py` dan `views.py` di aplikasi blog, serta mengatur URL dan templates untuk menampilkan konten blog.

Django untuk Proyek Skala Besar

Django juga sangat cocok untuk proyek dengan skala besar dan kebutuhan kompleks. Fitur dan arsitektur Django dirancang un-

Pengenalan Django

tuk mendukung pengembangan sistem besar dengan berbagai fitur, termasuk:

- **Modularitas:** Django mendukung arsitektur berbasis aplikasi, yang memungkinkan pengembang untuk memecah proyek besar menjadi beberapa aplikasi terpisah. Ini mempermudah pengelolaan dan pengembangan aplikasi secara bersamaan.
- **Scalability:** Django dirancang dengan perhatian terhadap performa dan skalabilitas. Fitur seperti caching, query optimization, dan kemampuan untuk menangani sejumlah besar permintaan secara bersamaan memastikan aplikasi dapat tumbuh seiring dengan meningkatnya pengguna.
- **Keamanan:** Django memiliki banyak fitur keamanan bawaan yang melindungi aplikasi dari berbagai serangan, seperti CSRF, XSS, dan SQL Injection, yang sangat penting untuk aplikasi besar dengan data sensitif.

Untuk proyek besar seperti platform e-commerce atau sistem manajemen konten, Django menyediakan berbagai alat dan fitur untuk mengelola kompleksitas:

1. **Custom Middleware dan Signals:** Untuk mengelola proses yang kompleks dan event-driven.
2. **Advanced ORM:** Untuk melakukan query database yang kompleks dan mengelola relasi antar model.
3. **Caching dan Load Balancing:** Untuk meningkatkan performa dan menangani lalu lintas yang tinggi.

Pengenalan Django

Sebagai contoh, ketika membangun platform e-commerce besar, Django memungkinkan kita untuk menggunakan berbagai aplikasi untuk menangani fungsi-fungsi seperti manajemen produk, pemrosesan pembayaran, dan pengelolaan pengguna, sambil menjaga setiap aplikasi terpisah namun saling terintegrasi dengan baik.

Django adalah framework yang sangat fleksibel dan dapat diadaptasi untuk berbagai skala proyek. Dengan menyediakan alat dan fitur yang kuat, Django memungkinkan pengembang untuk mulai dari proyek kecil dengan kebutuhan sederhana hingga sistem besar dengan kompleksitas tinggi. Fleksibilitas dan kekuatan Django memastikan bahwa aplikasi dapat berkembang seiring dengan pertumbuhan kebutuhan pengguna dan kompleksitas aplikasi.

Dengan langkah-langkah yang telah dijelaskan, kita dapat memanfaatkan Django secara efektif untuk berbagai ukuran proyek, memulai dengan cepat pada proyek kecil dan mengelola kompleksitas besar pada aplikasi yang lebih besar.

1.7.4 Kasus Penggunaan Django dalam Dunia Nyata

Django, sebagai framework web yang kuat dan fleksibel, telah digunakan oleh banyak aplikasi dan perusahaan di dunia nyata. Berikut adalah beberapa studi kasus yang menggambarkan bagaimana berbagai organisasi memanfaatkan Django untuk memenuhi kebutuhan mereka.

Instagram

Instagram adalah salah satu contoh terkenal dari aplikasi besar yang menggunakan Django. Awalnya, Instagram dibangun sebagai aplikasi berbasis Django, memanfaatkan kekuatan framework ini untuk menangani skala besar dan volume data yang tinggi. Django memberikan Instagram kemampuan untuk:

- **Menangani Lalu Lintas Tinggi:** Dengan fitur caching dan performa yang optimal, Django membantu Instagram menangani jutaan pengguna aktif dan foto yang diunggah setiap hari.
- **Skalabilitas:** Django memungkinkan Instagram untuk berkembang dengan mudah, menambah fitur baru seperti Stories dan IGTV tanpa mengorbankan performa.
- **Keamanan:** Django menyediakan berbagai fitur keamanan yang melindungi data pengguna dan mencegah serangan yang umum, yang sangat penting untuk platform media sosial.

Pinterest

Pinterest, platform berbagi gambar dan ide, juga menggunakan Django dalam arsitekturnya. Pinterest memanfaatkan Django untuk:

- **Manajemen Konten:** Django membantu Pinterest mengelola dan menyajikan konten visual dalam skala besar dengan efisiensi tinggi.
- **Pengelolaan Pengguna:** Dengan fitur authentication dan authorization Django, Pinterest dapat mengelola data pengguna dan preferensi mereka dengan aman.

Pengenalan Django

- **Peningkatan Performa:** Pinterest menggunakan Django untuk menyajikan konten dengan cepat dan efisien kepada penggunaannya, bahkan saat traffic sangat tinggi.

Disqus

Disqus, platform komentar yang digunakan oleh berbagai situs web, dibangun dengan Django. Beberapa manfaat yang diperoleh Disqus dari Django termasuk:

- **Pengelolaan Komentar:** Django mempermudah Disqus dalam mengelola komentar yang ditinggalkan oleh pengguna di berbagai situs web.
- **Integrasi Mudah:** Django memungkinkan Disqus untuk mengintegrasikan berbagai fitur tambahan dan beradaptasi dengan kebutuhan situs web yang berbeda.
- **Keamanan dan Skalabilitas:** Django memberikan perlindungan terhadap serangan dan memastikan bahwa sistem dapat menangani volume komentar yang besar.

The Washington Post

The Washington Post, salah satu surat kabar terbesar di Amerika Serikat, menggunakan Django untuk beberapa bagian dari situs web mereka. Django memberikan:

- **Kemampuan untuk Menangani Konten Berita:** Django mempermudah pengelolaan dan penyajian artikel berita secara efisien.
- **Fleksibilitas Desain:** Dengan Django, The Washington Post dapat menyesuaikan desain dan layout sesuai dengan kebutuhan editorial dan teknis.

Pengenalan Django

- ***Pengelolaan Data:*** Django membantu dalam pengelolaan data pengguna dan interaksi mereka dengan konten berita.

NASA

NASA menggunakan Django untuk berbagai aplikasi internal dan publik mereka. Dengan Django, NASA dapat:

- ***Mengelola Data dan Aplikasi:*** Django memungkinkan NASA untuk mengelola data penelitian dan aplikasi dengan tingkat akurasi dan efisiensi yang tinggi.
- ***Memfasilitasi Kolaborasi:*** Django mendukung alat kolaborasi dan pengelolaan proyek yang membantu tim NASA dalam bekerja bersama secara efektif.
- ***Menangani Volume Data Besar:*** Dengan kemampuan Django untuk menangani data besar, NASA dapat memproses dan menyajikan informasi yang kompleks.

Django telah membuktikan kemampuannya untuk digunakan dalam berbagai aplikasi dan organisasi, dari startup kecil hingga perusahaan besar dan lembaga pemerintah. Kemampuannya untuk menangani volume data yang besar, skalabilitas, keamanan, dan kemudahan integrasi membuatnya menjadi pilihan yang sangat baik untuk banyak jenis aplikasi. Studi kasus di atas menunjukkan bagaimana Django dapat diterapkan untuk memenuhi berbagai kebutuhan dan tantangan di dunia nyata.

Dengan memanfaatkan kekuatan Django, organisasi dapat membangun aplikasi yang kuat dan fleksibel, sambil mengelola kompleksitas dan pertumbuhan dengan efisiensi tinggi.

Kesimpulan Bab

Dalam Bab ini, kita telah mempelajari dasar-dasar Django, kerangka kerja web yang sangat powerful dan efisien. Kita mulai dengan memahami apa itu Django dan mengapa ia sangat populer di kalangan pengembang web. Django menawarkan berbagai keunggulan, seperti kemudahan dalam pengembangan aplikasi web berfitur lengkap dan fungsionalitas yang kuat dari out-of-the-box.

Kita juga membahas arsitektur Model-View-Controller (MVC) dalam konteks Django, yang dikenal dengan istilah Model-View-Template (MVT) di Django. Pemahaman tentang komponen utama Django, seperti aplikasi (app), template, URL, views, models, forms, serta static dan media files, adalah kunci untuk mengembangkan aplikasi web yang terstruktur dan mudah dikelola.

Komponen-Komponen Utama:

- ***App***: Modul modular yang memudahkan pengembangan berbagai bagian dari aplikasi web.
- ***Template***: Sistem templating yang memungkinkan pembuatan antarmuka pengguna yang dinamis.
- ***URLs***: Pengaturan URL yang menghubungkan request pengguna dengan views yang sesuai.
- ***Views***: Fungsi atau class yang memproses request dan menghasilkan response.
- ***Models***: Definisi struktur data yang dihubungkan dengan basis data.
- ***Forms***: Alat untuk menangani input dari pengguna dan validasi data.

Pengenalan Django

- ***Static Files dan Media Files***: Pengaturan file statis dan media yang diperlukan untuk antarmuka pengguna dan konten yang diunggah.

Memahami komponen-komponen ini akan mempermudah proses pengembangan aplikasi web Anda. Dengan fondasi yang kuat ini, Anda siap untuk memasuki tahap praktik di bab-bab berikutnya, di mana kita akan mulai membangun aplikasi Django secara langsung, dan menerapkan pengetahuan yang telah dipelajari untuk menciptakan solusi web yang nyata dan fungsional.

Dalam bab selanjutnya, kita akan mulai dengan persiapan proyek Django, langkah demi langkah, dan fokus pada penerapan teori ke dalam praktik. Jangan khawatir jika beberapa konsep terasa kompleks; langkah-langkah praktis akan membantu menjelaskan cara kerja Django secara mendalam.

Dengan semangat belajar dan eksplorasi, mari kita lanjutkan ke bab berikutnya dan mulai membangun aplikasi web Django kita!

BAB 2 - Persiapan Dan Instalasi Lingkungan

Pada bab ini, kita akan memulai perjalanan kita dengan proyek *platform_bot*, sebuah aplikasi bot yang akan berjalan menggunakan Django dan diintegrasikan dengan berbagai platform seperti Telegram dan WhatsApp. Tujuan utama dari bab ini adalah untuk mempersiapkan lingkungan pengembangan yang diperlukan untuk mengembangkan dan menjalankan bot ini secara efisien.

2.1 Tujuan Bab

Kita akan memulai dengan menyiapkan sistem operasi dan memastikan bahwa perangkat lunak yang diperlukan tersedia dan siap digunakan. Ini mencakup instalasi Python, yang merupakan bahasa pemrograman utama yang akan kita gunakan, serta pengaturan lingkungan virtual yang akan membantu kita menjaga proyek tetap terpisah dari proyek lain dan memastikan bahwa semua dependensi yang diperlukan tersedia.

Selanjutnya, kita akan menginstal Django, framework web yang akan menjadi fondasi aplikasi bot kita. Django akan memudahkan kita dalam membangun aplikasi web yang robust dan scalable. Setelah itu, kita akan memasang dan mengonfigurasi Ngrok, alat yang sangat berguna untuk menghubungkan aplikasi lokal kita dengan dunia luar melalui webhook. Ngrok akan memungkinkan bot kita untuk menerima dan merespons pesan dari plat-

Persiapan Dan Instalasi Lingkungan

form seperti Telegram dan WhatsApp secara real-time, bahkan ketika aplikasi berjalan di server lokal.

Selain itu, kita akan memilih dan menginstal framework bot yang sesuai untuk integrasi dengan platform yang kita targetkan. Ini akan mencakup pemasangan library tambahan dan konfigurasi dasar untuk memastikan bahwa bot dapat berkomunikasi dengan benar melalui webhook.

Terakhir, setelah semua instalasi selesai, kita akan melakukan verifikasi untuk memastikan bahwa setiap komponen dari lingkungan pengembangan kita berfungsi sebagaimana mestinya. Ini termasuk menjalankan server pengembangan Django, memeriksa koneksi Ngrok, dan memastikan bahwa bot dapat berinteraksi dengan platform yang telah dipilih.

Dengan menyelesaikan bab ini, Anda akan memiliki lingkungan pengembangan yang siap pakai untuk melanjutkan pengembangan bot pada bab-bab berikutnya. Langkah-langkah yang kita ambil di sini akan memberikan dasar yang kuat untuk membangun fitur-fitur lanjutan dan fungsionalitas bot di masa depan.

Bagian ini memberikan gambaran yang jelas dan terperinci mengenai apa yang akan dicapai dalam bab ini serta langkah-langkah yang akan diambil untuk menyiapkan lingkungan pengembangan untuk proyek *platform_bot*.

2.2 Persiapan Lingkungan

Sebelum kita melanjutkan dengan instalasi perangkat lunak, penting untuk memilih sistem operasi yang akan digunakan untuk pengembangan proyek *platform_bot*. Sistem operasi yang Anda pilih dapat mempengaruhi bagaimana Anda mengatur lingkungan pengembangan Anda, serta bagaimana Anda mengelola dan menjalankan aplikasi.

2.2.1 Memilih Sistem Operasi

Windows adalah sistem operasi yang umum digunakan dan seringkali menjadi pilihan pertama bagi banyak pengembang. Jika Anda menggunakan Windows, Anda perlu memastikan bahwa Anda memiliki alat seperti PowerShell atau Command Prompt untuk menjalankan perintah-perintah terminal. Anda juga harus mempertimbangkan menggunakan Windows Subsystem for Linux (WSL) jika Anda ingin mendapatkan pengalaman yang lebih dekat dengan lingkungan Linux.

macOS adalah pilihan yang sangat baik jika Anda lebih suka bekerja dalam lingkungan Unix-like. macOS sudah dilengkapi dengan banyak alat yang diperlukan untuk pengembangan, dan Anda dapat menggunakan Terminal untuk menjalankan perintah. Pengguna macOS biasanya menemukan bahwa alat-alat pengembangan tersedia secara langsung dan mudah diakses.

Linux adalah sistem operasi yang sangat populer di kalangan pengembang karena fleksibilitas dan kekuatannya. Jika Anda menggunakan Linux, Anda dapat menggunakan terminal untuk menginstal dan mengkonfigurasi perangkat lunak yang diperlukan dengan perintah yang efisien. Linux seringkali menjadi pilihan uta-

Persiapan Dan Instalasi Lingkungan

ma untuk server dan lingkungan produksi karena kemampuannya untuk di-customize dan kinerjanya yang stabil.

Terlepas dari sistem operasi yang Anda pilih, langkah-langkah dasar untuk mempersiapkan lingkungan pengembangan akan serupa. Anda perlu memastikan bahwa sistem Anda memiliki akses ke internet untuk mengunduh perangkat lunak yang diperlukan dan bahwa Anda memiliki hak akses yang cukup untuk menginstal dan mengkonfigurasi alat yang diperlukan.

Jika Anda menggunakan **Windows**, pastikan untuk menginstal Python dari situs resminya dan mempertimbangkan penggunaan alat manajer paket seperti Chocolatey untuk mengelola instalasi perangkat lunak. Anda dapat memulai instalasi Python dengan perintah berikut di Command Prompt:

```
$ python --version # Memeriksa versi Python yang terpasang
```

Jika Anda menggunakan **macOS**, Anda dapat menggunakan Homebrew untuk menginstal Python dan alat lainnya. Cek versi Python yang terpasang dengan perintah:

```
$ python3 --version # Memeriksa versi Python 3 yang terpasang
```

Untuk **Linux**, Anda dapat menggunakan manajer paket seperti apt atau yum tergantung pada distribusi Linux yang Anda gunakan. Periksa versi Python dengan perintah:

Persiapan Dan Instalasi Lingkungan

```
$ python3 --version # Periksa versi Python 3 yang terpasang
```

Setiap sistem operasi memiliki cara dan alatnya masing-masing untuk instalasi dan konfigurasi, tetapi pada akhirnya, tujuan kita adalah untuk memastikan bahwa kita memiliki lingkungan yang siap untuk pengembangan bot *platform_bot*. Pilihlah sistem operasi yang paling sesuai dengan kebutuhan Anda dan kenyamanan Anda dalam bekerja dengan alat dan perangkat lunak.

2.2.2 Memastikan Kebutuhan Sistem

Sebelum kita memulai instalasi perangkat lunak untuk proyek *platform_bot*, penting untuk memastikan bahwa sistem Anda memenuhi kebutuhan minimum untuk menjalankan lingkungan pengembangan dan aplikasi bot dengan lancar. Kebutuhan sistem ini terbagi menjadi dua kategori utama: perangkat keras dan perangkat lunak.

Persyaratan perangkat keras akan bervariasi tergantung pada kompleksitas proyek dan jumlah aplikasi yang dijalankan secara bersamaan. Untuk pengembangan bot menggunakan Django dan Ngrok, perangkat keras yang Anda butuhkan umumnya tidak terlalu tinggi. Namun, ada beberapa rekomendasi umum untuk memastikan kinerja yang optimal:

- **Prosesor:** Sebagian besar tugas pengembangan dan pengujian tidak memerlukan prosesor yang sangat kuat, tetapi prosesor multi-core modern akan memberikan performa yang lebih baik, terutama saat menjalankan beberapa aplikasi atau proses secara bersamaan.

Persiapan Dan Instalasi Lingkungan

- **RAM:** Disarankan memiliki minimal 4 GB RAM, namun 8 GB atau lebih akan memberikan pengalaman yang lebih baik dan memungkinkan Anda menjalankan beberapa aplikasi sekaligus tanpa masalah.
- **Ruang Penyimpanan:** Proyek pengembangan seperti ini tidak memerlukan banyak ruang penyimpanan, tetapi pastikan Anda memiliki setidaknya 10 GB ruang kosong untuk sistem operasi, perangkat lunak, dan proyek Anda. SSD (Solid State Drive) akan mempercepat akses data dan waktu loading aplikasi.
- **Koneksi Internet:** Koneksi internet yang stabil diperlukan untuk mengunduh perangkat lunak, melakukan pembaruan, dan menguji webhook dengan Ngrok.

Persyaratan perangkat lunak mencakup berbagai alat dan program yang harus diinstal untuk memulai pengembangan. Berikut adalah perangkat lunak utama yang Anda butuhkan:

- **Python:** Versi terbaru Python 3.x harus diinstal di sistem Anda. Python adalah bahasa pemrograman yang digunakan untuk mengembangkan aplikasi Django. Untuk memeriksa versi Python yang terpasang, Anda dapat menggunakan perintah berikut di terminal:

```
$ python3 --version # Memeriksa versi Python yang terpasang
```

- **Paket Pengelola:** pip adalah paket pengelola Python yang memungkinkan Anda untuk menginstal pustaka dan dependensi yang diperlukan. Biasanya, pip sudah ter-

Persiapan Dan Instalasi Lingkungan

masuk dalam instalasi Python. Anda dapat memeriksa versi pip dengan perintah:

```
$ pip --version # Memeriksa versi pip yang terpasang
```

- **Django:** Framework yang akan digunakan untuk membangun aplikasi web Anda. Instal Django menggunakan pip dengan perintah:

```
$ pip install django # Menginstal Django
```

- **Ngrok:** Alat untuk membuat tunnel HTTP ke server lokal Anda, memungkinkan webhook berfungsi dengan baik. Anda dapat mengunduh Ngrok dari situs resminya dan mengikuti instruksi instalasi untuk sistem operasi Anda. Setelah instalasi, Anda bisa memeriksa versi Ngrok dengan perintah:

```
$ ngrok version # Memeriksa versi Ngrok yang terpasang
```

- **Framework Bot:** Tergantung pada platform bot yang Anda pilih (seperti python-telegram-bot untuk Telegram), Anda akan memerlukan pustaka tambahan yang dapat diinstal menggunakan pip. Misalnya, untuk python-telegram-bot, perintahnya adalah:

```
$ pip install python-telegram-bot # Menginstal python-telegram-bot
```

Persiapan Dan Instalasi Lingkungan

Dengan memastikan bahwa perangkat keras dan perangkat lunak Anda memenuhi persyaratan ini, Anda akan dapat menjalankan *platform_bot* dengan lancar dan tanpa masalah teknis yang signifikan. Langkah ini adalah fondasi yang penting sebelum melanjutkan dengan instalasi lebih lanjut dan pengembangan proyek.

2.3 Instalasi Python Dan Virtual Environment

2.3.1 Instalasi Python

Python adalah bahasa pemrograman yang akan menjadi fondasi bagi pengembangan proyek *platform_bot*. Memastikan bahwa Python terinstal dengan benar di sistem Anda adalah langkah awal yang krusial. Berikut adalah panduan untuk mengunduh dan menginstal Python pada sistem operasi yang umum digunakan.

Cara Mengunduh dan Menginstal Python

Untuk memulai, Anda perlu mengunduh versi terbaru dari Python dari situs resmi Python. Buka situs web python.org dan pilih versi Python 3 yang sesuai dengan sistem operasi Anda. Pada halaman unduhan, Anda akan menemukan versi Python yang stabil dan direkomendasikan.

- **Windows:** Unduh installer Windows dan jalankan file yang diunduh. Pada jendela instalasi, pastikan untuk mencentang opsi “Add Python to PATH” sebelum melanjutkan. Ini akan mempermudah Anda dalam menjalankan Python dari Command Prompt. Klik tombol “Install Now” dan ikuti petunjuk untuk menyelesaikan instalasi.

Persiapan Dan Instalasi Lingkungan

- **macOS:** Unduh installer macOS dari situs Python dan buka file .pkg yang diunduh. Ikuti panduan instalasi yang muncul di layar. Python akan diinstal ke dalam direktori default dan dapat diakses melalui Terminal.
- **Linux:** Sebagian besar distribusi Linux sudah menyertakan Python secara default. Namun, jika perlu menginstal atau memperbarui Python, Anda dapat menggunakan manajer paket yang sesuai dengan distribusi Anda. Misalnya, pada Ubuntu, Anda dapat menjalankan:

```
$ sudo apt update # Memperbarui daftar paket
$ sudo apt install python3 # Menginstal Python
3
```

Memeriksa Versi Python yang Terpasang

Setelah instalasi Python selesai, penting untuk memverifikasi bahwa Python telah terinstal dengan benar dan memeriksa versinya. Ini memastikan bahwa versi yang terinstal sesuai dengan yang dibutuhkan proyek Anda.

- **Windows:** Buka Command Prompt dan jalankan perintah berikut:

```
$ python --version # Memeriksa versi Python
yang terpasang
```

- **macOS dan Linux:** Buka Terminal dan jalankan perintah berikut:

```
$ python3 --version # Memeriksa versi Python 3
yang terpasang
```

Persiapan Dan Instalasi Lingkungan

Perintah ini akan menampilkan versi Python yang terpasang di sistem Anda. Pastikan versi yang terpasang adalah Python 3.6 atau yang lebih baru, sesuai dengan kebutuhan proyek *platform_bot*.

Dengan Python terinstal dengan benar dan versi yang sesuai, Anda siap untuk melanjutkan ke langkah berikutnya yaitu mengatur lingkungan virtual, yang akan membantu Anda dalam mengelola dependensi proyek dan menjaga lingkungan pengembangan tetap bersih.

2.3.2 Mengatur Virtual Environment (Opsional tetapi Disarankan):

Untuk memastikan pengelolaan proyek *platform_bot* yang bersih dan terpisah dari proyek lainnya, kita akan menggunakan **virtual environment**. Virtual environment adalah alat yang memungkinkan kita untuk membuat lingkungan terisolasi untuk proyek Python. Ini memudahkan pengelolaan dependensi spesifik proyek tanpa mengganggu pengaturan global Python di sistem Anda.

Penjelasan tentang Virtual Environment

Virtual environment adalah direktori yang berisi salinan mandiri dari interpreter Python, serta folder untuk menginstal paket yang diperlukan untuk proyek. Dengan menggunakan virtual environment, Anda dapat:

- Menghindari konflik antara paket-paket yang dibutuhkan oleh berbagai proyek.

Persiapan Dan Instalasi Lingkungan

- Menjaga sistem Python global tetap bersih dari paket-paket yang mungkin hanya diperlukan untuk proyek tertentu.
- Mempermudah pengelolaan dependensi dengan mengisolasi lingkungan proyek.

Cara Membuat dan Mengaktifkan Virtual Environment

Untuk membuat dan mengaktifkan virtual environment, ikuti langkah-langkah berikut:

1. *Membuat Virtual Environment*

Pastikan Anda berada di direktori proyek Anda sebelum membuat virtual environment. Gunakan perintah **venv** yang disediakan oleh Python untuk membuat lingkungan virtual. Berikut adalah perintah untuk membuat virtual environment dengan nama **venv**:

```
$ python3 -m venv venv # Membuat virtual environment dengan nama 'venv'
```

Perintah ini akan membuat direktori baru bernama **venv** di dalam direktori proyek Anda. Direktori ini akan berisi salinan interpreter Python dan subdirektori untuk paket-paket yang diinstal.

2. *Mengaktifkan Virtual Environment*

Setelah virtual environment dibuat, Anda perlu mengaktifkannya untuk mulai bekerja di lingkungan tersebut. Metode aktivasi berbeda tergantung pada sistem operasi yang Anda gunakan.

- **Windows:** Jalankan perintah berikut di Command Prompt untuk mengaktifkan virtual environment:

Persiapan Dan Instalasi Lingkungan

```
$ venv\Scripts\activate # Mengaktifkan virtual environment di Windows
```

- **macOS** dan **Linux**: Jalankan perintah berikut di Terminal untuk mengaktifkan virtual environment:

```
$ source venv/bin/activate # Mengaktifkan virtual environment di macOS dan Linux
```

Setelah virtual environment diaktifkan, prompt terminal Anda akan berubah untuk menunjukkan bahwa Anda berada dalam lingkungan virtual. Misalnya, Anda akan melihat nama lingkungan virtual (misalnya (venv)) di awal baris perintah.

Menonaktifkan Virtual Environment

Untuk keluar dari virtual environment, cukup jalankan perintah berikut:

```
$ deactivate # Menonaktifkan virtual environment
```

Menonaktifkan virtual environment akan mengembalikan prompt terminal Anda ke pengaturan global Python Anda.

Dengan virtual environment yang telah diatur dan diaktifkan, Anda siap untuk menginstal dependensi proyek seperti Django dan pustaka lainnya tanpa mempengaruhi pengaturan global Python di sistem Anda. Langkah ini adalah fondasi untuk mengembangkan dan mengelola proyek **platform_bot** dengan cara yang terstruktur dan terpisah.

2.4 Instalasi Django

Setelah Anda menyiapkan Python dan virtual environment, langkah berikutnya adalah menginstal Django dan memulai proyek Django pertama Anda. Django adalah framework web yang kuat dan fleksibel, yang akan menjadi inti dari pengembangan aplikasi **platform_bot** Anda. Di bawah ini, kami akan menjelaskan proses instalasi Django dan cara membuat proyek Django pertama Anda.

2.4.1 Menggunakan pip untuk Instalasi Django

Pip adalah manajer paket untuk Python yang digunakan untuk menginstal dan mengelola paket pihak ketiga, termasuk Django.

Untuk menginstal Django, pastikan bahwa virtual environment Anda sudah aktif. Jika virtual environment belum diaktifkan, aktifkan terlebih dahulu dengan perintah yang sesuai untuk sistem operasi Anda.

Berikut adalah langkah-langkah untuk menginstal Django menggunakan pip:

Aktifkan Virtual Environment: Pastikan virtual environment Anda sudah aktif. Jika belum, aktifkan virtual environment sesuai dengan instruksi sebelumnya.

Di Windows:

Persiapan Dan Instalasi Lingkungan

```
$ myenv\Scripts\activate
```

Di macOS dan Linux:

```
$ source myenv/bin/activate
```

Instal Django: Setelah virtual environment aktif, jalankan perintah berikut untuk menginstal Django:

```
$ pip install django # Menginstal Django  
menggunakan pip
```

Perintah ini akan mengunduh dan menginstal Django beserta dependensinya ke dalam virtual environment Anda. Proses ini memerlukan koneksi internet untuk mengunduh paket-paket yang diperlukan dari PyPI (Python Package Index).

Memverifikasi Instalasi Django

Setelah instalasi selesai, penting untuk memverifikasi bahwa Django telah terinstal dengan benar dan siap digunakan. Untuk memverifikasi instalasi, Anda dapat memeriksa versi Django yang terpasang dengan perintah berikut:

```
$ django-admin --version # Memeriksa versi  
Django yang terpasang
```

Perintah ini akan menampilkan versi Django yang saat ini terpasang di virtual environment Anda. Pastikan versi yang terpasang adalah versi terbaru atau sesuai dengan yang dibutuhkan untuk proyek *platform_bot*.

Persiapan Dan Instalasi Lingkungan

Dengan Django terinstal, Anda sekarang siap untuk memulai proyek Django pertama Anda. Selanjutnya, Anda akan belajar bagaimana membuat dan mengonfigurasi proyek Django untuk *platform_bot* Anda, dan memulai pengembangan fitur yang diperlukan untuk aplikasi Anda. Pastikan untuk mengikuti langkah-langkah dengan cermat agar lingkungan pengembangan Anda siap digunakan dengan baik.

2.4.2 Membuat Proyek Django Pertama

Setelah berhasil menginstal Django, langkah berikutnya adalah membuat proyek Django pertama Anda. Proyek Django akan menjadi kerangka kerja utama untuk aplikasi *platform_bot*. Proses ini mencakup pembuatan proyek dan pemahaman tentang struktur dasar proyek yang dihasilkan.

Untuk membuat proyek Django baru, pastikan virtual environment Anda aktif. Kemudian, gunakan perintah `django-admin` untuk memulai pembuatan proyek. Misalnya, untuk membuat proyek dengan nama `platform_bot`, jalankan perintah berikut:

```
$ django-admin startproject platform_bot #  
Membuat proyek Django baru dengan nama  
'platform_bot'
```

Perintah ini akan membuat direktori baru bernama `platform_bot` yang berisi struktur dasar proyek Django.

Struktur Dasar Proyek Django

Persiapan Dan Instalasi Lingkungan

Setelah menjalankan perintah di atas, Django akan menghasilkan struktur direktori dasar untuk proyek Anda. Struktur ini menyediakan kerangka kerja yang diperlukan untuk mengembangkan aplikasi web Anda. Berikut adalah gambaran umum dari struktur direktori dasar proyek Django:

```
platform_bot/          # Direktori utama proyek
├── manage.py           # Skrip untuk mengelola
proyek Django (misalnya, menjalankan server,
membuat migrasi)
├── platform_bot/      # Direktori proyek
Django
│   ├── __init__.py     # Menandai direktori ini
sebagai paket Python
│   ├── asgi.py         # Konfigurasi untuk ASGI
(Asynchronous Server Gateway Interface)
│   ├── settings.py     # Berisi pengaturan dan
konfigurasi proyek
│   ├── urls.py         # Menangani pemetaan URL
untuk proyek
│   └── wsgi.py         # Konfigurasi untuk WSGI
(Web Server Gateway Interface)
```

- ***manage.py***: Skrip ini digunakan untuk berbagai tugas administratif seperti menjalankan server pengembangan, membuat migrasi, dan memulai aplikasi Django. Anda akan sering menggunakan skrip ini selama pengembangan proyek.
- ***platform_bot/***: Ini adalah direktori proyek yang sama dengan nama proyek yang Anda buat. Di dalam direktori ini terdapat file-file konfigurasi penting untuk proyek Django.

Persiapan Dan Instalasi Lingkungan

- **`__init__.py`**: Menandai direktori ini sebagai paket Python, memungkinkan Django untuk mengenalinya sebagai bagian dari proyek.
- **`asgi.py`**: Berisi konfigurasi untuk ASGI, yang digunakan untuk mengelola aplikasi asinkron dan komunikasi web real-time.
- **`settings.py`**: File ini berisi pengaturan dan konfigurasi utama untuk proyek Django Anda, seperti database, file statis, dan pengaturan aplikasi.
- **`urls.py`**: Menangani pemetaan URL untuk proyek Anda. Di sini Anda akan mendefinisikan pola URL yang digunakan untuk mengarahkan permintaan ke tampilan yang sesuai.
- **`wsgi.py`**: Berisi konfigurasi untuk WSGI, yang digunakan untuk komunikasi antara aplikasi Django dan server web.

Dengan struktur dasar proyek yang telah dibuat, Anda sekarang memiliki fondasi untuk memulai pengembangan aplikasi *platform_bot*. Langkah selanjutnya adalah mengonfigurasi dan menyiapkan aplikasi Django Anda untuk kebutuhan spesifik proyek.

2.5 Instalasi Dan Konfigurasi Ngrok

Ngrok adalah alat yang memungkinkan Anda untuk mengekspos server lokal Anda ke internet melalui URL publik yang aman. Ini sangat berguna untuk menguji aplikasi web yang sedang dikembangkan di lingkungan lokal, terutama ketika Anda perlu menun-

Persiapan Dan Instalasi Lingkungan

jukkan proyek kepada orang lain atau menguji integrasi dengan layanan eksternal yang memerlukan akses publik.

Ngrok adalah alat yang sangat berguna untuk mengakses aplikasi web lokal Anda dari luar jaringan lokal. Ini memungkinkan Anda untuk membuat terowongan HTTPS ke server lokal Anda, sehingga Anda dapat menguji dan berinteraksi dengan aplikasi Anda di lingkungan yang lebih luas, seperti webhook untuk bot Anda. Berikut adalah langkah-langkah untuk mengunduh dan menginstal Ngrok.

2.5.1 Mengunduh dan Menginstal Ngrok

Untuk mengunduh Ngrok, kunjungi situs resmi Ngrok di ngrok.com/download. Pada halaman ini, Anda akan menemukan berbagai opsi unduhan untuk sistem operasi yang berbeda, termasuk Windows, macOS, dan Linux.

- **Windows:** Pilih versi Windows dan unduh file ZIP. Ekstrak file ZIP yang diunduh ke direktori pilihan Anda.
- **macOS:** Pilih versi macOS dan unduh file ZIP atau file DMG. Jika Anda mengunduh file ZIP, ekstrak file tersebut. Jika Anda memilih file DMG, buka file DMG dan seret file `ngrok` ke dalam folder `Applications`.
- **Linux:** Pilih versi Linux dan unduh file TAR. Ekstrak file TAR dengan perintah berikut:

```
$ tar -xvf ngrok-stable-linux-amd64.tar.gz #  
Meng-ekstrak file TAR Ngrok
```


Persiapan Dan Instalasi Lingkungan

Dengan Ngrok yang terinstal Anda sekarang dapat mengakses dan menguji aplikasi *platform_bot* Anda secara remote. Langkah ini adalah bagian penting dalam memverifikasi fungsionalitas bot dan memastikan bahwa integrasi dengan layanan eksternal berjalan dengan baik.

2.5.2 Konfigurasi dan Menjalankan Ngrok

Setelah menginstal Ngrok, langkah berikutnya adalah mengkonfigurasi dan menjalankannya untuk mengekspose server lokal Anda ke internet.

Membuat Akun Ngrok (Opsional tetapi Direkomendasikan):

Ngrok menyediakan fitur tambahan seperti dasbor dan URL khusus jika Anda mendaftar untuk akun gratis di <https://ngrok.com/>. Setelah mendaftar, Anda akan mendapatkan token otentikasi yang dapat digunakan untuk mengonfigurasi Ngrok Anda.

Menambahkan Token Otentikasi (Opsional): Jika Anda memiliki akun Ngrok, tambahkan token otentikasi ke Ngrok dengan menjalankan perintah berikut:

```
$ ngrok authtoken YOUR_AUTH_TOKEN
```

Gantilah `YOUR_AUTH_TOKEN` dengan token otentikasi yang diberikan di dasbor akun Ngrok Anda.

Persiapan Dan Instalasi Lingkungan

Menjalankan Ngrok: Untuk mengekspose server lokal Django Anda, yang biasanya berjalan di `http://127.0.0.1:8000`, jalankan perintah berikut:

```
$ ngrok http 8000
```

Perintah ini akan membuka terowongan HTTP pada port 8000 dan memberi Anda URL publik yang dapat diakses oleh siapa saja di internet.

Output Ngrok: Setelah menjalankan perintah di atas, Anda akan melihat output seperti ini di terminal:

```
Session Status      online
Account             Your Name (Plan:
Free)
Version             3.0.0
Region             United States (us)
Web Interface
http://127.0.0.1:4040
Forwarding
http://abcdef1234.ngrok.io ->
http://localhost:8000
Forwarding
https://abcdef1234.ngrok.io ->
http://localhost:8000

Connections          ttl      opn
rt1      rt5      p50      p90
0         0.00    0.00    0.00    0.00
```

Pada output tersebut, Anda akan melihat dua URL, satu menggunakan HTTP dan satu lagi menggunakan HTTPS, misalnya:

Persiapan Dan Instalasi Lingkungan

```
Forwarding  
http://abcdef1234.ngrok.io ->  
http://localhost:8000  
Forwarding  
https://abcdef1234.ngrok.io ->  
http://localhost:8000
```

Anda dapat menggunakan salah satu dari URL ini untuk mengakses server lokal Anda dari internet.

Menghentikan Ngrok: Untuk menghentikan Ngrok, cukup tekan **Ctrl + C** di terminal.

Dengan mengikuti langkah-langkah di atas, Anda telah berhasil menginstal, mengonfigurasi, dan menjalankan Ngrok untuk mengekspose server lokal Anda ke internet. Pada bab-bab berikutnya, kita akan memulai pembuatan proyek Django Anda dan menggunakan Ngrok untuk menguji aplikasi tersebut secara online.

2.5.3 Mengatur Ngrok untuk Webhook

Setelah berhasil menginstal Ngrok, langkah berikutnya adalah mengatur Ngrok agar dapat digunakan untuk webhook. Webhook memungkinkan aplikasi Anda, seperti bot Telegram atau WhatsApp, untuk menerima data dari layanan eksternal secara real-time. Ngrok akan membuat terowongan HTTPS ke server lokal Anda, memungkinkan webhook mengakses aplikasi Django Anda dari luar jaringan lokal.

Persiapan Dan Instalasi Lingkungan

Menyiapkan dan Menjalankan Ngrok

Pertama-tama, pastikan bahwa server pengembangan Django Anda sudah berjalan. Biasanya, Anda akan menjalankan server Django di port 8000. Untuk memulai server Django, gunakan perintah berikut dari direktori proyek Anda:

```
$ python manage.py runserver 8000 # Menjalankan server pengembangan Django di port 8000
```

Setelah server Django aktif, buka terminal baru dan jalankan Ngrok untuk membuat terowongan ke port 8000. Gunakan perintah berikut untuk memulai Ngrok:

```
$ ./ngrok http 8000 # Membuat terowongan HTTPS ke port 8000
```

Perintah ini akan menghasilkan output yang menampilkan URL publik yang dapat digunakan untuk mengakses server lokal Anda. URL ini akan terlihat seperti berikut:

```
Forwarding https://<random-string>.ngrok.io -> http://localhost:8000
```

Menghubungkan Ngrok dengan Aplikasi Django

Sekarang, Anda perlu mengonfigurasi aplikasi Django Anda untuk menerima webhook dari layanan eksternal. Untuk contoh ini, mari kita gunakan bot Telegram. Anda harus mengonfigurasi bot Telegram untuk mengirimkan data ke URL Ngrok yang Anda dapatkan.

1. Mendapatkan Token API Telegram

Persiapan Dan Instalasi Lingkungan

Pertama, pastikan Anda memiliki token API untuk bot Telegram Anda. Token ini diperoleh dari BotFather di Telegram.

2. Mengonfigurasi Webhook di Telegram

Gunakan token API dan URL Ngrok untuk mengonfigurasi webhook. Jalankan perintah berikut untuk mengatur webhook dengan menggunakan `curl` atau alat serupa:

```
$ curl -F "url=https://<random-string>.ngrok.io/webhook/"  
https://api.telegram.org/bot<YOUR_BOT_TOKEN>/setWebhook
```

Gantilah `<random-string>` dengan string acak dari URL Ngrok yang diberikan, dan `<YOUR_BOT_TOKEN>` dengan token API bot Telegram Anda.

3. Menyiapkan URL Webhook di Django

Di aplikasi Django Anda, buat tampilan yang akan menangani data dari webhook. Misalnya, buat file `views.py` di dalam aplikasi bot Anda dengan kode berikut:

```
# bot/views.py  
from django.http import JsonResponse  
from django.views.decorators.csrf import  
csrf_exempt  
import json  
  
@csrf_exempt  
def webhook(request):  
    if request.method == 'POST':  
        data = json.loads(request.body)  
        # Proses data webhook dari Telegram  
        return JsonResponse({'status': 'ok'})
```

Persiapan Dan Instalasi Lingkungan

```
return JsonResponse({'status': 'failed'})
```

Kemudian, tambahkan URL untuk tampilan webhook ini ke dalam `urls.py` aplikasi Anda:

```
# bot/urls.py
from django.urls import path
from .views import webhook

urlpatterns = [
    path('webhook/', webhook, name='webhook'),
]
```

Dengan pengaturan ini, Ngrok akan meneruskan permintaan HTTPS ke server lokal Django Anda, dan Django akan menangani data webhook dari bot Telegram.

Sekarang, Ngrok telah berhasil diatur untuk menerima dan meneruskan webhook ke aplikasi Django Anda. Anda dapat memantau log Ngrok untuk melihat apakah webhook diterima dengan benar dan memeriksa respons yang diberikan oleh aplikasi Django Anda.

2.6 Instalasi Dan Konfigurasi Bot Framework

Dalam pengembangan bot, memilih framework yang tepat adalah langkah awal yang krusial. Framework bot akan menentukan bagaimana bot Anda berinteraksi dengan pengguna dan layanan lain.

Persiapan Dan Instalasi Lingkungan

2.6.1 Memilih Framework Bot

Ada beberapa pilihan populer untuk framework bot yang dapat Anda pilih, masing-masing dengan fitur dan integrasi yang berbeda. Berikut adalah beberapa opsi framework bot yang umum digunakan:

1. Telegram Bot

Telegram merupakan salah satu platform yang sangat mendukung pengembangan bot. Telegram menyediakan API yang kuat dan dokumentasi yang jelas untuk memudahkan pengembangan bot. Bot Telegram memungkinkan interaksi yang kompleks dengan pengguna dan dapat menangani berbagai jenis konten, seperti pesan teks, gambar, dan file.

Kelebihan:

- API yang lengkap dan fleksibel
- Dukungan untuk berbagai jenis konten
- Dokumentasi dan komunitas yang aktif

Cara Memulai: Untuk memulai dengan bot Telegram, Anda perlu mendaftar bot melalui BotFather di Telegram dan mendapatkan token API. Selanjutnya, Anda dapat menggunakan token tersebut untuk mengakses API Telegram dari aplikasi Django Anda.

2. WhatsApp Bot

WhatsApp adalah platform komunikasi yang sangat populer dengan jutaan pengguna aktif di seluruh dunia. Untuk membuat bot di WhatsApp, Anda bisa menggunakan layanan seperti Twilio

Persiapan Dan Instalasi Lingkungan

atau WhatsApp Business API yang memungkinkan integrasi bot dengan WhatsApp.

Kelebihan:

- Akses ke basis pengguna yang luas
- Integrasi mudah dengan Twilio dan layanan serupa
- Fitur pengiriman pesan multimedia

Cara Memulai: Untuk menggunakan WhatsApp API, Anda harus mendaftar melalui penyedia API seperti Twilio dan mengikuti panduan mereka untuk mengonfigurasi bot dan mendapatkan kredensial API.

3. Facebook Messenger Bot

Facebook Messenger adalah platform lain yang menawarkan fitur bot. Bot Messenger dapat diintegrasikan dengan halaman Facebook dan memungkinkan interaksi langsung dengan pengguna melalui pesan.

Kelebihan:

- Integrasi dengan Facebook Pages
- Dukungan untuk pesan teks, gambar, dan video
- Fitur canggih untuk interaksi pengguna

Cara Memulai: Untuk membuat bot Messenger, Anda perlu mendaftar untuk Facebook Developer dan mengonfigurasi bot melalui [Facebook Messenger Platform](#). Anda akan mendapatkan token akses untuk berinteraksi dengan API Messenger.

Persiapan Dan Instalasi Lingkungan

4. Platform Lainnya

Selain framework bot di atas, ada juga berbagai platform lain yang mungkin sesuai dengan kebutuhan Anda, seperti Slack, Microsoft Teams, atau Discord. Setiap platform memiliki fitur unik dan API yang berbeda, sehingga Anda bisa memilih sesuai dengan tujuan dan audiens target Anda.

Kelebihan dan Kekurangan:

- **Slack:** Terintegrasi dengan alat kolaborasi, ideal untuk aplikasi bisnis.
- **Microsoft Teams:** Terintegrasi dengan Office 365, cocok untuk lingkungan perusahaan.
- **Discord:** Populer di kalangan komunitas game, mendukung banyak jenis interaksi.

Cara Memulai: Untuk platform lain, ikuti panduan masing-masing untuk mendaftar dan mengonfigurasi bot sesuai dengan dokumentasi yang disediakan oleh penyedia layanan.

Dengan memilih framework bot yang sesuai, Anda dapat memulai pengembangan bot yang efektif dan sesuai dengan kebutuhan Anda. Setelah memilih framework, langkah berikutnya adalah menginstal dan mengonfigurasi SDK atau API yang diperlukan untuk berinteraksi dengan platform pilihan Anda.

Persiapan Dan Instalasi Lingkungan

2.6.2 Instalasi dan Konfigurasi Bot Framework

Setelah memilih framework bot yang sesuai dengan kebutuhan proyek Anda, langkah selanjutnya adalah menginstal library yang diperlukan dan melakukan konfigurasi dasar agar bot Anda dapat berjalan dengan baik. Pada bagian ini, kita akan fokus pada instalasi library dan penyiapan konfigurasi awal untuk memulai pengembangan bot.

Instalasi Library yang Diperlukan

Untuk memulai, Anda harus memastikan bahwa semua library yang dibutuhkan oleh bot framework telah diinstal pada environment proyek Django Anda. Sebagai contoh, jika Anda memilih untuk menggunakan Telegram sebagai platform bot,

Anda bisa menginstal library yang mendukung interaksi dengan API Telegram, seperti `python-telegram-bot`.

Pertama-tama, pastikan virtual environment Anda aktif. Jika belum, Anda bisa mengaktifkannya dengan perintah berikut:

```
$ source venv/bin/activate
```

Setelah virtual environment aktif, langkah selanjutnya adalah menginstal library yang dibutuhkan. Untuk bot Telegram, instal library `python-telegram-bot` menggunakan `pip`:

```
$ pip install python-telegram-bot
```

Pastikan library tersebut terinstal dengan benar, dan periksa apakah sudah terpasang dengan menjalankan:

Persiapan Dan Instalasi Lingkungan

```
$ pip freeze
```

Jika Anda menggunakan platform lain, seperti WhatsApp melalui Twilio, Anda perlu menginstal library `twilio`:

```
$ pip install twilio
```

Library yang terinstal ini akan memungkinkan Anda untuk berinteraksi dengan API platform bot yang telah dipilih, baik itu Telegram, WhatsApp, atau lainnya.

2.7 Instalasi Dan Penggunaan Code Editor

Code editor adalah alat penting dalam pengembangan perangkat lunak, termasuk untuk proyek Django seperti *platform_bot*. Memilih code editor yang tepat dapat meningkatkan produktivitas dan membuat pengembangan lebih efisien. Pada sub bab ini, kita akan membahas instalasi dan penggunaan code editor, serta fitur-fitur yang mendukung pengembangan aplikasi Django.

Dalam pengembangan perangkat lunak, pemilihan code editor yang tepat sangat penting untuk meningkatkan produktivitas. Salah satu editor teks yang sangat fleksibel dan kuat adalah Vim. Meskipun Vim terkenal dengan kurva belajar yang cukup curam, namun setelah Anda menguasainya, Vim bisa menjadi alat yang sangat efisien untuk pengembangan.

Persiapan Dan Instalasi Lingkungan

Visual Studio Code, juga salah satu code editor yang paling populer dan banyak digunakan dalam pengembangan web.

2.7.1 Memilih Code Editor

Terdapat banyak code editor yang dapat Anda pilih untuk pengembangan Python dan Django. Beberapa yang populer adalah:

- **Visual Studio Code (VS Code):** Code editor ringan namun sangat kuat yang mendukung banyak bahasa pemrograman dan memiliki banyak ekstensi yang berguna.
- **PyCharm:** IDE yang dirancang khusus untuk Python dan Django, menawarkan fitur-fitur canggih untuk pengembangan web.
- **Sublime Text:** Editor teks yang cepat dan ringan dengan dukungan untuk berbagai plugin.
- **Vim:** editor teks berbasis terminal yang sangat populer di kalangan pengembang perangkat lunak

2.7.2 Vim

Vim adalah editor teks berbasis terminal yang sangat populer di kalangan pengembang perangkat lunak, terutama mereka yang sering bekerja di lingkungan Unix/Linux. Vim mendukung banyak fitur canggih seperti syntax highlighting, split windows, macros, dan banyak lagi.

Instalasi Vim

Untuk mulai menggunakan Vim, Anda perlu menginstalnya terlebih dahulu. Berikut adalah langkah-langkah instalasi Vim di berbagai sistem operasi:

Persiapan Dan Instalasi Lingkungan

1. **Instalasi di Ubuntu/Debian:** Jika Anda menggunakan distribusi berbasis Debian atau Ubuntu, Anda dapat menginstal Vim dengan perintah berikut:

```
$ sudo apt update  
$ sudo apt install vim
```

2. **Instalasi di Fedora/CentOS:** Untuk distribusi berbasis Red Hat, seperti Fedora atau CentOS, gunakan perintah berikut:

```
$ sudo dnf install vim
```

3. **Instalasi di macOS:** Di macOS, Anda bisa menginstal Vim menggunakan Homebrew:

```
$ brew install vim
```

4. **Instalasi di Windows:** Untuk pengguna Windows, Anda bisa menginstal Vim dengan mendownload installer dari situs resmi Vim. Pilih installer yang sesuai dengan versi Windows Anda dan ikuti petunjuk instalasi.
5. **Verifikasi Instalasi:** Setelah instalasi selesai, Anda dapat memverifikasi bahwa Vim telah terpasang dengan benar dengan menjalankan perintah berikut di terminal:

```
$ vim --version
```

Outputnya akan menunjukkan versi Vim yang diinstal beserta informasi konfigurasi lainnya.

Persiapan Dan Instalasi Lingkungan

Konfigurasi Dasar Vim

Setelah Vim diinstal, Anda mungkin ingin mengkonfigurasinya agar sesuai dengan preferensi pengembangan Anda. Berikut adalah beberapa konfigurasi dasar yang bisa Anda lakukan di Vim:

1. **Membuat File Konfigurasi:** Vim menyimpan konfigurasi pengguna di file `.vimrc` yang terletak di direktori home Anda. Jika file ini belum ada, Anda bisa membuatnya dengan perintah:

```
$ touch ~/.vimrc
```

2. **Pengaturan Dasar di `.vimrc`:** Berikut adalah beberapa pengaturan dasar yang sering digunakan:

```
set number           " Menampilkan nomor
baris
set tabstop=4         " Mengatur tab
menjadi 4 spasi
set shiftwidth=4      " Mengatur indentasi
menjadi 4 spasi
set expandtab         " Mengubah tab
menjadi spasi
syntax on            " Mengaktifkan
syntax highlighting
set autoindent        " Mengaktifkan auto
indentasi
```

Untuk mengedit file `.vimrc`, buka file tersebut dengan Vim:

```
$ vim ~/.vimrc
```

Kemudian, masukkan konfigurasi di atas dan simpan.

Persiapan Dan Instalasi Lingkungan

3. **Navigasi Dasar di Vim:** Vim memiliki mode yang berbeda untuk navigasi dan pengeditan teks:

- **Normal Mode:** Mode default setelah Vim dibuka. Anda dapat melakukan navigasi dan operasi pengeditan teks.
- **Insert Mode:** Mode untuk memasukkan teks. Masuk ke mode ini dengan menekan **i**, dan keluar kembali ke Normal Mode dengan **ESC**.
- **Visual Mode:** Mode untuk memilih teks. Masuk ke mode ini dengan menekan **v**.

4. **Contoh Penggunaan Vim:**

- **Menulis Teks:** Untuk mulai menulis teks, buka file dengan Vim:

```
$ vim namafile.txt
```

Tekan **i** untuk masuk ke Insert Mode, lalu ketik teks yang Anda inginkan. Setelah selesai, tekan **ESC** untuk keluar dari Insert Mode.

- **Menyimpan dan Keluar:** Untuk menyimpan file dan keluar dari Vim, tekan **ESC** untuk memastikan Anda berada di Normal Mode, kemudian ketik **:wq** dan tekan **Enter**.

```
:wq
```

- **Memilih Teks:** Untuk memilih teks, gunakan Visual Mode. Tekan **v** untuk masuk ke Visual Mode, gunakan tombol panah untuk memilih teks yang diinginkan, lalu lakukan operasi seperti me-

Persiapan Dan Instalasi Lingkungan

nyalin (y), memotong (d), atau menempelkan (p).

- **Membatalkan Perubahan:** Untuk membatalkan perubahan terakhir, tekan u di Normal Mode. Untuk mengulangi perubahan yang dibatalkan, tekan `Ctrl + r`.

Dengan mengikuti panduan di atas, Anda telah menginstal dan melakukan konfigurasi dasar Vim, serta memahami beberapa perintah dasar yang sering digunakan. Pada tahap berikutnya, kita akan melanjutkan ke konfigurasi dan penggunaan editor lain yang mungkin Anda butuhkan dalam pengembangan proyek Django Anda.

2.7.3 Instalasi Visual Studio Code

Visual Studio Code (VSCode) adalah salah satu editor kode sumber terbuka yang paling populer dan kaya fitur yang dikembangkan oleh Microsoft. VSCode mendukung berbagai bahasa pemrograman dan dilengkapi dengan ekosistem ekstensi yang kuat, menjadikannya pilihan yang ideal untuk pengembangan proyek Django.

1. Unduh Visual Studio Code:

Kunjungi situs resmi Visual Studio Code di <https://code.visualstudio.com/> dan unduh versi terbaru untuk sistem operasi Anda.

2. Instal Visual Studio Code:

- **Di Windows:**

Persiapan Dan Instalasi Lingkungan

Setelah mengunduh file instalasi `.exe`, jalankan file tersebut dan ikuti instruksi di layar untuk menyelesaikan instalasi.

- **Di macOS:**

Setelah mengunduh file `.dmg`, buka file tersebut dan seret ikon Visual Studio Code ke folder Aplikasi.

- **Di Linux:**

Anda dapat menginstal Visual Studio Code menggunakan package manager atau mengunduh file `.deb` atau `.rpm` dari situs web mereka.

```
# Untuk distribusi berbasis Debian
$ sudo dpkg -i code_x.x.x_amd64.deb
```

```
# Untuk distribusi berbasis RPM
$ sudo rpm -i code-x.x.x.rpm
```

3. Menyiapkan Visual Studio Code untuk Pengembangan Django:

Setelah instalasi selesai, buka Visual Studio Code. Untuk mendukung pengembangan Django, Anda perlu menginstal beberapa ekstensi tambahan:

- **Python Extension:** Ekstensi ini memberikan dukungan untuk Python di Visual Studio Code, termasuk IntelliSense, linting, dan debugging.
- **Django Extension:** Ekstensi ini menambahkan fitur khusus untuk Django, seperti snippet dan template support.

Persiapan Dan Instalasi Lingkungan

Untuk menginstal ekstensi, buka Visual Studio Code dan pergi ke tab Extensions (ikon kotak di sidebar kiri) dan cari "Python" dan "Django". Klik tombol install pada ekstensi yang sesuai.

4. *Mengonfigurasi Visual Studio Code untuk Proyek Django:*

- ***Buka Proyek Django:***

Anda dapat membuka folder proyek Django Anda di Visual Studio Code dengan memilih **File > Open Folder** dan memilih folder proyek Anda.

- ***Menyiapkan Environment:***

Pastikan virtual environment yang Anda buat aktif. Anda dapat memilih interpreter Python yang sesuai di Visual Studio Code dengan menekan **Ctrl+Shift+P** (Windows/Linux) atau **Cmd+Shift+P** (macOS) dan mencari "Python: Select Interpreter". Pilih interpreter yang ada di dalam virtual environment Anda.

- ***Menggunakan Terminal Terintegrasi:***

Visual Studio Code menyediakan terminal terintegrasi yang memungkinkan Anda menjalankan perintah Django secara langsung dari dalam editor. Anda dapat membuka terminal dengan menekan **Ctrl+** (Windows/Linux) atau **Cmd+** (macOS).

Persiapan Dan Instalasi Lingkungan

Dengan Visual Studio Code terinstal dan dikonfigurasi, Anda siap untuk mulai menulis dan mengelola kode untuk proyek ***platform_bot*** Anda. Visual Studio Code akan menyediakan lingkungan pengembangan yang efisien dan menyenangkan, lengkap dengan fitur-fitur yang mendukung pengembangan Django secara efektif.

Dengan menginstal ekstensi yang tepat, VSCode menjadi alat yang sangat kuat untuk mengembangkan proyek Django. Setelah konfigurasi dan instalasi, Anda akan memiliki lingkungan pengembangan yang siap untuk memulai pengembangan proyek Django dengan efisiensi yang tinggi.

2.8 Menyiapkan Browser Dan Terminal/Command Line

Sebelum memulai pengembangan proyek Django, penting untuk memastikan bahwa Anda memiliki browser dan terminal atau command line yang siap digunakan. Alat-alat ini adalah komponen utama dalam siklus pengembangan dan pengujian proyek Django. Berikut adalah panduan untuk menyiapkan browser dan terminal Anda.

2.8.1 Browser

Browser adalah alat penting untuk melihat hasil dari aplikasi web yang Anda kembangkan. Semua interaksi dengan proyek Django yang telah berjalan akan dilakukan melalui browser.

1. ***Pilih Browser:***

Persiapan Dan Instalasi Lingkungan

- **Google Chrome:** Browser populer yang memiliki banyak alat pengembangan (Developer Tools) yang kuat.
- **Mozilla Firefox:** Dikenal karena alat pengembangan yang kaya dan dukungan yang baik untuk pengujian.
- **Microsoft Edge:** Berdasarkan Chromium, memiliki fitur pengembangan yang mirip dengan Chrome.
- **Safari:** Browser default di macOS, juga menyediakan alat pengembangan yang baik.

2. Mengaktifkan Developer Tools:

- Developer Tools adalah bagian dari browser yang memungkinkan Anda untuk memeriksa elemen halaman, melihat konsol, melacak jaringan, dan melakukan debugging JavaScript.
- Cara mengaktifkan Developer Tools:
 - **Google Chrome/Edge:** Tekan `Ctrl + Shift + I` (Windows/Linux) atau `Cmd + Option + I` (macOS).
 - **Mozilla Firefox:** Tekan `Ctrl + Shift + I` (Windows/Linux) atau `Cmd + Option + I` (macOS).
 - **Safari:** Anda perlu mengaktifkan menu Developer melalui pengaturan: buka **Safari > Preferences > Advanced**, lalu centang "Show Develop menu in menu bar".

3. Testing Responsiveness:

- Semua browser modern memiliki fitur untuk menguji responsivitas situs Anda. Ini penting un-

Persiapan Dan Instalasi Lingkungan

tuk memastikan situs Anda berfungsi dengan baik di berbagai perangkat.

- Akses mode responsif di Developer Tools untuk mensimulasikan bagaimana situs akan terlihat di perangkat seperti ponsel dan tablet.

2.8.2 Terminal/Command Line

Terminal atau command line adalah alat yang digunakan untuk menjalankan perintah, seperti menjalankan server Django, mengelola paket Python, dan bekerja dengan sistem file.

1. *Terminal di Windows:*

- **Command Prompt:** Alat bawaan di Windows yang dapat diakses dengan mencari "cmd" di Start Menu.
- **PowerShell:** Alat yang lebih canggih dengan lebih banyak fitur, juga dapat ditemukan di Start Menu.
- **Windows Terminal:** Terminal modern yang mendukung banyak tab dan profil, termasuk PowerShell dan Command Prompt. Anda dapat menginstalnya dari Microsoft Store.

Membuka Command Prompt:

```
$ cmd
```

Membuka PowerShell:

```
$ powershell
```

Membuka Windows Terminal:

Persiapan Dan Instalasi Lingkungan

```
$ wt
```

2. *Terminal di macOS:*

- **Terminal:** Aplikasi bawaan macOS yang dapat diakses melalui Spotlight atau folder `Applications > Utilities > Terminal`.

Membuka Terminal:

```
$ terminal
```

3. *Terminal di Linux:*

- **Terminal:** Tersedia di semua distribusi Linux, biasanya dapat diakses dari menu aplikasi atau dengan menekan `Ctrl + Alt + T`.

Membuka Terminal:

```
$ gnome-terminal
```

4. *Tips Menggunakan Terminal:*

- **Navigasi Sistem File:** Gunakan perintah seperti `cd`, `ls`, dan `pwd` untuk berpindah direktori dan melihat konten.
- **Mengedit File:** Pada banyak sistem, Anda bisa menggunakan editor teks seperti `nano`, `vim`, atau `code` untuk mengedit file langsung dari terminal.
- **Menjalankan Perintah Django:** Setelah proyek Django dibuat, banyak interaksi akan terjadi me-

Persiapan Dan Instalasi Lingkungan

lalui terminal, seperti menjalankan server (\$ python manage.py runserver).

5. *Memperbarui dan Menginstal Paket:*

- *Windows:*

```
$ python -m pip install --upgrade  
pip
```

- *macOS/Linux:*

```
$ python3 -m pip install --upgrade  
pip
```

Dengan browser dan terminal yang siap digunakan, Anda akan memiliki alat yang diperlukan untuk memulai pengembangan proyek Django Anda. Kedua alat ini akan menjadi bagian integral dari alur kerja Anda, memungkinkan Anda untuk menjalankan, menguji, dan mengembangkan aplikasi secara efisien.

2.9 Verifikasi Lingkungan

2.9.1 Menguji Instalasi Django

Setelah semua langkah instalasi telah selesai, penting untuk memverifikasi bahwa Django telah diinstal dengan benar dan dapat berjalan tanpa masalah. Pada bagian ini, kita akan menjalankan server pengembangan bawaan Django dan memeriksa apakah aplikasi Django dapat diakses melalui browser.

Persiapan Dan Instalasi Lingkungan

1. Menjalankan Server Pengembangan Django

Django menyediakan server pengembangan bawaan yang sangat berguna untuk menguji aplikasi selama proses pengembangan. Untuk menjalankan server ini, Anda harus terlebih dahulu memastikan bahwa Anda berada di direktori proyek Django tempat Anda membuat proyek sebelumnya.

Jika virtual environment Anda belum aktif, aktifkan terlebih dahulu dengan perintah berikut:

```
$ source venv/bin/activate
```

Setelah virtual environment aktif, navigasi ke direktori proyek Django Anda. Misalnya, jika proyek Django Anda bernama **platform_bot**, masuk ke dalam folder tersebut:

```
$ cd platform_bot
```

Pastikan bahwa struktur proyek Django Anda sudah benar, dengan file dan folder utama seperti `manage.py`, `settings.py`, dan `urls.py` ada di dalamnya. Kemudian, jalankan server pengembangan Django menggunakan perintah berikut:

```
$ python manage.py runserver
```

Perintah ini akan memulai server pengembangan pada alamat `http://127.0.0.1:8000/` secara default. Anda akan melihat output di terminal yang mengonfirmasi bahwa server berjalan dengan sukses, seperti berikut:

Persiapan Dan Instalasi Lingkungan

```
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
September 16, 2024 - 14:30:10
Django version 4.2.0, using settings
'platform_bot.settings'
Starting development server at
http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

2. Memeriksa Akses ke Aplikasi

Setelah server pengembangan berjalan, buka browser Anda dan akses alamat `http://127.0.0.1:8000/`. Jika instalasi Django berhasil dan server berjalan dengan baik, Anda akan melihat halaman selamat datang Django yang menampilkan pesan "The install worked successfully! Congratulations!".

Halaman ini mengindikasikan bahwa Django telah diinstal dan dikonfigurasi dengan benar. Jika Anda tidak melihat halaman ini, atau jika ada pesan error di terminal atau di browser, periksa kembali konfigurasi dan pastikan semua dependensi terinstal dengan benar.

Jika server tidak bisa diakses atau muncul error, berikut adalah beberapa langkah pengecekan:

- Pastikan port 8000 tidak digunakan oleh aplikasi lain.
- Pastikan semua library dan dependensi telah diinstal dengan benar dalam virtual environment.
- Lihat log error di terminal untuk pesan yang lebih spesifik.

Persiapan Dan Instalasi Lingkungan

Dengan berhasilnya pengujian server ini, berarti lingkungan pengembangan Django sudah siap untuk digunakan dalam proyek ***platform_bot***. Anda sekarang bisa mulai membangun aplikasi dan mengembangkan berbagai fitur yang diinginkan.

2.9.2 Menguji Ngrok dan Webhook

Setelah berhasil menguji Django, langkah berikutnya adalah memastikan bahwa Ngrok berjalan dengan baik dan dapat digunakan untuk menghubungkan aplikasi Django lokal Anda dengan internet melalui webhook. Ngrok sangat membantu ketika Anda mengembangkan bot yang membutuhkan webhook, seperti bot Telegram atau WhatsApp, sehingga layanan eksternal dapat berkomunikasi dengan server lokal Anda.

Memastikan Ngrok Bekerja dengan Baik

Pertama, kita akan menjalankan Ngrok untuk memastikan bahwa terowongan ke server lokal Django dapat dibuka dan diakses dari internet. Ngrok memungkinkan kita untuk membuat URL publik yang mengarahkan ke server pengembangan lokal pada port tertentu.

Jalankan perintah berikut untuk memulai Ngrok dan membuat terowongan ke server pengembangan Django yang berjalan di port 8000:

```
$ ngrok http 8000
```

Persiapan Dan Instalasi Lingkungan

Ngrok akan mengeluarkan output di terminal yang menampilkan URL publik yang bisa diakses dari internet, misalnya:

```
Session Status      online
Account
your_email@example.com (Plan: Free)
Version             2.3.40
Region              United States (us)
Web Interface
http://127.0.0.1:4040
Forwarding
http://abcdef1234.ngrok.io ->
http://localhost:8000
Forwarding
https://abcdef1234.ngrok.io ->
http://localhost:8000
```

Di sini, URL publik `https://abcdef1234.ngrok.io` adalah alamat yang bisa digunakan untuk mengakses server lokal Django melalui internet. Pastikan untuk menyimpan URL ini, karena akan digunakan untuk webhook bot.

Menguji Webhook untuk Bot

Webhook memungkinkan bot berkomunikasi dengan server Anda saat ada pesan atau perintah yang dikirimkan pengguna. Untuk menguji webhook, kita akan menghubungkan URL Ngrok yang telah dibuat dengan aplikasi bot Anda.

Jika Anda menggunakan bot Telegram, Anda bisa mengatur webhook bot dengan menggunakan perintah API Telegram. Misalnya, gunakan perintah berikut untuk menghubungkan webhook bot dengan URL Ngrok:

Persiapan Dan Instalasi Lingkungan

```
$ curl -F  
"url=https://abcdef1234.ngrok.io/webhook/"  
https://api.telegram.org/bot<YOUR_BOT_TOKEN>/set  
Webhook
```

Gantilah <YOUR_BOT_TOKEN> dengan token bot Telegram Anda, dan pastikan bahwa endpoint `/webhook/` di aplikasi Django Anda sudah dikonfigurasi untuk menangani pesan dari bot. Jika webhook berhasil diatur, Telegram akan mengirimkan data ke endpoint `/webhook/` setiap kali ada pesan baru.

Selanjutnya, buka terminal Django dan pastikan server pengembangan berjalan. Jika webhook berfungsi dengan baik, Anda akan melihat request dari Telegram masuk ke server Anda setiap kali bot menerima pesan. Request ini dapat dilihat di terminal Django atau di interface Ngrok di `http://127.0.0.1:4040` yang menampilkan log request.

Contoh log request di Ngrok bisa terlihat seperti ini:

```
POST /webhook/ 200 OK
```

Jika Anda melihat log seperti di atas, itu berarti webhook berhasil dipasang dan bot Anda sudah bisa berkomunikasi dengan server Django melalui Ngrok.

Jika ada error, periksa log terminal Ngrok dan Django untuk mengetahui apakah ada kesalahan pada endpoint webhook atau konfigurasi bot.

Persiapan Dan Instalasi Lingkungan

Dengan berhasilnya pengujian ini, Anda telah menyelesaikan langkah verifikasi bahwa Ngrok dan webhook bekerja dengan baik. Bot Anda sekarang siap menerima dan merespons pesan dari platform seperti Telegram atau WhatsApp.

Kesimpulan Bab

Pada Bab ini, kita telah melalui berbagai tahapan penting untuk menyiapkan lingkungan pengembangan dan menjalankan proyek **platform_bot**. Mulai dari instalasi Python, pengaturan virtual environment, hingga instalasi Django, dan pengaturan alat bantu seperti Ngrok dan editor kode. , setiap langkah dirancang untuk memastikan bahwa Anda memiliki fondasi yang kuat untuk melanjutkan pengembangan bot berbasis webhook dengan Django.

Berikut adalah ringkasan dari langkah-langkah yang telah kita lakukan:

1. *Instalasi Python*

Kita memulai dengan menginstal Python, yang menjadi dasar bagi pengembangan aplikasi Django dan bot. Versi Python yang digunakan harus sesuai dengan kebutuhan Django dan framework bot yang dipilih.

2. *Mengatur Virtual Environment*

Untuk menjaga konsistensi dan mencegah konflik antar-dependensi proyek, kita mengatur virtual environment. Ini memastikan bahwa setiap proyek memiliki dependensi yang terpisah dan tidak saling mengganggu.

3. *Instalasi Django*

Setelah virtual environment siap, kita menginstal Django, framework utama yang akan kita gunakan untuk membangun platform bot. Instalasi ini diverifikasi dengan menjalankan server pengembangan dan memastikan bahwa aplikasi Django dapat diakses melalui browser.

Persiapan Dan Instalasi Lingkungan

4. *Pengaturan Ngrok*

Ngrok digunakan untuk membuat server lokal Django kita dapat diakses dari internet. Dengan Ngrok, kita bisa menghubungkan webhook dari platform seperti Telegram atau WhatsApp ke server lokal kita, sehingga bot dapat menerima dan merespons pesan dari pengguna.

5. *Menguji Ngrok dan Webhook*

Setelah Ngrok terhubung, kita memastikan bahwa webhook berfungsi dengan baik, dan bot dapat menerima request dari platform eksternal. Ngrok membantu menghubungkan server pengembangan dengan layanan bot, dan pengujian webhook memastikan bahwa koneksi ini berjalan dengan lancar.

Tips dan Saran untuk Pemecahan Masalah:

1. **Masalah Koneksi Ngrok:** Jika Ngrok tidak dapat membuat terowongan atau tidak dapat diakses dari luar, pastikan bahwa firewall atau konfigurasi jaringan Anda tidak memblokir akses ke port yang digunakan. Selain itu, periksa apakah server lokal Anda berjalan di port yang benar (8000 untuk Django secara default).
2. **Kesalahan Webhook:** Jika webhook tidak berfungsi atau bot tidak merespons pesan, periksa kembali URL yang digunakan saat mengatur webhook, pastikan URL Ngrok yang diberikan adalah URL HTTPS. Selain itu, periksa endpoint webhook di aplikasi Django apakah sudah dikonfigurasi dengan benar untuk menerima request dari platform bot.

Persiapan Dan Instalasi Lingkungan

3. ***Dependensi yang Tidak Kompatibel:*** Jika Anda mengalami masalah saat menginstal paket Python atau dependensi lain, periksa versi Python dan Django yang Anda gunakan. Seringkali masalah kompatibilitas muncul karena versi paket yang tidak sesuai. Coba gunakan virtual environment untuk memastikan setiap proyek memiliki lingkungan yang bersih dan terisolasi.
4. ***Pengujian Manual Server Django:*** Jika server pengembangan Django tidak bisa dijalankan, periksa apakah ada masalah di dalam kode atau pengaturan proyek. Cek file `settings.py` untuk memastikan semua konfigurasi sudah benar, dan periksa log terminal untuk pesan error yang mungkin membantu dalam pemecahan masalah.

Dengan semua langkah yang telah dilakukan, Anda sekarang memiliki dasar yang kuat untuk melanjutkan pengembangan platform bot Anda. Pastikan untuk selalu melakukan pengujian secara berkala selama proses pengembangan dan mengikuti best practice untuk menjaga kualitas dan keamanan proyek Anda.

BAB 3 - Memahami dan Menggunakan Django Admin

Dalam bab ini, kita akan mengeksplorasi Django Admin, sebuah alat yang sangat berguna untuk pengelolaan data dalam aplikasi Django. Django Admin berfungsi sebagai panel administrasi bawaan yang memungkinkan developer dan administrator untuk mengelola model dan data dengan mudah dan efisien. Kita akan mulai dengan pengantar tentang apa itu Django Admin, termasuk manfaat dan fitur utamanya. Selanjutnya, kita akan membahas langkah-langkah untuk membuat superuser yang akan memberikan akses penuh ke panel admin.

Setelah itu, kita akan melihat cara mendaftarkan model ke dalam Django Admin dan bagaimana kita bisa mempersonalisasi tampilan serta menambahkan fitur-fitur tambahan untuk meningkatkan pengalaman pengguna. Selain itu, kita juga akan membahas aspek keamanan, termasuk cara mengelola izin pengguna dan membatasi akses ke panel admin. Terakhir, kita akan mengidentifikasi masalah umum yang mungkin dihadapi saat menggunakan Django Admin dan solusi untuk mengatasinya.

Dengan pemahaman yang mendalam tentang Django Admin, kita dapat memaksimalkan potensi aplikasi Django kita dalam pengelolaan data, menjadikannya lebih efisien dan user-friendly.

Memahami dan Menggunakan Django Admin

3.1 Apa Itu Django Admin?

Django Admin merupakan panel administrasi bawaan yang disediakan oleh framework Django. Sebagai bagian integral dari Django, Admin dirancang untuk memudahkan pengelolaan data dan model yang digunakan dalam aplikasi web. Dengan antarmuka yang intuitif dan mudah digunakan, Django Admin memungkinkan pengembang dan administrator untuk melakukan berbagai tugas administratif, seperti menambah, mengedit, atau menghapus data, tanpa perlu menulis kode tambahan.

Manfaat utama dari menggunakan Django Admin adalah kemampuannya untuk mempercepat proses pengelolaan aplikasi. Dengan hanya beberapa langkah konfigurasi, pengguna dapat mengakses dan mengelola model-model yang telah didefinisikan dalam aplikasi. Misalnya, setelah membuat model, cukup dengan menambahkan kode sederhana pada file `admin.py`, model tersebut akan langsung tersedia dalam antarmuka Admin. Penggunaan Django Admin mengurangi kebutuhan untuk membuat antarmuka pengguna kustom untuk pengelolaan data, yang seringkali memakan waktu dan sumber daya.

Ketika menggunakan Django Admin, pengguna mendapatkan fitur-fitur canggih seperti pencarian, filter, dan pengelolaan hak akses pengguna. Fitur-fitur ini sangat berguna untuk mengatur data dalam jumlah besar dan memastikan bahwa hanya pengguna tertentu yang memiliki hak akses untuk melakukan tindakan tertentu. Selain itu, Django Admin juga memungkinkan pengguna untuk mengustomisasi tampilan dan fungsionalitas sesuai kebutuhan.

Memahami dan Menggunakan Django Admin

an, sehingga memberikan fleksibilitas yang lebih dalam pengelolaan aplikasi.

Dengan pemahaman yang mendalam tentang Django Admin, pengguna dapat memaksimalkan potensinya dan meningkatkan efisiensi dalam mengelola aplikasi web yang mereka kembangkan. Melalui bab ini, kita akan menjelajahi lebih lanjut tentang bagaimana cara mengatur dan menggunakan Django Admin dengan efektif, mulai dari pembuatan superuser hingga personalisasi antarmuka.

3.2 Fitur Utama Django Admin

Django Admin dilengkapi dengan berbagai fitur canggih yang mendukung pengelolaan data secara efisien. Fitur-fitur ini tidak hanya mempermudah administrator dalam mengelola model, tetapi juga memberikan kemudahan bagi pengguna untuk berinteraksi dengan data dalam aplikasi. Berikut ini adalah beberapa fitur utama yang disediakan oleh Django Admin.

Salah satu fitur paling penting dari Django Admin adalah manajemen model. Setelah model didefinisikan dalam aplikasi, pengguna dapat dengan mudah mengelolanya melalui antarmuka Admin. Dengan menambahkan beberapa baris kode pada file `admin.py`, model akan secara otomatis tersedia dalam Django Admin. Misalnya, untuk mendaftarkan model `Blog` yang telah dibuat, pengguna hanya perlu menulis kode berikut:

```
# admin.py
```

Memahami dan Menggunakan Django Admin

```
from django.contrib import admin
from .models import Blog # Mengimpor model Blog

admin.site.register(Blog) # Mendaftarkan model
Blog ke Django Admin
```

Setelah kode tersebut ditambahkan, pengguna dapat langsung mengakses model `Blog` melalui panel Admin, melakukan operasi seperti menambah, mengedit, atau menghapus entri dengan mudah.

Django Admin juga menawarkan fitur pencarian yang sangat berguna. Dengan fitur ini, pengguna dapat mencari entri tertentu dalam model dengan cepat dan efisien. Pengembang dapat menambahkan fungsionalitas pencarian pada model dengan mendefinisikan atribut `search_fields` di dalam kelas Admin. Misalnya, jika model `Blog` memiliki atribut `title` dan `content`, pengguna dapat menambahkan kode berikut untuk memungkinkan pencarian berdasarkan kedua atribut tersebut:

```
# admin.py
class BlogAdmin(admin.ModelAdmin):
    search_fields = ['title', 'content'] #
    Menentukan field yang bisa dicari

admin.site.register(Blog, BlogAdmin) #
Mendaftarkan model Blog dengan konfigurasi
pencarian
```

Fitur filter juga menjadi salah satu kekuatan utama Django Admin. Dengan fitur ini, pengguna dapat memfilter entri berdasarkan kriteria tertentu, sehingga memudahkan pencarian data yang relevan dalam jumlah besar. Pengembang dapat menambahkan

Memahami dan Menggunakan Django Admin

fitur filter dengan menggunakan atribut `list_filter` pada kelas Admin. Sebagai contoh, jika model `Blog` memiliki field `published_date`, pengguna dapat mengatur filter berdasarkan tanggal publikasi dengan menambahkan kode berikut:

```
# admin.py
class BlogAdmin(admin.ModelAdmin):
    list_filter = ['published_date'] #
    Menentukan field yang dapat digunakan untuk
    memfilter

admin.site.register(Blog, BlogAdmin) #
Mendaftarkan model Blog dengan konfigurasi
filter
```

Fitur-fitur ini membuat Django Admin menjadi alat yang sangat kuat dan fleksibel untuk pengelolaan data. Selain itu, Django Admin juga memungkinkan pengguna untuk mengatur hak akses dengan lebih baik. Dengan manajemen pengguna dan grup, administrator dapat menentukan siapa yang memiliki akses ke bagian tertentu dari aplikasi, sehingga meningkatkan keamanan dan pengelolaan data.

Secara keseluruhan, fitur-fitur yang ditawarkan oleh Django Admin sangat membantu dalam mempercepat proses pengelolaan aplikasi web. Dengan memahami dan memanfaatkan fitur-fitur ini, pengguna dapat lebih efektif dalam mengelola data, meningkatkan produktivitas, dan menjamin keamanan aplikasi yang mereka kembangkan. Dalam bab-bab selanjutnya, kita akan menjelajahi lebih dalam mengenai bagaimana cara mengimplementasikan fitur-fitur ini secara praktis dalam aplikasi Django.

Memahami dan Menggunakan Django Admin

3.3 Membuat Superuser

Dalam bagian ini, kita akan membahas langkah-langkah untuk membuat superuser, yang merupakan langkah penting untuk mengakses Django Admin dengan penuh otorisasi. Superuser adalah akun pengguna dengan hak akses tertinggi dalam sistem, memungkinkan pengguna untuk melakukan semua tindakan di Django Admin, termasuk manajemen model, pengaturan izin, dan pengelolaan data lainnya.

3.3.1 Apa Itu Superuser?

Superuser dalam Django berfungsi sebagai administrator utama yang memiliki kendali penuh atas semua aspek aplikasi. Dengan kata lain, superuser dapat mengakses dan mengelola semua model yang terdaftar dalam Django Admin, serta melakukan tindakan yang mungkin dibatasi untuk pengguna biasa. Hal ini sangat berguna dalam konteks aplikasi yang memiliki banyak pengguna atau data yang kompleks, di mana hanya orang tertentu yang seharusnya memiliki akses untuk melakukan perubahan penting.

Dalam konteks Django Admin, superuser adalah pengguna dengan hak akses tertinggi yang memiliki kemampuan untuk mengelola semua aspek dari aplikasi. Peran superuser sangat penting karena ia dapat melakukan tindakan administratif tanpa batasan, termasuk mengelola model, pengguna, dan konfigurasi aplikasi lainnya. Superuser biasanya digunakan oleh pengembang atau administrator sistem untuk mengawasi dan mengelola data, serta untuk memastikan bahwa aplikasi berjalan dengan baik.

Memahami dan Menggunakan Django Admin

Saat membuat superuser, pengguna akan diberikan akses penuh ke antarmuka Django Admin. Ini berarti bahwa mereka dapat menambah, mengedit, dan menghapus entri di semua model yang telah terdaftar. Sebagai contoh, jika aplikasi memiliki model **B**log, superuser dapat membuat dan mengelola postingan, mengedit kategori, serta melihat semua data yang terkait dengan model tersebut. Dengan demikian, superuser berfungsi sebagai pengelola utama aplikasi, memastikan semua proses administratif dapat dilakukan dengan efisien.

Peran superuser juga mencakup pengelolaan pengguna lain. Dengan hak akses yang dimiliki, superuser dapat membuat pengguna baru, mengubah izin akses mereka, dan menambahkan mereka ke dalam grup tertentu. Hal ini sangat penting dalam konteks aplikasi yang lebih besar, di mana mungkin ada banyak pengguna dengan berbagai tingkat akses. Dengan demikian, superuser berfungsi sebagai pengawas yang memastikan bahwa setiap pengguna memiliki hak akses yang sesuai dan bertanggung jawab terhadap pengelolaan data.

Secara keseluruhan, keberadaan superuser dalam Django Admin adalah kunci untuk pengelolaan aplikasi yang efektif. Dengan pemahaman yang jelas tentang peran dan fungsinya, pengguna dapat memanfaatkan sepenuhnya kemampuan Django Admin untuk menjaga data tetap aman dan terorganisir dengan baik. Dalam sub-bab selanjutnya, kita akan membahas langkah-langkah konkret untuk membuat superuser dan bagaimana cara login ke dalam antarmuka Django Admin.

Memahami dan Menggunakan Django Admin

3.3.2 Langkah-Langkah Membuat Superuser

Untuk memanfaatkan sepenuhnya fitur-fitur yang ditawarkan oleh Django Admin, langkah pertama yang perlu dilakukan adalah membuat superuser. Proses ini sangat sederhana dan dapat diselesaikan dalam beberapa langkah. Berikut ini adalah panduan langkah demi langkah untuk membuat superuser menggunakan terminal.

Pertama-tama, pastikan bahwa server Django berjalan dan bahwa database telah terkonfigurasi dengan benar. Jika kamu menggunakan SQLite, database biasanya akan dibuat secara otomatis saat proyek pertama kali dijalankan. Namun, jika menggunakan database lain seperti PostgreSQL atau MySQL, pastikan bahwa koneksi telah diatur dengan baik di file `settings.py`.

Setelah memastikan semua persyaratan telah dipenuhi, buka terminal dan navigasikan ke direktori proyek Django kamu. Di dalam direktori proyek, jalankan perintah berikut untuk memulai proses pembuatan superuser:

```
$ python manage.py createsuperuser # Perintah  
untuk membuat superuser
```

Setelah menjalankan perintah ini, sistem akan meminta beberapa informasi penting untuk membuat akun superuser. Pertama, kamu akan diminta untuk memasukkan username. Username ini akan digunakan untuk login ke Django Admin, jadi pilihlah nama yang mudah diingat dan unik.

Memahami dan Menggunakan Django Admin

Setelah itu, sistem akan meminta alamat email. Meskipun email ini tidak selalu diperlukan untuk akses ke Django Admin, sangat dianjurkan untuk memberikan email yang valid. Email ini dapat digunakan untuk pemulihan password di masa depan atau untuk menerima notifikasi terkait akun.

Setelah memasukkan email, kamu akan diminta untuk membuat password. Password ini harus cukup kuat untuk menjaga keamanan aplikasi. Django akan memvalidasi kekuatan password yang kamu buat, jadi pastikan untuk mengikuti pedoman keamanan yang diberikan. Jika password yang kamu masukkan tidak memenuhi syarat, kamu akan diminta untuk memasukkan password yang baru.

Berikut adalah contoh interaksi di terminal saat membuat superuser:

```
Username (leave blank to use 'admin'): admin #  
Memasukkan username  
Email address: admin@example.com # Memasukkan email  
Password: # Memasukkan password  
Password (again): # Memasukkan password sekali lagi untuk konfirmasi
```

Setelah semua informasi dimasukkan dan password terverifikasi, superuser akan berhasil dibuat. Kamu akan melihat pesan konfirmasi yang menyatakan bahwa superuser telah dibuat dengan sukses.

Memahami dan Menggunakan Django Admin

Dengan superuser yang telah dibuat, kamu kini dapat login ke Django Admin menggunakan username dan password yang telah ditentukan. Buka browser dan masukkan URL berikut:

```
http://localhost:8000/admin # URL untuk mengakses Django Admin
```

Setelah halaman login terbuka, masukkan username dan password yang telah kamu buat sebelumnya. Jika semua informasi benar, kamu akan diarahkan ke dashboard Django Admin, di mana kamu dapat mulai mengelola model dan data aplikasi.

Membuat superuser adalah langkah awal yang krusial untuk memanfaatkan Django Admin dengan efektif. Dengan superuser yang siap, kamu kini memiliki kontrol penuh atas aplikasi dan dapat melakukan berbagai tindakan administratif dengan mudah. Selanjutnya, kita akan membahas cara login ke Django Admin dan menjelajahi antarmukanya.

3.3.3 Login ke Django Admin dengan Superuser

Setelah berhasil membuat superuser, langkah selanjutnya adalah melakukan login ke Django Admin untuk mulai mengelola aplikasi. Proses login ini sangat sederhana dan memungkinkan kamu untuk mengakses semua fitur yang tersedia di antarmuka administrasi.

Pertama, pastikan bahwa server Django sedang berjalan. Jika belum menjalankannya, buka terminal dan navigasikan ke direktori

Memahami dan Menggunakan Django Admin

proyek Django kamu. Kemudian, jalankan perintah berikut untuk memulai server:

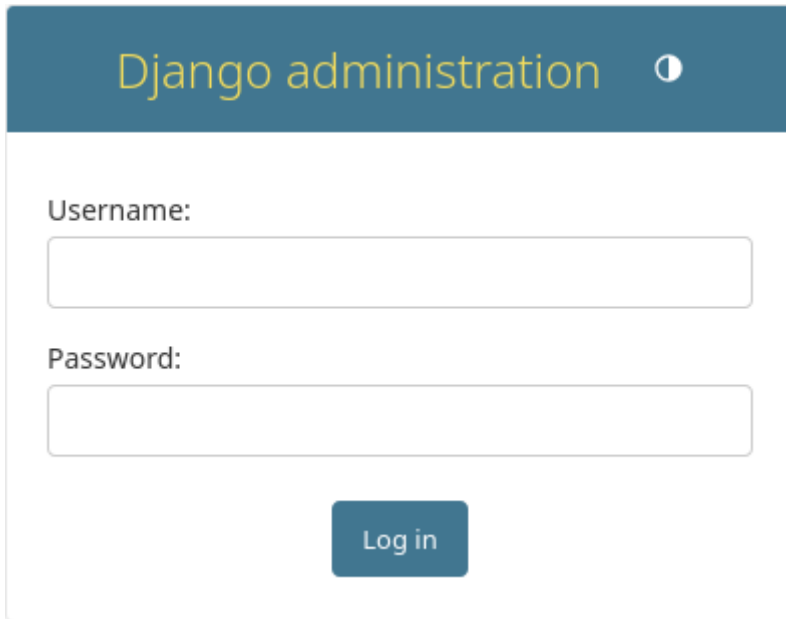
```
$ python manage.py runserver # Perintah untuk menjalankan server Django
```

Setelah server aktif, buka browser dan ketikkan URL berikut untuk mengakses halaman login Django Admin:

```
http://localhost:8000/admin # URL untuk mengakses Django Admin
```

Setelah halaman login terbuka, kamu akan melihat formulir yang meminta username dan password. Masukkan username dan password yang telah kamu buat saat proses pembuatan superuser. Contoh tampilan formulir login adalah sebagai berikut:

Memahami dan Menggunakan Django Admin

The image shows the Django administration login interface. At the top, there is a dark blue header bar with the text "Django administration" in a light yellow font, followed by a small white circular icon. Below the header, the page has a light gray background. It features two input fields: the first is labeled "Username:" and the second is labeled "Password:". Both labels are in a dark gray font. Below the password field is a blue rectangular button with the text "Log in" in white. The entire form is enclosed in a thin gray border.

Setelah memasukkan informasi yang benar, klik tombol "Log in" untuk masuk ke dalam Django Admin. Jika semua data yang dimasukkan benar, kamu akan diarahkan ke dashboard utama Django Admin.

Tampilan awal dashboard ini memberikan gambaran umum mengenai model yang telah terdaftar dalam aplikasi. Di sisi kiri layar, terdapat menu navigasi yang mencantumkan semua model yang tersedia. Kamu akan melihat kategori berdasarkan model yang telah kamu daftarkan, seperti `Blog`, `Users`, atau model la-

Memahami dan Menggunakan Django Admin

innya. Masing-masing kategori dapat diperluas untuk menampilkan entri yang telah dibuat dalam model tersebut.

Di bagian atas, terdapat beberapa opsi navigasi, termasuk kemampuan untuk logout dan akses ke dokumentasi Django. Dashboard ini juga menyediakan fitur pencarian dan filter yang memungkinkan kamu untuk dengan cepat menemukan data tertentu.

Salah satu aspek menarik dari antarmuka Django Admin adalah kesederhanaannya. Meskipun memiliki banyak fitur, desainnya tetap bersih dan intuitif, memudahkan pengguna baru untuk memahami bagaimana cara mengelola data dengan efektif. Pada bagian kanan, terdapat tombol yang memungkinkan kamu untuk menambah entri baru untuk setiap model, sehingga memudahkan dalam melakukan operasi dasar seperti penambahan, pengeditan, atau penghapusan data.

Setelah login, kamu dapat menjelajahi berbagai bagian dari Django Admin, mulai dari mengelola data, melakukan pencarian, hingga menggunakan fitur filter yang telah dijelaskan sebelumnya. Dengan superuser yang aktif, kamu memiliki kontrol penuh atas aplikasi, dan ini membuka banyak kemungkinan untuk pengelolaan yang lebih efisien.

Dengan langkah-langkah ini, kamu telah berhasil login ke Django Admin menggunakan superuser. Selanjutnya, kita akan membahas bagaimana menambahkan model ke dalam Django Admin agar dapat dikelola dengan lebih baik.

Memahami dan Menggunakan Django Admin

3.4 Menambahkan Model Ke Django Admin

Setelah berhasil login ke Django Admin, langkah selanjutnya yang perlu dilakukan adalah mendaftarkan model yang telah dibuat agar dapat dikelola melalui antarmuka administrasi. Proses ini cukup sederhana dan melibatkan perubahan pada file `admin.py` dalam aplikasi yang bersangkutan.

3.4.1 Mendaftarkan Model ke Django Admin

Model dalam Django adalah representasi dari struktur data yang akan digunakan dalam aplikasi. Setelah model didefinisikan, kamu perlu memberi tahu Django bahwa model tersebut harus tersedia di dalam Django Admin. Untuk melakukannya, buka file `admin.py` yang terletak dalam folder aplikasi. Jika file ini belum ada, kamu dapat membuatnya.

Misalnya, jika kita memiliki model bernama `Blog` yang telah didefinisikan di file `models.py`, kita perlu mendaftarkan model tersebut di file `admin.py`. Berikut adalah contoh sederhana dari kode yang diperlukan:

```
# admin.py
from django.contrib import admin
from .models import Blog # Mengimpor model Blog

admin.site.register(Blog) # Mendaftarkan model
Blog ke Django Admin
```

Pada kode di atas, pertama-tama kita mengimpor kelas `admin` dari `django.contrib`. Kemudian, kita juga mengimpor mo-

Memahami dan Menggunakan Django Admin

del `Blog` yang telah kita buat sebelumnya. Dengan menggunakan metode `admin.site.register()`, kita mendaftarkan model `Blog` ke dalam Django Admin.

Setelah menyimpan perubahan pada file `admin.py`, kamu perlu memastikan bahwa server Django masih berjalan. Jika belum, jalankan perintah berikut di terminal:

```
$ python manage.py runserver # Menjalankan  
server Django
```

Setelah server berjalan, buka kembali antarmuka Django Admin di browser dengan URL:

```
http://localhost:8000/admin # URL untuk  
mengakses Django Admin
```

Setelah login, kamu akan melihat model `Blog` yang baru saja didaftarkan muncul di menu navigasi di sebelah kiri. Mengklik model tersebut akan membawa kamu ke halaman yang menampilkan semua entri yang ada dalam model `Blog`. Jika saat ini belum ada entri, kamu akan melihat opsi untuk menambah entri baru.

Dengan model yang terdaftar, kamu kini dapat melakukan berbagai operasi seperti menambah, mengedit, dan menghapus postingan blog langsung dari antarmuka Django Admin. Ini sangat mempermudah pengelolaan data dan meningkatkan efisiensi dalam pengembangan aplikasi.

Memahami dan Menggunakan Django Admin

Pendaftaran model ke Django Admin adalah langkah penting yang memungkinkan administrator untuk mengelola data secara efektif. Dengan memahami cara mendaftarkan model, kamu dapat memperluas fungsionalitas aplikasi dengan mudah, serta memanfaatkan kekuatan Django Admin untuk meningkatkan produktivitas dalam pengelolaan data. Dalam sub-bab berikutnya, kita akan membahas cara personalisasi tampilan model di Django Admin agar lebih sesuai dengan kebutuhan aplikasi.

3.4.2 Personalisasi Tampilan Model di Admin

Setelah mendaftarkan model ke dalam Django Admin, langkah selanjutnya adalah mempersonalisasi tampilan model tersebut agar lebih mudah digunakan dan informatif. Django menyediakan beberapa fitur yang memungkinkan kita untuk mengontrol tampilan dan fungsionalitas model di antarmuka Admin, seperti `list_display`, `list_filter`, dan `search_fields`. Dengan menggunakan fitur-fitur ini, kita dapat meningkatkan pengalaman pengguna saat mengelola data.

Untuk memulai, buka kembali file `admin.py` di mana model Blog telah didaftarkan. Untuk mempersonalisasi tampilan model, kita perlu membuat subclass dari `admin.ModelAdmin` dan mengoverride atribut yang ingin kita modifikasi. Berikut adalah contoh bagaimana cara menambahkan kontrol atas tampilan model Blog:

```
# admin.py
from django.contrib import admin
from .models import Blog # Mengimpor model Blog
```


Memahami dan Menggunakan Django Admin

```
class BlogAdmin(admin.ModelAdmin):
    list_display = ('title', 'author',
                    'created_at') # Menampilkan kolom tertentu
    list_filter = ('author', 'created_at') #
    # Menambahkan filter berdasarkan penulis dan
    # tanggal
    search_fields = ('title', 'content') #
    # Memungkinkan pencarian berdasarkan judul dan
    # konten

admin.site.register(Blog, BlogAdmin) #
# Mendaftarkan model Blog dengan personalisasi
```

Dalam kode di atas, kita mendefinisikan kelas `BlogAdmin` yang merupakan subclass dari `admin.ModelAdmin`. Dengan menggunakan atribut `list_display`, kita dapat menentukan kolom mana saja yang ingin ditampilkan dalam daftar entri model `Blog`. Pada contoh ini, kita menampilkan judul, penulis, dan tanggal dibuat. Ini memberikan informasi yang lebih jelas dan komprehensif kepada pengguna saat melihat daftar entri.

Selanjutnya, atribut `list_filter` memungkinkan kita untuk menambahkan filter di sisi kanan halaman daftar. Dengan filter ini, pengguna dapat dengan mudah menyaring entri berdasarkan penulis dan tanggal. Ini sangat berguna, terutama jika terdapat banyak entri dalam model.

Terakhir, atribut `search_fields` memberikan kemampuan pencarian yang lebih baik dengan memungkinkan pengguna mencari entri berdasarkan judul dan konten. Ini sangat membantu saat mencari informasi spesifik di antara banyak entri, sehingga mempercepat proses manajemen data.

Memahami dan Menggunakan Django Admin

Setelah melakukan perubahan ini, pastikan untuk menyimpan file `admin.py` dan, jika perlu, restart server Django agar perubahan dapat diterapkan. Kamu dapat melakukannya dengan perintah berikut di terminal:

```
$ python manage.py runserver # Menjalankan kembali server Django
```

Setelah server aktif, buka kembali Django Admin di browser dan navigasikan ke model `Blog`. Kamu akan melihat tampilan yang telah diperbarui sesuai dengan pengaturan yang telah ditentukan. Dengan tampilan yang dipersonalisasi, pengelolaan data menjadi jauh lebih efisien dan mudah bagi administrator.

Memanfaatkan fitur-fitur seperti `list_display`, `list_filter`, dan `search_fields` adalah kunci untuk menciptakan antarmuka administrasi yang ramah pengguna.

Dengan pendekatan ini, kita dapat memberikan kemudahan kepada pengguna dalam mengelola data dan meningkatkan produktivitas dalam aplikasi. Pada sub-bab selanjutnya, kita akan membahas cara mengubah tampilan Django Admin untuk menyesuaikan kebutuhan aplikasi lebih lanjut.

3.5 Personalisasi Django Admin

Django Admin adalah alat yang sangat berguna untuk mengelola data, tetapi seringkali kita ingin menyesuaikan tampilan antarmukanya agar lebih sesuai dengan identitas aplikasi kita atau untuk meningkatkan pengalaman pengguna. Dalam sub-bab ini, kita

Memahami dan Menggunakan Django Admin

akan membahas bagaimana cara menyesuaikan tampilan Django Admin dengan menambahkan atribut CSS sederhana dan mengubah template admin.

3.5.1 Mengubah Tampilan Django Admin

Salah satu cara termudah untuk menyesuaikan tampilan Django Admin adalah dengan menambahkan CSS kustom. Untuk melakukan ini, kita perlu membuat direktori baru dalam aplikasi kita yang akan menyimpan file CSS. Pertama, buat folder bernama `static` di dalam aplikasi jika belum ada. Di dalam folder ini, buat folder CSS untuk menyimpan file CSS kustom kita.

Misalnya, kita bisa membuat file CSS bernama `admin_custom.css`. Berikut adalah langkah-langkah yang dapat diikuti:

Buat Struktur Folder: Di dalam direktori aplikasi, buat struktur folder berikut:

```
your_app/  
  static/  
    css/  
      admin_custom.css # File CSS kustom
```

Tambahkan Gaya Kustom: Buka file `admin_custom.css` dan tambahkan beberapa gaya kustom. Misalnya, kita bisa mengubah warna latar belakang dan font:

```
/* admin_custom.css */  
body {  
  background-color: #f5f5f5; /* Mengubah  
  warna latar belakang */  
}
```

Memahami dan Menggunakan Django Admin

```
font-family: 'Arial', sans-serif; /*
Mengubah font */
}

.dashboard-header {
    background-color: #007bff; /* Mengubah
warna header */
    color: white; /* Mengubah warna teks header
*/
}
```

Menyertakan CSS Kustom di Django Admin: Selanjutnya, kita perlu memberi tahu Django untuk menyertakan file CSS kustom ini di dalam antarmuka admin. Kita dapat melakukannya dengan menambahkan kode berikut ke dalam file `admin.py`:

```
# admin.py
from django.contrib import admin
from django.utils.html import format_html
from django.conf import settings

class CustomAdminSite(admin.AdminSite):
    site_header = 'My Custom Admin' # Mengubah
    judul halaman admin

    def get_urls(self):
        urls = super().get_urls()
        custom_css_url =
f"{settings.STATIC_URL}css/admin_custom.css" #
URL file CSS
        return [format_html('<link
rel="stylesheet" type="text/css" href="{}">',
custom_css_url)] + urls

admin_site =
CustomAdminSite(name='custom_admin') #
Menggunakan CustomAdminSite
```

Memahami dan Menggunakan Django Admin

Dengan kode di atas, kita membuat kelas baru bernama `CustomAdminSite` yang memungkinkan kita untuk menyesuaikan judul halaman admin dan menyertakan file CSS kustom yang telah kita buat. Kita juga menggunakan `get_urls` untuk menyertakan CSS ke dalam antarmuka admin.

Mendaftarkan Model dengan *CustomAdminSite*: Terakhir, kita perlu memastikan bahwa model kita terdaftar menggunakan `CustomAdminSite` yang telah kita buat. Misalnya, jika kita memiliki model `Blog`, kita dapat mendaftarkannya seperti ini:

```
# admin.py
from .models import Blog # Mengimpor model Blog

admin_site.register(Blog) # Mendaftarkan model
Blog dengan CustomAdminSite
```

Setelah menyimpan perubahan ini, jalankan server Django jika belum berjalan:

```
$ python manage.py runserver # Menjalankan
server Django
```

Kemudian, buka kembali antarmuka Django Admin di browser dan periksa apakah perubahan telah diterapkan. Kamu akan melihat tampilan yang lebih kustom dan sesuai dengan gaya yang diinginkan.

Menyesuaikan tampilan Django Admin dengan CSS kustom adalah cara yang efektif untuk menciptakan antarmuka yang lebih menarik dan sesuai dengan merek aplikasi kamu. Dengan lang-

Memahami dan Menggunakan Django Admin

kah-langkah ini, kamu dapat memberikan sentuhan personal pada admin panel, yang pada akhirnya akan meningkatkan pengalaman pengguna. Pada sub-bab selanjutnya, kita akan membahas cara menambahkan aksi khusus di Django Admin untuk meningkatkan fungsionalitasnya.

3.5.2 Menambahkan Aksi Khusus di Django Admin

Salah satu fitur hebat yang ditawarkan oleh Django Admin adalah kemampuan untuk menambahkan aksi kustom. Aksi ini memungkinkan administrator untuk melakukan operasi tertentu secara massal pada satu atau beberapa entri model. Misalnya, kita dapat membuat aksi untuk menghapus beberapa item sekaligus atau melakukan tindakan khusus yang lebih kompleks sesuai dengan kebutuhan aplikasi.

Untuk menambahkan aksi kustom ke model di Django Admin, kita akan melanjutkan dengan model `Blog` yang telah kita daftarkan sebelumnya. Mari kita buat aksi kustom yang akan menghapus semua entri yang dipilih oleh pengguna. Berikut adalah langkah-langkah yang perlu dilakukan:

Buka File `admin.py`: Pastikan kamu berada di file `admin.py` di mana model `Blog` terdaftar.

Definisikan Fungsi Aksi Kustom: Kita perlu mendefinisikan fungsi yang akan dijalankan saat aksi dipilih. Fungsi ini akan menerima dua parameter: `modeladmin` dan `queryset`. Berikut adalah contoh fungsi untuk menghapus entri yang dipilih:

Memahami dan Menggunakan Django Admin

```
# admin.py
from django.contrib import admin
from django.contrib import messages
from .models import Blog # Mengimpor model Blog

def delete_selected_blogs(modeladmin, request,
queryset):
    # Menghapus entri yang dipilih
    deleted_count, _ = queryset.delete() #
    Menghapus entri
    messages.success(request, f'{deleted_count}
    entri berhasil dihapus.') # Menampilkan pesan
    sukses

delete_selected_blogs.short_description = 'Hapus
    entri yang dipilih' # Deskripsi untuk aksi
```

Dalam kode di atas, kita mendefinisikan fungsi `delete_selected_blogs` yang akan menghapus entri yang dipilih. Fungsi ini juga menggunakan `messages` untuk memberikan umpan balik kepada pengguna setelah aksi dijalankan.

Menambahkan Aksi Kustom ke ModelAdmin: Selanjutnya, kita perlu menambahkan fungsi aksi kustom ini ke dalam kelas `BlogAdmin` yang telah kita buat sebelumnya. Berikut adalah pembaruan yang perlu dilakukan:

```
# admin.py
class BlogAdmin(admin.ModelAdmin):
    list_display = ('title', 'author',
'created_at') # Menampilkan kolom tertentu
    list_filter = ('author', 'created_at') #
    Menambahkan filter
```

Memahami dan Menggunakan Django Admin

```
search_fields = ('title', 'content') #  
Memungkinkan pencarian  
actions = [delete_selected_blogs] #  
Menambahkan aksi kustom ke dalam admin  
  
admin.site.register(Blog, BlogAdmin) #  
Mendaftarkan model Blog dengan personalisasi
```

Dengan menambahkan `actions = [delete_selected_blogs]`, kita memberi tahu Django Admin untuk menyertakan aksi kustom yang baru saja kita buat.

Jalankan Server dan Uji Aksi: Setelah melakukan perubahan ini, simpan file `admin.py` dan jalankan server Django jika belum berjalan:

```
$ python manage.py runserver # Menjalankan  
server Django
```

Sekarang, buka antarmuka Django Admin di browser dan navigasikan ke model **Blog**. Pilih beberapa entri dengan mencentang kotak di sebelah kiri masing-masing entri, lalu pilih aksi "Hapus entri yang dipilih" dari dropdown aksi di atas daftar. Setelah mengklik "Go," semua entri yang dipilih akan dihapus, dan kamu akan melihat pesan sukses di bagian atas halaman.

Menambahkan aksi kustom di Django Admin memungkinkan administrator untuk melakukan tugas tertentu secara efisien dan meningkatkan kontrol atas data. Dengan aksi seperti "Hapus entri yang dipilih," pengguna dapat mengelola data dengan lebih mudah dan cepat, yang merupakan keuntungan besar dalam pengelolaan aplikasi.

3.6 Fitur-Fitur Tambahan Django Admin

Dalam bagian ini, kita akan membahas fitur tambahan yang dapat meningkatkan fungsionalitas Django Admin, seperti menggunakan inline forms untuk mengedit model terkait langsung di halaman admin.

3.6.1 Inline Forms di Django Admin

Salah satu fitur yang sangat berguna dalam Django Admin adalah kemampuan untuk menggunakan "Inline Forms." Fitur ini memungkinkan pengguna untuk menampilkan dan mengedit model terkait langsung dari halaman admin tanpa perlu membuka halaman terpisah. Ini sangat berguna ketika kita memiliki hubungan antar model, seperti model `Blog` yang mungkin memiliki beberapa komentar yang terkait.

Untuk mendemonstrasikan penggunaan inline forms, mari kita asumsikan kita telah memiliki dua model: `Blog` dan `Comment`. Model `Comment` akan memiliki relasi satu-ke-banyak dengan model `Blog`, di mana satu blog dapat memiliki banyak komentar. Berikut adalah langkah-langkah untuk mengimplementasikan fitur ini.

Definisikan Model `Comment`: Pertama, kita perlu memastikan bahwa kita memiliki model `Comment` yang terkait dengan mo-

Memahami dan Menggunakan Django Admin

del Blog. Berikut adalah contoh bagaimana model ini dapat didefinisikan:

```
# models.py
from django.db import models

class Blog(models.Model):
    title = models.CharField(max_length=200)
    content = models.TextField()
    created_at =
models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.title

class Comment(models.Model):
    blog = models.ForeignKey(Blog,
on_delete=models.CASCADE,
related_name='comments') # Hubungan dengan Blog
    author = models.CharField(max_length=100)
    text = models.TextField()
    created_at =
models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return f'Comment by {self.author} on
{self.blog}'
```

Dalam contoh ini, model `Comment` memiliki field `blog` yang merupakan foreign key yang mengarah ke model `Blog`, sehingga setiap komentar terkait dengan satu blog.

Menambahkan Inline Form di Admin: Selanjutnya, kita perlu menambahkan inline form untuk model `Comment` dalam admin

Memahami dan Menggunakan Django Admin

panel model Blog. Buka file `admin.py` dan tambahkan kode berikut:

```
# admin.py
from django.contrib import admin
from .models import Blog, Comment # Mengimpor
model Blog dan Comment

class CommentInline(admin.TabularInline): #
    Menggunakan TabularInline untuk tampilan tabel
    model = Comment # Model yang akan digunakan
    extra = 1 # Jumlah form kosong yang akan
    ditampilkan

class BlogAdmin(admin.ModelAdmin):
    list_display = ('title', 'created_at') #
    Menampilkan kolom tertentu
    inlines = [CommentInline] # Menambahkan
    inline form untuk komentar

admin.site.register(Blog, BlogAdmin) #
Mendaftarkan model Blog dengan personalisasi
```

Dalam kode di atas, kita membuat kelas `CommentInline` yang merupakan subclass dari `admin.TabularInline`. Kelas ini akan menampilkan form untuk model `Comment` di dalam halaman admin untuk model `Blog`. Kita juga menambahkan parameter `extra` untuk menentukan jumlah form kosong yang ingin ditampilkan.

Jalankan Server dan Uji Inline Forms: Setelah menyimpan perubahan pada file `admin.py`, jalankan server Django jika belum berjalan:

Memahami dan Menggunakan Django Admin

```
$ python manage.py runserver # Menjalankan  
server Django
```

Selanjutnya, buka antarmuka Django Admin di browser dan navigasikan ke model `Blog`. Saat membuat atau mengedit entri blog, kamu sekarang akan melihat form untuk menambahkan komentar di bawah form utama blog. Ini memungkinkan pengguna untuk menambahkan komentar baru langsung tanpa harus meninggalkan halaman.

Penggunaan inline forms sangat menguntungkan ketika kita memiliki model dengan hubungan yang kompleks, karena hal ini membuat pengelolaan data menjadi lebih efisien dan terorganisir. Dengan cara ini, pengguna dapat dengan mudah melihat dan mengedit data terkait dalam satu tampilan, yang meningkatkan pengalaman pengguna secara keseluruhan.

3.6.2 Membuat Filter Kustom

Django Admin tidak hanya menyediakan kemampuan untuk menampilkan dan mengelola data, tetapi juga memungkinkan kita untuk membuat filter kustom. Filter kustom ini memungkinkan administrator untuk dengan mudah menyaring data berdasarkan kriteria tertentu yang relevan dengan aplikasi. Ini sangat berguna ketika kita memiliki banyak entri dan ingin memfokuskan perhatian pada subset data tertentu.

Mari kita lihat bagaimana cara membuat filter kustom untuk model `Blog` dan `Comment`. Misalnya, kita ingin membuat filter kustom yang memungkinkan pengguna untuk menyaring komentar berdasarkan penulisnya.

Memahami dan Menggunakan Django Admin

Membuat Kelas Filter Kustom: Untuk membuat filter kustom, kita perlu mendefinisikan sebuah kelas yang mewarisi dari `admin.SimpleListFilter`. Berikut adalah contoh bagaimana cara mendefinisikan filter kustom ini:

```
# admin.py
from django.contrib import admin
from .models import Blog, Comment # Mengimpor
model Blog dan Comment

class
CommentAuthorFilter(admin.SimpleListFilter):
    title = 'Penulis Komentar' # Judul filter
yang akan ditampilkan di admin
    parameter_name = 'author' # Nama parameter
yang akan digunakan dalam URL

    def lookups(self, request, model_admin):
        authors = set([c.author for c in
Comment.objects.all()]) # Mengambil daftar unik
penulis komentar
        return [(author, author) for author in
authors] # Mengembalikan daftar tuple (nilai,
label)

    def queryset(self, request, queryset):
        if self.value(): # Jika ada nilai
filter yang dipilih
            return
queryset.filter(author=self.value()) #
Mengembalikan queryset yang difilter
            return queryset # Mengembalikan
queryset tanpa filter
```

Memahami dan Menggunakan Django Admin

Dalam kode di atas, kelas `CommentAuthorFilter` mendefinisikan filter untuk menyaring komentar berdasarkan penulis. Metode `lookups` digunakan untuk menghasilkan daftar penulis unik dari semua komentar yang ada, sedangkan metode `queryset` akan mengembalikan queryset yang difilter berdasarkan penulis yang dipilih.

Menambahkan Filter Kustom ke Admin: Selanjutnya, kita perlu menambahkan filter kustom ini ke dalam kelas `BlogAdmin`. Berikut adalah pembaruan yang perlu dilakukan di file `admin.py`:

```
# admin.py
class BlogAdmin(admin.ModelAdmin):
    list_display = ('title', 'created_at') #
    Menampilkan kolom tertentu
    list_filter = (CommentAuthorFilter,) #
    Menambahkan filter kustom

admin.site.register(Blog, BlogAdmin) #
Mendaftarkan model Blog dengan personalisasi
```

Dengan menambahkan `list_filter = (CommentAuthorFilter,)`, kita memberi tahu Django Admin untuk menyertakan filter kustom yang baru saja kita buat di panel admin model `Blog`.

Jalankan Server dan Uji Filter Kustom: Setelah menyimpan perubahan di file `admin.py`, jalankan server Django jika belum berjalan:

Memahami dan Menggunakan Django Admin

```
$ python manage.py runserver # Menjalankan  
server Django
```

Sekarang, buka antarmuka Django Admin di browser dan navigasikan ke model `Comment`. Di sisi kanan, kamu akan melihat filter baru bernama "Penulis Komentar." Ketika pengguna memilih penulis dari daftar dan menerapkan filter, Django Admin akan menyaring komentar untuk hanya menampilkan komentar yang ditulis oleh penulis tersebut.

Dengan menambahkan filter kustom di Django Admin, kita meningkatkan fungsionalitas dan kemudahan penggunaan antarmuka admin. Pengguna dapat dengan cepat menemukan data yang relevan berdasarkan kriteria tertentu, yang sangat penting ketika berhadapan dengan jumlah data yang besar.

3.6.3 Export Data dari Django Admin

Salah satu fitur yang sangat berguna dalam Django Admin adalah kemampuan untuk mengekspor data langsung dari halaman admin, seperti ke dalam format CSV. Ini memungkinkan administrator untuk mengambil data dari aplikasi dan menggunakannya di luar sistem, seperti untuk analisis lebih lanjut, laporan, atau untuk keperluan pengolahan data lainnya.

Mari kita lihat bagaimana cara menambahkan fitur ekspor data ke dalam model `Comment` yang telah kita bahas sebelumnya.

Membuat Fungsi Ekspor CSV: Pertama, kita perlu membuat fungsi yang akan melakukan ekspor data ke dalam format CSV. Fungsi ini akan mengambil queryset dari model dan mengonver-

Memahami dan Menggunakan Django Admin

sinya menjadi format CSV. Berikut adalah contoh bagaimana fungsi ini dapat didefinisikan:

```
# admin.py
import csv
from django.http import HttpResponse
from django.contrib import admin
from .models import Comment # Mengimpor model
Comment

class CommentAdmin(admin.ModelAdmin):
    list_display = ('author', 'text',
                    'created_at') # Menampilkan kolom tertentu

    def export_as_csv(self, request, queryset):
# Fungsi untuk mengekspor data
        response =
HttpResponse(content_type='text/csv') #
Mengatur tipe konten sebagai CSV
        response['Content-Disposition'] =
'attachment; filename="comments.csv"' #
Menentukan nama file

        writer = csv.writer(response) # Membuat
objek penulis CSV
        writer.writerow(['Author', 'Text',
                        'Created At']) # Menulis header CSV

        for comment in queryset: # Mengiterasi
setiap komentar dalam queryset
            writer.writerow([comment.author,
                            comment.text, comment.created_at]) # Menulis
data komentar

        return response # Mengembalikan
response dengan data CSV
```


Memahami dan Menggunakan Django Admin

Dalam kode di atas, kita mendefinisikan fungsi `export_as_csv` yang akan mengekspor data komentar ke dalam format CSV. Fungsi ini mengatur jenis konten ke `text/csv`, menulis header CSV, dan kemudian menulis setiap komentar ke dalam file CSV.

Menambahkan Tindakan Kustom di Admin: Selanjutnya, kita perlu menambahkan tindakan kustom ini ke dalam antarmuka admin untuk model `Comment`. Kita bisa melakukannya dengan menambahkan properti `actions` dalam kelas admin. Berikut adalah pembaruan yang perlu dilakukan:

```
# admin.py
class CommentAdmin(admin.ModelAdmin):
    list_display = ('author', 'text',
                    'created_at') # Menampilkan kolom tertentu
    actions = ['export_as_csv'] # Menambahkan
    tindakan kustom untuk ekspor CSV

admin.site.register(Comment, CommentAdmin) #
Mendaftarkan model Comment dengan personalisasi
```

Dengan menambahkan `actions = ['export_as_csv']`, kita memberi tahu Django Admin untuk menyertakan opsi ekspor CSV dalam daftar tindakan yang tersedia di antarmuka admin.

Jalankan Server dan Uji Fitur Ekspor: Setelah menyimpan perubahan di file `admin.py`, jalankan server Django jika belum berjalan:

```
$ python manage.py runserver # Menjalankan
server Django
```

Memahami dan Menggunakan Django Admin

Sekarang, buka antarmuka Django Admin di browser dan navigasikan ke model `Comment`. Saat kamu memilih beberapa komentar dan memilih tindakan "Export as CSV" dari dropdown aksi, file CSV akan diunduh ke komputer dengan nama `comment-s.csv`. File ini akan berisi semua komentar yang dipilih dengan kolom `Author`, `Text`, dan `Created At`.

Dengan menambahkan fitur ekspor data ke dalam Django Admin, kita memberikan fleksibilitas kepada pengguna untuk mengelola dan menggunakan data mereka sesuai kebutuhan. Ini sangat berguna dalam situasi di mana data perlu dibagikan atau dianalisis lebih lanjut di luar platform aplikasi.

3.7 Keamanan Dalam Django Admin

Keamanan adalah aspek krusial dalam pengelolaan aplikasi web, terutama saat menggunakan antarmuka admin. Django Admin memberikan alat yang kuat untuk mengelola akses pengguna dan izin, yang sangat penting untuk menjaga integritas dan keamanan data. Dalam bagian ini, kita akan membahas bagaimana cara mengelola izin pengguna di Django Admin, termasuk membatasi akses berdasarkan kelompok pengguna (`user groups`) dan izin (`permissions`) yang dapat diatur untuk memastikan bahwa hanya pengguna yang berwenang yang dapat mengakses data sensitif atau melakukan tindakan tertentu.

Memahami dan Menggunakan Django Admin

3.7.1 Mengelola Izin Pengguna (User Permissions)

Mengelola izin pengguna di Django Admin melibatkan penggunaan sistem izin yang disediakan oleh Django, yang memungkinkan kita untuk mengontrol siapa yang dapat melakukan tindakan tertentu pada model dan data. Django secara otomatis membuat izin untuk setiap model yang kita daftarkan di admin, yang meliputi izin untuk membaca, menambahkan, mengedit, dan menghapus data.

Pertama, kita perlu memahami bahwa setiap pengguna di Django dapat ditempatkan dalam kelompok yang disebut "user groups".

Kelompok ini dapat memiliki izin yang sama, sehingga memudahkan pengelolaan akses untuk sekelompok pengguna yang memiliki peran serupa. Misalnya, kita bisa memiliki kelompok untuk "Editor", yang memiliki izin untuk mengedit konten, dan kelompok "Viewer", yang hanya memiliki izin untuk melihat konten.

Untuk mengelola izin pengguna, kita dapat melakukannya melalui antarmuka Django Admin dengan langkah-langkah berikut:

Setelah login ke Django Admin, navigasikan ke bagian "Users" atau "Groups". Di sini, kita dapat menambah atau mengedit pengguna dan kelompok. Saat menambahkan atau mengedit pengguna, kita akan melihat opsi untuk menetapkan izin secara langsung atau melalui kelompok. Ketika mengedit kelompok, ki-

Memahami dan Menggunakan Django Admin

ta dapat memilih izin yang ingin diberikan kepada kelompok tersebut, yang kemudian akan berlaku untuk semua pengguna dalam kelompok tersebut.

Sebagai contoh, jika kita memiliki model `Article`, Django akan otomatis membuat izin seperti `add_article`, `change_article`, dan `delete_article`. Kita dapat memilih izin-izin ini untuk diberikan kepada kelompok pengguna tertentu. Jika kita ingin membatasi akses seorang pengguna untuk hanya membaca artikel tanpa bisa mengedit atau menghapusnya, kita dapat mengatur izin dengan tepat saat menambahkan pengguna ke dalam kelompok.

Setelah menetapkan izin, pengguna hanya akan dapat melihat dan mengakses data sesuai dengan izin yang diberikan. Ini membantu memastikan bahwa data yang sensitif dan penting tidak diubah atau dihapus oleh pengguna yang tidak berwenang.

Dengan menggunakan sistem izin ini, kita tidak hanya meningkatkan keamanan aplikasi tetapi juga menjaga agar data tetap aman dan teratur. Sistem ini sangat fleksibel dan memungkinkan penyesuaian yang sesuai dengan kebutuhan bisnis spesifik. Di sub-bab berikutnya, kita akan membahas langkah-langkah lebih lanjut untuk membatasi akses ke Django Admin, termasuk cara mengatur URL kustom untuk halaman admin demi meningkatkan keamanan.

Memahami dan Menggunakan Django Admin

3.7.2 Membatasi Akses ke Django Admin

Membatasi akses ke Django Admin adalah langkah penting dalam menjaga keamanan aplikasi web. Dengan melakukan hal ini, kita dapat mencegah akses yang tidak sah dan melindungi data sensitif dari pengguna yang tidak berwenang. Dalam sub-bab ini, kita akan menjelaskan beberapa langkah yang dapat diambil untuk meningkatkan keamanan Django Admin, termasuk penggunaan URL kustom dan pengaturan izin pengguna.

Langkah pertama dalam membatasi akses adalah mengubah URL default untuk halaman admin. Secara default, URL admin di Django dapat diakses melalui `/admin/`. Ini dapat menjadi target yang mudah bagi penyerang, yang sering kali mencoba untuk mengakses halaman admin. Dengan mengganti URL ini ke sesuatu yang lebih tidak terduga, kita dapat menambah lapisan perlindungan ekstra. Untuk melakukannya, kita perlu mengedit file `urls.py` di proyek Django kita.

Misalnya, kita bisa mengubah URL admin menjadi `/my-secret-admin/` dengan cara berikut:

```
# urls.py
from django.contrib import admin
from django.urls import path

urlpatterns = [
    path('my-secret-admin/', admin.site.urls),
    # Mengubah URL admin
]
```

Memahami dan Menggunakan Django Admin

Dengan menambahkan perubahan ini, akses ke admin sekarang hanya dapat dilakukan melalui URL baru yang telah kita tetapkan. Ini membuatnya lebih sulit untuk ditemukan oleh penyerang yang berusaha mengakses halaman admin.

Langkah berikutnya adalah memastikan bahwa hanya pengguna yang memiliki izin yang tepat yang dapat mengakses Django Admin. Seperti yang telah dibahas sebelumnya, kita dapat mengelola izin pengguna dan kelompok untuk membatasi akses. Namun, kita juga dapat mempertimbangkan untuk menggunakan otentikasi tambahan, seperti otentikasi dua faktor (2FA), untuk meningkatkan keamanan.

Selain itu, kita bisa membatasi akses ke halaman admin berdasarkan alamat IP. Ini bisa dilakukan dengan menggunakan middleware yang memeriksa alamat IP pengguna sebelum mengizinkan akses ke Django Admin. Kita dapat menulis middleware sederhana seperti berikut:

```
# middleware.py
from django.http import HttpResponseRedirect
from django.urls import reverse

class AdminIPRestrictionMiddleware:
    ALLOWED_IPS = ['192.168.1.1'] # Ganti
    dengan alamat IP yang diizinkan

    def __init__(self, get_response):
        self.get_response = get_response

    def __call__(self, request):
        if
request.path.startswith(reverse('admin:index'))
```

Memahami dan Menggunakan Django Admin

```
and request.META['REMOTE_ADDR'] not in
self.ALLOWED_IPS:
    return HttpResponseForbidden("You
are not allowed to access this page.")
    return self.get_response(request)
```

Dalam contoh di atas, kita membuat middleware yang akan memeriksa apakah alamat IP pengguna ada dalam daftar yang diizinkan sebelum mengizinkan akses ke halaman admin. Jika alamat IP pengguna tidak ada dalam daftar, mereka akan menerima respons "Forbidden".

Dengan menerapkan langkah-langkah ini, kita dapat secara signifikan meningkatkan keamanan Django Admin dan melindungi aplikasi kita dari akses yang tidak sah. Keamanan adalah proses yang berkelanjutan, dan penting untuk selalu memantau dan memperbarui pengaturan keamanan sesuai dengan kebutuhan aplikasi. Di sub-bab berikutnya, kita akan merangkum poin-poin penting yang telah dibahas dalam bab ini dan membahas pentingnya memahami dan memaksimalkan penggunaan Django Admin.

3.8 Mengatasi Masalah Umum Di Django Admin

Ketika bekerja dengan Django Admin, kita mungkin menemui beberapa masalah umum yang dapat menghambat akses atau penggunaan antarmuka administrasi. Dalam sub-bab ini, kita akan mengidentifikasi masalah-masalah yang sering terjadi, seperti "Login tidak valid" dan "Superuser tidak bisa mengakses

Memahami dan Menggunakan Django Admin

halaman admin," serta memberikan penjelasan dan solusi untuk mengatasinya.

3.8.1 Masalah Umum saat Mengakses Django Admin

Salah satu masalah paling umum yang dihadapi pengguna adalah kesulitan dalam login ke halaman admin. Pesan kesalahan "Login tidak valid" sering muncul ketika pengguna mencoba untuk masuk tetapi gagal. Masalah ini biasanya terjadi karena beberapa alasan:

1. **Kesalahan Kredensial:** Pengguna mungkin memasukkan username atau password yang salah. Penting untuk memastikan bahwa kredensial yang dimasukkan benar dan sesuai dengan yang telah ditetapkan saat membuat superuser. Jika ada keraguan, kita dapat mencoba mengatur ulang password menggunakan perintah berikut di terminal:

```
$ python manage.py changepassword [username] #  
Mengganti password untuk user tertentu
```

Dengan mengganti [username] dengan nama pengguna yang ingin diubah, kita akan diminta untuk memasukkan password baru.

2. **Superuser Tidak Terdaftar:** Jika pengguna mencoba login menggunakan akun yang bukan superuser, mereka mungkin tidak memiliki akses ke halaman admin. Pastikan bahwa superuser telah dibuat dengan benar. Jika superuser tidak ada, kita perlu membuatnya menggunakan perintah:

Memahami dan Menggunakan Django Admin

```
$ python manage.py createsuperuser # Membuat  
superuser baru
```

Saat membuat superuser, pastikan untuk mengisi semua informasi yang diminta, termasuk username, email, dan password.

3. **Isu dengan Database:** Terkadang, masalah ini dapat disebabkan oleh kesalahan pada database. Jika ada perubahan pada model yang tidak diterapkan dengan migrasi, kita perlu menjalankan perintah migrasi untuk memastikan semua perubahan diterapkan:

```
$ python manage.py migrate # Menerapkan semua  
migrasi
```

Jika setelah langkah-langkah di atas masalah masih berlanjut, kita juga perlu memeriksa file log aplikasi untuk menemukan petunjuk lebih lanjut mengenai kesalahan yang terjadi.

Masalah lain yang sering muncul adalah ketika superuser tidak dapat mengakses halaman admin sama sekali. Ini bisa disebabkan oleh beberapa faktor, termasuk:

1. **Akses yang Dibatasi:** Jika kita telah mengatur batasan akses menggunakan middleware atau pengaturan lain, pastikan bahwa superuser memiliki izin yang diperlukan untuk mengakses halaman admin. Kita dapat memeriksa pengaturan izin di Django Admin dan memastikan bahwa superuser termasuk dalam kelompok yang memiliki izin penuh.
2. **Kesalahan Konfigurasi URL:** Jika URL admin telah diubah dan tidak dikonfigurasi dengan benar, superuser

Memahami dan Menggunakan Django Admin

mungkin tidak dapat mengaksesnya. Pastikan bahwa URL yang digunakan sesuai dengan pengaturan yang ada di `urls.py`.

3. ***Masalah dengan Dependencies***: Terkadang, masalah ini dapat disebabkan oleh konflik dalam dependencies atau paket yang digunakan dalam proyek Django. Pastikan semua paket terinstal dan diperbarui dengan benar.

Dengan memahami masalah umum ini dan langkah-langkah untuk mengatasinya, kita dapat lebih cepat menemukan solusi saat mengalami kesulitan dalam mengakses Django Admin. Di sub-bab selanjutnya, kita akan membahas lebih lanjut tentang debugging dan solusi untuk masalah-masalah yang lebih kompleks yang mungkin muncul selama penggunaan Django Admin.

3.8.2 Debugging dan Solusi

Setelah mengidentifikasi masalah umum yang dapat menghambat akses dan penggunaan Django Admin, langkah berikutnya adalah melakukan debugging untuk menemukan akar permasalahan dan menerapkan solusi yang tepat. Dalam sub-bab ini, kita akan membahas beberapa pendekatan yang dapat digunakan untuk mengatasi masalah seperti kesalahan pada model, kesalahan izin, dan masalah lainnya yang sering muncul di Django Admin.

Salah satu sumber masalah yang sering terjadi adalah kesalahan pada model. Jika model tidak didefinisikan dengan benar, atau jika ada perubahan yang dilakukan pada model tanpa menerapkan migrasi, ini bisa menyebabkan Django Admin tidak dapat berfungsi dengan baik. Untuk mengatasi hal ini, kita perlu memasti-

Memahami dan Menggunakan Django Admin

kan bahwa semua perubahan pada model sudah diterapkan. Kita bisa melakukannya dengan menjalankan perintah berikut:

```
$ python manage.py makemigrations # Membuat file migrasi  
untuk perubahan model  
$ python manage.py migrate # Menerapkan semua migrasi yang  
tertunda
```

Perintah pertama akan membuat file migrasi yang mencerminkan perubahan yang telah kita buat di model, sedangkan perintah kedua akan menerapkannya ke database. Jika ada masalah yang terdeteksi selama migrasi, Django akan memberikan pesan kesalahan yang membantu kita memahami apa yang salah.

Selanjutnya, kita juga perlu memperhatikan pengaturan izin pengguna. Kadang-kadang, pengguna mungkin tidak memiliki izin yang diperlukan untuk mengakses fitur tertentu di Django Admin. Untuk memeriksa dan mengelola izin, kita bisa masuk ke Django Admin menggunakan akun superuser dan pergi ke bagian "Users" dan "Groups". Pastikan pengguna memiliki izin yang sesuai dengan tugas mereka. Jika tidak, kita bisa menambahkan izin yang diperlukan dengan mudah.

Jika pengguna menghadapi kesalahan saat mengakses halaman tertentu, seperti "403 Forbidden," ini biasanya disebabkan oleh masalah izin. Pastikan bahwa pengguna tersebut adalah bagian dari kelompok yang memiliki akses ke halaman admin. Kita juga dapat menggunakan `User` dan `Group` untuk menetapkan izin secara lebih spesifik.

Memahami dan Menggunakan Django Admin

Selain itu, jika terdapat kesalahan lain yang muncul saat menjalankan server, kita dapat memeriksa log kesalahan untuk mendapatkan informasi lebih lanjut. Mengaktifkan logging di Django dapat membantu dalam hal ini. Dalam file `settings.py`, kita dapat menambahkan pengaturan untuk logging:

```
# settings.py
LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'handlers': {
        'file': {
            'level': 'DEBUG',
            'class': 'logging.FileHandler',
            'filename': 'django_debug.log', #
File log
        },
    },
    'loggers': {
        'django': {
            'handlers': ['file'],
            'level': 'DEBUG',
            'propagate': True,
        },
    },
}
```

Dengan pengaturan ini, semua log Django akan dicatat dalam file `django_debug.log`. Kita dapat memeriksa file tersebut untuk menemukan petunjuk lebih lanjut tentang kesalahan yang terjadi.

Memahami dan Menggunakan Django Admin

Dengan mengikuti langkah-langkah debugging ini, kita dapat lebih mudah menemukan dan memperbaiki masalah yang muncul saat menggunakan Django Admin. Mengetahui cara mengatasi kesalahan ini tidak hanya meningkatkan pengalaman pengguna, tetapi juga memastikan bahwa aplikasi kita berjalan dengan lancar.

Memahami dan Menggunakan Django Admin

Kesimpulan Bab

Dalam bab ini, kita telah mengeksplorasi berbagai aspek penting dari Django Admin, yang merupakan alat yang sangat kuat untuk manajemen konten dalam aplikasi Django. Kita memulai dengan pengantar tentang peran dan manfaat Django Admin, kemudian melanjutkan dengan cara membuat superuser dan memanfaatkan fitur-fitur utama yang ada.

Kita membahas bagaimana mendaftarkan model ke dalam Django Admin, serta cara personalisasi tampilan dan fungsionalitasnya agar lebih sesuai dengan kebutuhan pengguna. Fitur seperti `list_display`, `list_filter`, dan `search_fields` memberikan kontrol yang lebih besar dalam menampilkan data, sementara kemampuan untuk menambahkan aksi kustom memungkinkan kita untuk memperluas fungsionalitas sesuai dengan kebutuhan bisnis.

Di sisi lain, kita juga menyentuh aspek keamanan yang krusial, seperti pengelolaan izin pengguna dan langkah-langkah untuk membatasi akses ke halaman admin. Ini penting untuk memastikan bahwa hanya pengguna yang berwenang yang dapat mengakses dan mengelola data sensitif.

Tidak kalah pentingnya, kita telah mengidentifikasi dan memberikan solusi untuk masalah umum yang mungkin dihadapi pengguna saat berinteraksi dengan Django Admin. Dengan memahami cara melakukan debugging dan mengelola kesalahan, kita dapat memastikan aplikasi berjalan dengan baik dan pengalaman pengguna tetap positif.

Memahami dan Menggunakan Django Admin

Secara keseluruhan, bab ini menekankan pentingnya pemahaman yang mendalam tentang Django Admin untuk memaksimalkan potensi aplikasi Django kita. Dengan menggunakan fitur-fitur yang disediakan dan menerapkan praktik terbaik dalam pengelolaan izin serta keamanan, kita dapat menciptakan sistem manajemen yang efektif dan efisien, mendukung kebutuhan bisnis dan meningkatkan produktivitas.

BAB 4 - Memulai Awal proyek

Dalam bab ini kita akan menyiapkan proyek Django untuk platform bot kita. Proses ini melibatkan pembuatan proyek baru dan mengonfigurasi struktur folder serta file yang diperlukan untuk memulai pengembangan.

4.1 Pembuatan Proyek Django

Kita akan menyiapkan proyek Django untuk platform bot kita. Proses ini melibatkan pembuatan proyek baru dan mengonfigurasi struktur folder serta file yang diperlukan untuk memulai pengembangan.

Untuk memulai, kita perlu membuat proyek Django baru. Pastikan Anda telah menginstal Django di lingkungan pengembangan Anda.

Setelah Django terinstal, Anda bisa membuat proyek baru dengan perintah berikut:

```
$ django-admin startproject platform_bot
```

Perintah ini akan membuat folder bernama `platform_bot` yang berisi file dan folder dasar untuk proyek Django Anda.

Memulai Awal proyek

Struktur folder ini akan mencakup file `manage.py` dan folder `platform_bot` yang berisi file konfigurasi utama.

Di dalam direktori `platform_bot`, Anda akan menemukan struktur proyek sebagai berikut:

- `manage.py`: Skrip untuk mengelola proyek Django, seperti menjalankan server pengembangan, membuat migrasi, dan lainnya.
- `platform_bot/`: Direktori yang berisi pengaturan utama proyek Django.
 - `__init__.py`: File kosong yang menandakan bahwa direktori ini adalah sebuah package Python.
 - `settings.py`: File yang berisi pengaturan konfigurasi untuk proyek Django.
 - `urls.py`: File yang menentukan pola URL untuk proyek Django.
 - `wsgi.py`: File yang berfungsi sebagai titik masuk untuk WSGI server.

Dengan mengikuti langkah-langkah di atas, kita telah berhasil membuat proyek Django pertama kita. Selanjutnya, kita akan memulai eksplorasi lebih lanjut tentang pengaturan proyek Django dan mulai membangun aplikasi web dengan Django.

4.1.1 Membuat Folder `apps`

Dalam pengembangan proyek Django, sering kali bermanfaat untuk mengorganisasi aplikasi Anda ke dalam folder terpisah yang memudahkan manajemen dan pengembangan. Untuk proyek

Memulai Awal proyek

`platform_bot`, kita akan membuat folder `apps` untuk menampung semua aplikasi yang terkait dengan proyek ini. Ini adalah praktik yang baik untuk menjaga struktur proyek tetap rapi dan terorganisir.

Alasan dan Cara Membuat Folder `apps`

Membuat folder `apps` bertujuan untuk menyederhanakan struktur proyek dan memisahkan aplikasi-aplikasi berbeda dalam satu tempat. Ini membantu dalam mengelola aplikasi dengan lebih mudah, terutama saat proyek berkembang menjadi lebih kompleks.

Langkah pertama adalah membuat folder `apps` di direktori proyek Anda. Untuk melakukannya, navigasikan ke direktori proyek `platform_bot` dan buat folder baru bernama `apps`:

```
$ mkdir apps
```

Setelah folder `apps` dibuat, kita perlu memindahkan aplikasi yang sudah ada dan menambahkan aplikasi baru ke dalam folder ini. Secara default, aplikasi yang akan Anda buat harus ditempatkan dalam folder `apps`.

Struktur Folder di Dalam `apps`

Di dalam folder `apps`, kita akan membuat folder untuk masing-masing aplikasi yang diperlukan dalam proyek ini. Untuk `platform_bot`, kita akan membuat aplikasi berikut:

Memulai Awal proyek

- **authentication:** Untuk sistem registrasi, login, dan logout pengguna.
- **dashboard:** Untuk halaman dashboard tempat pengguna mengelola bot mereka.
- **users:** Untuk halaman user profile dan mengelola profile mereka.
- **webhook:** Untuk menangani webhook yang berinteraksi dengan bot.
- **bots:** Untuk logika dan pengelolaan bot itu sendiri.

Untuk membuat aplikasi-aplikasi ini, gunakan perintah Django berikut di dalam folder `apps`:

```
$ cd apps
$ django-admin startapp authentication
$ django-admin startapp dashboard
$ django-admin startapp users
$ django-admin startapp webhook
$ django-admin startapp bots
```

Setelah menjalankan perintah tersebut, folder `apps` akan memiliki struktur seperti ini:

```
platform_bot/
├── apps/
│   ├── authentication/
│   ├── dashboard/
│   ├── users/
│   ├── webhook/
│   └── bots/
```

Memulai Awal proyek

```
— platform_bot/
  — __init__.py
  — asgi.py
  — settings.py
  — urls.py
  — wsgi.py
— manage.py
— db.sqlite3
```

*Menambahkan Folder apps ke **INSTALLED_APPS***

Setelah membuat aplikasi, Anda perlu memastikan bahwa Django mengetahui aplikasi-aplikasi ini dengan menambahkannya ke daftar `INSTALLED_APPS` di file `settings.py`. Buka `platform_bot/settings.py` dan tambahkan setiap aplikasi ke daftar `INSTALLED_APPS` seperti berikut:

```
# File: platform_bot/settings.py

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'apps.authentication', # Aplikasi
    autentikasi
    'apps.dashboard', # Aplikasi dashboard
    'apps.users', # Aplikasi users
    'apps.webhook', # Aplikasi webhook
    'apps.bots', # Aplikasi bots
]
```

Memulai Awal proyek

Dengan menambahkan aplikasi ke dalam `INSTALLED_APPS`, Django akan mengenali dan memuat aplikasi-aplikasi tersebut saat proyek dijalankan.

Dengan folder `apps` dan aplikasi yang telah dibuat, proyek Anda kini memiliki struktur yang lebih terorganisir, memudahkan pengembangan dan pemeliharaan aplikasi di masa mendatang. Selanjutnya, kita akan menyelami pembuatan dan konfigurasi aplikasi secara lebih detail.

Dengan konfigurasi ini, proyek Django kita siap untuk dikembangkan lebih lanjut. Selanjutnya, kita akan menambahkan aplikasi yang diperlukan dan mengatur struktur folder yang lebih detail.

4.1.2 Memperbaiki `apps.py`

Apa itu `apps.py`?

File `apps.py` dalam proyek Django adalah tempat di mana konfigurasi untuk setiap aplikasi Django didefinisikan. Di dalam file ini, kita biasanya mendefinisikan kelas yang mewarisi dari `AppConfig`, yang digunakan untuk mengatur beberapa opsi dan pengaturan terkait aplikasi tertentu.

Mengapa Mengubah `apps.py`?

Secara default, Django menghasilkan file `apps.py` dengan nama aplikasi yang disederhanakan, tetapi saat menggunakan struktur proyek yang lebih kompleks, seperti folder `apps` untuk memisahkan aplikasi, penting untuk memastikan bahwa path aplika-

Memulai Awal proyek

si didefinisikan dengan benar. Jika path aplikasi tidak diatur dengan benar, Django mungkin tidak dapat menemukan aplikasi yang dimaksud, yang dapat menyebabkan masalah dalam pemuatan dan pengelolaan aplikasi.

Di bawah ini adalah cara memperbarui file `apps.py` untuk setiap aplikasi yang telah kita buat, memastikan bahwa path aplikasi ditentukan dengan benar:

Authentication

```
# apps/authentication/apps.py

from django.apps import AppConfig

class AuthenticationConfig(AppConfig):
    default_auto_field =
'django.db.models.BigAutoField'
    name = 'apps.authentication'
```

Webhook

```
# apps/webhook/apps.py

from django.apps import AppConfig

class WebhookConfig(AppConfig):
    default_auto_field =
'django.db.models.BigAutoField'
    name = 'apps.webhook'
```

Dashboard

Memulai Awal proyek

```
# apps/dashboard/apps.py

from django.apps import AppConfig

class DashboardConfig(AppConfig):
    default_auto_field =
'django.db.models.BigAutoField'
    name = 'apps.dashboard'
```

Users

```
# apps/users/apps.py

from django.apps import AppConfig

class UsersConfig(AppConfig):
    default_auto_field =
'django.db.models.BigAutoField'
    name = 'apps.users'
```

Bots

```
# apps/bots/apps.py

from django.apps import AppConfig

class BotsConfig(AppConfig):
    default_auto_field =
'django.db.models.BigAutoField'
    name = 'apps.bots'
```

Alasan Mengubah apps.py:

1. **Konsistensi Struktur Proyek:** Jika menggunakan struktur proyek yang terpisah di dalam folder `apps`, path aplikasi di `apps.py` harus disesuaikan dengan struktur ini agar

Memulai Awal proyek

Django dapat menemukan dan memuat aplikasi dengan benar.

2. **Pengelolaan Aplikasi:** Menyertakan path yang benar memastikan bahwa semua aplikasi dikenali oleh Django, sehingga memudahkan pengelolaan dan integrasi aplikasi dalam proyek.
3. **Mencegah Kesalahan:** Mengatur path yang benar menghindari kesalahan yang mungkin terjadi jika Django tidak dapat menemukan aplikasi yang ditentukan, yang dapat menyebabkan masalah saat menjalankan server atau migrasi database.

Dengan memperbarui `apps.py` sesuai dengan struktur proyek, kita memastikan bahwa semua aplikasi dapat diakses dan dikelola dengan benar oleh Django. Jika masalah masih terjadi, penting untuk memeriksa konfigurasi dan kode lainnya dalam proyek untuk mengidentifikasi potensi masalah lainnya.

4.1.3 Memahami Struktur Dasar Aplikasi

Dengan folder `apps` dan aplikasi yang sudah dibuat, langkah berikutnya adalah memahami struktur dasar untuk masing-masing aplikasi. Struktur dasar ini meliputi model, view, dan template yang akan digunakan oleh aplikasi. Selain itu, kita juga akan mengonfigurasi routing di `urls.py` untuk menghubungkan URL dengan view yang sesuai.

Model

Model di Django adalah representasi data aplikasi Anda. Model mendefinisikan struktur data yang akan disimpan di database,

Memulai Awal proyek

termasuk jenis data dan hubungan antar data. Setiap model di Django adalah subclass dari `django.db.models.Model` dan biasanya didefinisikan di dalam file `models.py`.

Sebagai contoh, mari kita buat model dasar untuk aplikasi authentication yang menangani informasi pengguna. Buka file `apps/authentication/models.py` dan tambahkan kode berikut:

```
# File: apps/authentication/models.py

from django.db import models
from django.contrib.auth.models import AbstractUser

class CustomUser(AbstractUser):
    # Menambahkan atribut tambahan jika
    # diperlukan
    pass
```

Di sini, `CustomUser` adalah subclass dari `AbstractUser`, yang memungkinkan kita untuk memperluas model pengguna default Django jika diperlukan.

View

View di Django bertanggung jawab untuk menangani logika aplikasi dan menghasilkan respons yang dikirimkan ke pengguna. View diimplementasikan di file `views.py` dan menghubungkan data dari model dengan template.

Memulai Awal proyek

Sebagai contoh, untuk aplikasi dashboard, kita bisa membuat view sederhana untuk menampilkan halaman utama dashboard. Buka file `apps/dashboard/views.py` dan tambahkan kode berikut:

```
# File: apps/dashboard/views.py

from django.shortcuts import render

def dashboard_home(request):
    return render(request,
        'dashboard/home.html')
```

View ini akan merender template `home.html` yang terletak di folder `templates/dashboard/`.

Template

Template di Django adalah file HTML yang menentukan bagaimana data ditampilkan kepada pengguna. Template digunakan untuk merender data yang dikirimkan oleh view.

Untuk aplikasi dashboard, kita akan membuat template dasar di folder `templates/dashboard/`. Buat file baru bernama `home.html` di dalam folder tersebut dan tambahkan kode HTML berikut:

```
<!-- File:
apps/dashboard/templates/dashboard/home.html -->

<!DOCTYPE html>
<html>
<head>
```

Memulai Awal proyek

```
<title>Dashboard Home</title>
</head>
<body>
  <h1>Welcome to Your Dashboard</h1>
</body>
</html>
```

Template ini adalah halaman dasar yang akan ditampilkan ketika pengguna mengakses halaman dashboard.

Konfigurasi Routing di `urls.py`

Routing di Django menghubungkan URL dengan view yang sesuai. Anda perlu mengonfigurasi file `urls.py` di setiap aplikasi untuk mendefinisikan pola URL dan view yang akan menangani URL tersebut.

Mari kita mulai dengan aplikasi `dashboard`. Buka file `apps/dashboard/urls.py` dan tambahkan konfigurasi berikut:

```
# File: apps/dashboard/urls.py

from django.urls import path
from . import views

urlpatterns = [
    path('', views.dashboard, name='dashboard'),
]
```

Konfigurasi ini menghubungkan URL root dari aplikasi `dashboard` dengan view `dashboard`.

Memulai Awal proyek

Selanjutnya, Anda perlu memasukkan URL aplikasi dashboard ke dalam routing utama proyek. Buka file `platform_bot/urls.py` dan tambahkan konfigurasi berikut:

```
# File: platform_bot/urls.py

from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('dashboard/',
include('apps.dashboard.urls')), # Menyertakan
URL aplikasi dashboard
]
```

Konfigurasi ini memastikan bahwa URL yang diawali dengan `/dashboard/` akan dipetakan ke aplikasi dashboard dan view dashboard.

Dengan menyusun model, view, template, dan konfigurasi routing, Anda telah membangun fondasi dasar untuk aplikasi Django Anda. Selanjutnya, kita akan melanjutkan dengan menambahkan fungsionalitas lebih lanjut dan menyiapkan aplikasi lainnya.

4.1.4 Mengelola Static Files

Setelah memahami struktur dasar aplikasi, langkah berikutnya adalah mengelola file static File static seperti CSS, JavaScript, dan gambar, adalah bagian penting dalam pengembangan aplikasi web. Mengelola file-file ini dengan benar akan memastikan

Memulai Awal proyek

tampilan dan fungsionalitas aplikasi Anda sesuai dengan yang diharapkan.

Cara Mengelola File Static seperti CSS dan JavaScript

File static digunakan untuk mengatur tampilan dan interaktivitas aplikasi Anda. Ini termasuk file CSS untuk styling, file JavaScript untuk fungsionalitas interaktif, dan gambar yang digunakan di berbagai tempat dalam aplikasi. Dalam proyek Django, file static biasanya dikelompokkan dalam folder `static`.

Untuk proyek `platform_bot`, kita akan menyimpan file static di dalam folder `apps` untuk menjaga semuanya dalam satu tempat. Berikut adalah langkah-langkah untuk mengelola file static:

Buat Struktur Folder Static

Di dalam folder `apps`, buat subfolder bernama `static`. Di dalam folder `static`, Anda bisa membuat subfolder tambahan untuk CSS, JavaScript, dan gambar:

```
$ mkdir -p apps/static/  
$ mkdir -p apps/static/assets  
$ mkdir -p apps/static/assets/css  
$ mkdir -p apps/static/assets/js  
$ mkdir -p apps/static/assets/img
```

Menambahkan File Static

Memulai Awal proyek

Tempatkan file CSS, JavaScript, dan gambar Anda di subfolder yang sesuai. Sebagai contoh, tambahkan file CSS di folder `css`, file JavaScript di folder `js`, dan gambar di folder `img`.

Misalnya, buat file CSS bernama `styles.css` di folder `css` dengan konten berikut:

```
/* File: apps/static/assets/css/styles.css */

body {
    font-family: Arial, sans-serif;
}

h1 {
    color: #333;
}
```

Dan buat file JavaScript bernama `Script.js` di folder `js` dengan konten berikut:

```
// File: apps/static/assets/js/scripts.js

document.addEventListener('DOMContentLoaded',
function() {
    console.log('JavaScript loaded');
});
```

Untuk gambar, Anda bisa menambahkan file gambar seperti `logo.png` di folder `images`.

Konfigurasi Static di Setting

Agar Django bisa menemukan dan menyajikan file static dengan benar, Anda perlu mengonfigurasi `settings.py`. Buka file

Memulai Awal proyek

platform_bot/settings.py dan pastikan pengaturan STATIC_URL dan STATICFILES_DIRS telah diatur dengan benar:

```
# File: platform_bot/settings.py

STATIC_URL = '/static/'

STATICFILES_DIRS = [
    BASE_DIR / 'apps/static',
]
```

Pengaturan STATIC_URL menentukan URL dasar untuk file static, sedangkan STATICFILES_DIRS memberitahu Django di mana mencari file static di luar folder aplikasi standar.

Penempatan File Static di Dalam Proyek

Untuk memastikan file static dan template terorganisir dengan baik, berikut adalah struktur folder yang diharapkan:

```
platform_bot/
├── apps/
│   ├── authentication/
│   ├── dashboard/
│   ├── users/
│   ├── webhook/
│   ├── bots/
│   ├── static/
│   │   └── assets/
│   │       ├── css/
│   │       ├── js/
│   │       └── img/
```

Memulai Awal proyek

```
platform_bot/  
├── __init__.py  
├── asgi.py  
├── settings.py  
├── urls.py  
├── wsgi.py  
├── manage.py  
└── db.sqlite3
```

Dengan konfigurasi ini, Django akan dapat menyajikan file static dan template dengan benar. Selanjutnya, kita akan melanjutkan dengan menambahkan fungsionalitas tambahan ke aplikasi dan memastikan semua komponen bekerja dengan baik.

4.1.5 Menyusun File Templates di Folder apps

Untuk memastikan bahwa template di proyek Django Anda terstruktur dengan baik dan mudah diakses, Kita perlu mengikuti beberapa langkah penting dalam penyusunan file template. Di proyek kita, folder `templates` akan berada di dalam folder ***apps***. Ini memastikan bahwa template dapat diakses secara langsung sesuai dengan aplikasi yang relevan.

Berikut adalah langkah-langkah untuk mengatur file template dalam proyek Django, khususnya dengan struktur folder `templates` yang berada di dalam folder ***apps***.

Membuat Folder templates

Pertama, buat folder `templates` di dalam folder ***apps***.

```
$ mkdir -p apps/templates/
```


Memulai Awal proyek

```
$ mkdir -p apps/templates/home
$ mkdir -p apps/templates/account
$ mkdir -p apps/templates/dashboard
$ mkdir -p apps/templates/users
$ mkdir -p apps/templates/bots
```

Struktur folder proyek Anda akan terlihat seperti ini:

```
platform_bot/
├── apps/
│   ├── authentication/
│   ├── dashboard/
│   ├── users/
│   ├── webhook/
│   ├── bot/
│   ├── static/
│   └── templates/
│       ├── assets/
│       ├── home/
│       ├── account/
│       ├── dashboard/
│       ├── users/
│       └── bots/
├── platform_bot/
│   ├── __init__.py
│   ├── asgi.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
├── manage.py
└── db.sqlite3
```

Memulai Awal proyek

Di sini, folder `templates` berada di dalam folder ***apps***, dan di dalamnya, Anda memiliki sub-folder `authentication` yang berisi file template seperti `home.html`. Folder `templates` juga bisa berisi file template global seperti `base.html` yang digunakan di seluruh aplikasi.

Konfigurasi Template di `settings.py`

Untuk memastikan Django dapat menemukan file template Anda, Anda perlu mengonfigurasi pengaturan `TEMPLATES` di file `settings.py`. Pastikan Anda menambahkan konfigurasi yang mengarah ke folder `templates` di dalam aplikasi.

```
# File: platform_bot/settings.py

import os

# Konfigurasi template
TEMPLATES = [
    {
        'BACKEND':
'django.template.backends.django.DjangoTemplates',
        'DIRS': [ BASE_DIR / 'apps/templates'],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
'django.template.context_processors.debug',
'django.template.context_processors.request',
'django.contrib.auth.context_processors.auth',
```

Memulai Awal proyek

```
'django.contrib.messages.context_processors.messages',

'platform_bot.context_processors.cfg_assets_root', # Menambahkan context processor
],
},
],
```

Dengan konfigurasi ini, Django akan mencari template di folder `apps/templates` dan juga di sub-folder aplikasi, seperti `apps/authentication/templates`.

Dengan mengikuti langkah-langkah ini, Anda akan memiliki struktur folder `templates` yang terorganisir dengan baik, memastikan bahwa semua file template mudah diakses dan digunakan dalam proyek Django Anda.

4.1.6 Membuat Context Processor

Apa itu Context Processor?

Context processor di Django adalah fitur yang memungkinkan kita untuk menambahkan data global yang dapat diakses di semua template tanpa harus mengirimkannya secara eksplisit melalui view. Context processor mengizinkan kita untuk mendefinisikan data yang akan tersedia di setiap template, sehingga memudahkan pengelolaan data yang konsisten di seluruh aplikasi web.

Fungsi Context Processor

Memulai Awal proyek

Fungsi dari context processor adalah menyediakan data tambahan yang diperlukan oleh template di seluruh aplikasi secara otomatis. Dengan menggunakan context processor, kita tidak perlu menyertakan data tersebut di setiap view yang mengrender template, yang mengurangi duplikasi kode dan meningkatkan efisiensi pengelolaan data global.

Membuat File Context Processor

Pertama, buat file baru bernama `context_processors.py` di dalam folder `apps/`. File ini akan berisi fungsi context processor yang mengambil data dari pengaturan dan menyediakan-nya untuk template.

Gunakan editor teks untuk membuat file:

```
$ vim context_processors.py
```

Isi file `context_processors.py` dengan kode berikut:

```
# File: apps/context_processors.py

from django.conf import settings

def cfg_assets_root(request):
    # Mengambil nilai ASSETS_ROOT dari
    settings.py dan mengirimkannya ke template
    return {'ASSETS_ROOT': settings.ASSETS_ROOT}
```

Kode di atas mendefinisikan fungsi `cfg_assets_root`, yang mengambil nilai `ASSETS_ROOT` dari file `settings.py` dan mengembalikannya sebagai variabel yang dapat diakses di semua template.

Memulai Awal proyek

Menghubungkan Context Processor dengan Template

Setelah membuat context processor, langkah selanjutnya adalah mendaftarkannya di file `settings.py` agar dapat digunakan di semua template.

Buka file `settings.py` dan tambahkan context processor ke dalam konfigurasi `TEMPLATES`:

```
# File: platform_bot/settings.py

# Assets Management
ASSETS_ROOT = '/static/assets'

TEMPLATES = [
    {
        'BACKEND':
'django.template.backends.django.DjangoTemplates
',
        'DIRS': [BASE_DIR / 'apps/templates'],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                # Context processors bawaan
Django
'django.template.context_processors.debug',
'django.template.context_processors.request',
'django.contrib.auth.context_processors.auth',
'django.contrib.messages.context_processors.mess
ages',
```

Memulai Awal proyek

```
# Context processor khusus kita
'apps.context_processors.cfg_assets_root', #
Menambahkan context processor
    ],
    },
    },
]
```

Penambahan context processor

apps.context_processors.cfg_assets_root di bagian `context_processors` memastikan bahwa variabel **ASSETS_ROOT** tersedia di setiap template di seluruh aplikasi, tanpa perlu menambahkannya secara manual di setiap view.

Menggunakan Context Processor untuk Data Dinamis

Dengan context processor yang sudah ditambahkan, kita dapat dengan mudah mengakses data dinamis seperti `ASSETS_ROOT` di setiap template. Sebagai contoh, untuk menampilkan direktori aset pada halaman Home, kita dapat memanfaatkan variabel yang disediakan oleh context processor tanpa harus menambahkannya secara eksplisit dalam setiap view.

Langkah ini menyederhanakan cara kita menambahkan elemen dinamis ke halaman web, membuat pengelolaan data global menjadi lebih mudah dan terorganisir. Dengan context processor, kita mengurangi kebutuhan untuk menduplikasi data di banyak tempat dan memastikan bahwa data yang sama konsisten di seluruh template aplikasi.

4.1.7 Pengaturan Media di `settings.py`

Pastikan pengaturan `MEDIA_URL` dan `MEDIA_ROOT` sudah benar:

```
# settings.py

import os

# Media files
MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
```

Tambahkan URL Media di `urls.py`

Tambahkan konfigurasi untuk melayani file media selama pengembangan:

```
# platform_bot/urls.py
from django.contrib import admin
from django.urls import path, include
from django.conf import settings
from django.conf.urls.static import static

urlpatterns = [
    path('admin/', admin.site.urls),
    path('',
include('apps.authentication.urls')), #
Menyertakan URL aplikasi authentication
]
if settings.DEBUG:
    urlpatterns += static(settings.MEDIA_URL,
document_root=settings.MEDIA_ROOT)
```

4.2 Membuat Halaman Home (Landing Page)

Halaman home atau landing page adalah titik awal bagi pengguna saat mengunjungi aplikasi Anda. Ini adalah tempat pertama yang akan dilihat oleh pengguna dan berfungsi sebagai pengantar untuk berbagai fitur yang tersedia di aplikasi. Dalam konteks proyek `platform_bot`, halaman home akan menyediakan pintu gerbang bagi pengguna untuk melakukan registrasi, login, dan mendapatkan informasi dasar tentang aplikasi.

4.2.1 Tujuan Halaman Home

Halaman home memiliki peran krusial dalam aplikasi Anda. Ini adalah tempat di mana pengguna pertama kali berinteraksi dengan aplikasi dan sering kali merupakan bagian penting dari pengalaman pengguna. Halaman ini biasanya dirancang untuk menarik perhatian dan memberikan navigasi yang jelas ke fitur utama aplikasi. Dalam proyek `platform_bot`, halaman home akan menyambut pengguna dengan informasi singkat tentang platform bot dan menyediakan opsi untuk login atau registrasi.

Apa yang Akan Ditampilkan di Halaman Home

Di halaman home, kita akan menampilkan beberapa elemen kunci:

1. **Deskripsi Singkat Tentang Platform Bot:** Menyediakan informasi dasar tentang apa itu `platform_bot` dan bagaimana ia dapat membantu pengguna.
2. **Tombol Login dan Registrasi:** Memudahkan pengguna untuk masuk atau membuat akun baru.

Memulai Awal proyek

3. **Link ke Informasi Tambahan:** Mengarahkan pengguna ke halaman lain di aplikasi untuk informasi lebih lanjut.

Untuk membuat halaman home ini, kita akan fokus pada aplikasi `authentication`, karena ini adalah aplikasi yang akan menangani registrasi dan login pengguna.

4.2.2 Membuat View untuk Halaman Home

Dalam Django, view adalah komponen yang menangani logika untuk permintaan HTTP dan memberikan respons yang sesuai. Untuk halaman home, kita akan membuat sebuah view yang merender template HTML yang telah kita buat.

View di Django adalah fungsi Python yang menerima permintaan HTTP dan mengembalikan respons HTTP. Untuk membuat view di Django, Anda harus menambahkan fungsi view di dalam file `views.py` pada aplikasi yang relevan. Dalam kasus ini, kita akan menambahkan view di aplikasi `authentication`.

Buka file `apps/authentication/views.py` dan tambahkan fungsi berikut:

```
# File: apps/authentication/views.py

from django.shortcuts import render

# View untuk halaman home
def home(request):
    context = {
        'ASSETS_ROOT': '/static/assets', #
        # Tambahan path untuk asset static jika diperlukan
    }
```

Memulai Awal proyek

```
    }  
    return render(request, 'home/home.html',  
context)
```

Dalam contoh tersebut, context 'ASSETS_ROOT' dikirim ke template, memungkinkan Anda menggunakan variabel ini di template untuk mengelola file statis dengan lebih dinamis.

Fungsi `home` di atas menggunakan `render` untuk merender template `home.html`. Fungsi `render` memudahkan Anda untuk menghubungkan data dari view ke template HTML yang ditampilkan kepada pengguna.

*Penjelasan tentang **render** dan Konteks Data*

Fungsi `render` adalah alat yang sangat berguna dalam Django. Fungsi ini menggabungkan template HTML dengan data konteks yang diberikan dan mengembalikan sebuah `HttpResponse` yang dapat dikirimkan ke pengguna.

Sintaks fungsi `render` adalah sebagai berikut:

```
render(request, template_name, context=None)
```

- `request`: Objek permintaan HTTP.
- `template_name`: Nama file template HTML yang akan dirender.
- `context`: (Opsional) Sebuah dictionary yang berisi data yang ingin disertakan dalam template.

Memulai Awal proyek

Di template HTML, Anda dapat menggunakan data dari konteks seperti ini:

```
<!-- File:
apps/templates/authentication/base.html -->

<link href="{{ ASSETS_ROOT }}/css/style.css"
rel="stylesheet"/>
```

View ini akan merender template `home.html` yang akan kita buat di langkah berikutnya.

Pada contoh di atas, variabel `ASSETS_ROOT` kini tersedia di template, dan kita menggunakannya untuk menentukan jalur ke gambar atau file statis lainnya. Ini memudahkan pengelolaan jalur aset secara dinamis, terutama jika aset disimpan di lokasi tertentu yang dapat berubah.

4.2.3 Membuat Template untuk Halaman Home

Template inheritance adalah fitur di Django yang memungkinkan Anda untuk membuat template dasar yang dapat diwariskan oleh template lain. Ini sangat berguna untuk menjaga konsistensi tampilan di seluruh aplikasi Anda.

Selanjutnya, kita akan membuat file template untuk halaman home. Buka folder bernama `home` di dalam folder `apps/templates`. Di dalam folder ini, buat file bernama `base.html` dan `home.html`:

Memulai Awal proyek

```
$ touch apps/templates/home/base.html
$ touch apps/templates/home/home.html
```

Misalnya, Kita bisa membuat template dasar bernama `base.html` di folder `apps/templates` yang berisi elemen-elemen umum seperti header dan footer:

```
<!-- File: apps/templates/home/base.html -->

<!DOCTYPE html>
<html>
<head>
  <title>{% block title %}Platform Bot{%
endblock %}</title>

</head>
<body>
  <header>
    <h1>Platform Bot</h1>
    <nav>
      <a href="{% url 'home' %}">Home</a>
      <a href="#">Login</a>
      <a href="#">Register</a>
    </nav>
  </header>
  <main>
    {% block content %}
    <!-- Konten spesifik halaman akan
dimasukkan di sini -->
    {% endblock %}
  </main>
  <footer>
    <p>&copy; 2024 Platform Bot. All rights
reserved.</p>
  </footer>
```

Memulai Awal proyek

```
</body>
</html>
```

Kemudian, dalam template `home.html`, Anda dapat mewarisi dari `base.html` dan mengisi blok konten yang telah didefinisikan:

```
<!-- File: apps/templates/home/home.html -->

{% extends 'home/base.html' %}

{% block title %}Home - Platform Bot{% endblock %}

{% block content %}
    <h2>Selamat datang Platform Bot</h2>
    <p>Gerbang Anda untuk mengeloladan
    berintraksi dengan bot Anda.</p>
{% endblock %}
```

Dengan menggunakan template inheritance, Anda bisa menjaga template HTML Anda lebih terorganisir dan memudahkan pemeliharaan aplikasi.

Template ini akan menampilkan pesan sambutan, deskripsi singkat, dan tautan ke halaman login dan registrasi.

4.2.4 Menyusun Layout Halaman Home

Setelah berhasil membuat template dan view untuk halaman home, langkah selanjutnya adalah menyusun layout halaman agar tampak menarik, modern, dan responsif di berbagai perangkat, termasuk smartphone dan tablet. Layout yang responsif sangat

Memulai Awal proyek

penting karena memungkinkan halaman web menyesuaikan tampilannya berdasarkan ukuran layar pengguna.

Penjelasan Layout Responsif

Layout responsif adalah teknik desain web di mana elemen-elemen pada halaman dapat menyesuaikan posisi dan ukurannya secara dinamis sesuai dengan lebar layar. Ini penting karena pengguna mengakses aplikasi web dari berbagai perangkat dengan resolusi yang berbeda-beda, seperti ponsel, tablet, laptop, atau desktop.

Agar layout responsif, kita perlu menggunakan aturan CSS yang fleksibel, seperti menggunakan unit pengukuran yang adaptif (misalnya, `em`, `rem`, atau `%`), serta menggunakan grid system atau framework CSS yang dirancang khusus untuk desain responsif, seperti Bootstrap.

Penggunaan CSS dan Framework seperti Bootstrap

Salah satu cara paling efisien untuk membuat layout responsif adalah dengan menggunakan framework CSS seperti Bootstrap. Bootstrap menyediakan grid system yang fleksibel serta komponen yang dapat digunakan dengan mudah untuk membuat tampilan halaman lebih responsif dan interaktif.

Langkah pertama, kita harus menambahkan file Bootstrap ke dalam proyek. Anda bisa menambahkan file CSS Bootstrap secara langsung melalui CDN atau mengunduhnya dan menyimpannya di folder static.

Memulai Awal proyek

Menambahkan Bootstrap melalui CDN

Anda bisa menambahkan Bootstrap ke halaman home dengan cara menambahkan link CDN Bootstrap di bagian `<head>` dari template `base.html`.

```
<!-- File: apps/templates/home/base.html -->

<!DOCTYPE html>
<html>
<head>
    <title>{% block title %}Platform Bot{%
endblock %}</title>
    <!-- Menambahkan Bootstrap dari CDN -->
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/b
ootstrap.min.css" rel="stylesheet" integrity="sha384-
QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0J
MhjY6hW+ALEwIH" crossorigin="anonymous">

</head>
<body>
    <header class="bg-primary text-white text-
center py-3">
        <h1>Platform Bot</h1>
        <nav>
            <a href="{% url 'home' %}"
class="text-white">Home</a>
            <a href="#" class="text-
white">Login</a>
            <a href="#" class="text-
white">Register</a>
        </nav>
    </header>
    <main class="container mt-5">
        {% block content %}
```

Memulai Awal proyek

```
        {% endblock %}
    </main>
    <footer class="text-center py-4 bg-dark
text-white">
        <p>&copy; 2024 Platform Bot. All rights
reserved.</p>
    </footer>
</body>
</html>
```

Dengan menambahkan kode di atas, Anda sudah menyiapkan Bootstrap untuk digunakan di seluruh halaman web Anda. Selanjutnya, kita akan mengaplikasikan grid system Bootstrap di halaman `home.html`.

Menyusun Layout Halaman Home dengan Bootstrap

Sekarang kita akan menambahkan grid system pada halaman `home` agar tampilannya rapi dan responsif. Dalam template `home.html`, kita bisa menggunakan kelas-kelas Bootstrap untuk membuat struktur grid.

```
<!-- File: apps/templates/home/home.html -->

{% extends 'home/base.html' %}

{% block title %}Home - Platform Bot{% endblock %}

{% block content %}
<div class="jumbotron text-center">
    <h1 class="display-4">Selamat Datang di
Platform Bot</h1>
    <p class="lead">Gerbang Anda untuk mengelola
dan berinteraksi dengan bot Anda.</p>
    <hr class="my-4">
</div>
```


Memulai Awal proyek

```
<p>Buat dan kelola bot Anda dengan mudah  
melalui platform kami.</p>  
<a class="btn btn-primary btn-lg" href="#"  
role="button">Mulai Sekarang</a>  
</div>  
  
<div class="row">  
  <div class="col-md-4">  
    <h3>Mudah Digunakan</h3>  
    <p>Platform kami dirancang agar  
sederhana dan intuitif, memungkinkan Anda  
membuat dan mengelola bot dengan mudah.</p>  
  </div>  
  <div class="col-md-4">  
    <h3>Fleksibel</h3>  
    <p>Sesuaikan bot Anda sesuai kebutuhan  
dengan pengaturan dan alat yang fleksibel dari  
kami.</p>  
  </div>  
  <div class="col-md-4">  
    <h3>Dapat Diskalakan</h3>  
    <p>Apakah Anda mengelola satu bot atau  
ratusan, platform kami dapat berkembang sesuai  
kebutuhan bisnis Anda.</p>  
  </div>  
</div>  
{% endblock %}
```

Dalam layout di atas:

- **Jumbotron:** Bootstrap menyediakan komponen jumbotron, yang merupakan elemen besar yang menarik perhatian, biasanya digunakan untuk memberikan pesan utama atau highlight di halaman.
- **Grid System:** Kita menggunakan row dan col-md-4 untuk membuat grid tiga kolom yang responsif. Setiap kolom akan menyesuaikan ukurannya berdasarkan ukur-

Memulai Awal proyek

an layar pengguna. Pada layar kecil, kolom ini akan bertumpuk secara vertikal, sementara pada layar yang lebih lebar, mereka akan tampil dalam satu baris.

Menggunakan File CSS Kustom (opsional)

Selain menggunakan Bootstrap, Anda bisa menambahkan gaya kustom melalui file CSS kustom. Buat file `styles.css` di folder `static/css` dan tambahkan gaya kustom Anda di sana. Contoh sederhana:

```
/* File: apps/static/assets/css/styles.css */  
  
body {  
    background-color: #f8f9fa;  
}  
  
header {  
    background-color: #007bff;  
    color: white;  
    padding: 10px;  
}  
  
footer {  
    background-color: #343a40;  
    color: white;  
    padding: 20px;  
}  
  
.jumbotron {  
    background-color: #e9ecef;  
    padding: 2rem 1rem;  
}
```

Memulai Awal proyek

Pastikan Anda sudah menyertakan file ini di template `base.html` agar berlaku di semua halaman:

```
<link href="{{ ASSETS_ROOT }}/css/style.css"
rel="stylesheet"/>
```

Dengan langkah-langkah di atas, Anda telah berhasil menyusun layout halaman home yang responsif menggunakan Bootstrap dan file CSS kustom. Layout ini tidak hanya menarik secara visual, tetapi juga fleksibel dan mudah diakses dari berbagai perangkat, memastikan pengalaman pengguna yang optimal.

4.2.5 Menambahkan Navigasi dan Elemen Interaktif

Setelah menyusun layout dasar untuk halaman home, langkah selanjutnya adalah menambahkan navigasi dan elemen interaktif agar pengguna dapat melakukan tindakan seperti login dan registrasi dengan mudah. Navigasi yang baik harus intuitif dan mudah diakses dari perangkat apa pun. Selain itu, elemen interaktif seperti tombol dan formulir akan memudahkan pengguna dalam menavigasi halaman dan berinteraksi dengan aplikasi.

Membuat Navigasi untuk Login dan Registrasi

Navigasi adalah salah satu elemen terpenting di halaman home. Pada proyek ini, kita akan membuat navigasi yang berisi dua opsi utama: **Login** dan **Registrasi**. Kedua elemen ini akan terletak di bagian kanan atas halaman untuk memastikan aksesibilitas yang mudah bagi pengguna.

Memulai Awal proyek

Navigasi ini dapat dibuat menggunakan komponen *Navbar* dari Bootstrap, yang sudah kita tambahkan sebelumnya. Berikut ini adalah contoh penambahan navigasi di dalam file `apps/templates/authentication/base.html`:

```
<!-- File: apps/templates/home/base.html -->
<!DOCTYPE html>
<html>
<head>
  <title>Platform Bot</title>
  <!-- Menghubungkan CSS Bootstrap dari CDN -->
  <link
    href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMHjY6hW+ALEwIH" crossorigin="anonymous">
  </head>
<body>
  <!-- Bagian header untuk navigasi -->
  <header>
    <nav class="navbar navbar-expand-lg navbar-light bg-light">
      <a class="navbar-brand" href="#">Platform Bot</a>
      <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNav" aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
      </button>
    </nav>
  </header>
</body>
</html>
```

Memulai Awal proyek

```
        <div class="collapse navbar-  
collapse" id="navbarNav">  
            <ul class="navbar-nav ms-auto">  
                <li class="nav-item">  
                    <a class="nav-link"  
href="#">Masuk</a>  
                </li>  
                <li class="nav-item">  
                    <a class="nav-link"  
href="#">Daftar</a>  
                </li>  
            </ul>  
        </div>  
    </nav>  
</header>
```

Penjelasan Kode:

1. **Navbar:** Navigasi ini menggunakan elemen navbar dari Bootstrap, yang sudah diatur agar responsif. Jika pengguna membuka halaman di perangkat seluler, menu ini akan berubah menjadi tombol dropdown untuk menghemat ruang.
2. **Masuk dan Daftar:** Dua tautan di navigasi akan mengarah ke halaman login ({% url 'login' %}) dan registrasi ({% url 'register' %}). Tautan ini memudahkan pengguna untuk segera melakukan tindakan setelah melihat halaman home. Tapi karena kita belum membuat halaman dan views untuk login dan register jadi kita belum menambahkannya.

Menambahkan JavaScript untuk Interaksi Tambahan

Jika Anda ingin menambahkan elemen interaktif lebih lanjut, seperti validasi formulir atau transisi, Bootstrap juga menyediakan

Memulai Awal proyek

pustaka JavaScript yang dapat Anda gunakan. Kita sudah menambahkan pustaka ini di bagian bawah halaman sebelumnya:

```
<!-- Menyertakan JavaScript Bootstrap -->
<script
src="https://cdn.jsdelivr.net/npm/@popperjs/core
@2.11.8/dist/umd/popper.min.js"
integrity="sha384-I7E8VVD/ismYTF4hNIPjVp/Zjvgyol
6VFvRkX/vR+Vc4jQkC+hVqc2pM80Dewa9r"
crossorigin="anonymous"></script>
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.
3/dist/js/bootstrap.min.js" integrity="sha384-
0pUGZvbkm6XF6gxjEnlmuGrJXVbNuzT9qBBavbLwCs0GabYf
Zo0T0to5eqruptLy"
crossorigin="anonymous"></script>
```

Ini memungkinkan kita menambahkan interaksi dinamis di halaman, seperti animasi dropdown, modal, dan lain-lain, tanpa menulis banyak kode JavaScript manual. Anda dapat dengan mudah mengaktifkan komponen-komponen interaktif ini dengan menambahkan atribut HTML yang sesuai pada elemen yang diinginkan.

Kode lengkap

```
<!-- File: apps/templates/home/base.html -->
<!DOCTYPE html>
<html>
<head>
  <title>Platform Bot</title>
  <!-- Menghubungkan CSS Bootstrap dari CDN --
>
```

Memulai Awal proyek

```
<link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/b
ootstrap.min.css" rel="stylesheet" integrity="sha384-
QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0J
MhjY6hW+ALEwIH" crossorigin="anonymous">

</head>
<body>
  <!-- Bagian header untuk navigasi -->
  <header>
    <nav class="navbar navbar-expand-lg
navbar-light bg-light">
      <a class="navbar-brand"
href="#">Platform Bot</a>
      <button class="navbar-toggler"
type="button" data-toggle="collapse" data-
target="#navbarNav" aria-controls="navbarNav"
aria-expanded="false" aria-label="Toggle
navigation">
        <span class="navbar-toggler-
icon"></span>
      </button>
      <div class="collapse navbar-
collapse" id="navbarNav">
        <ul class="navbar-nav ms-auto">
          <li class="nav-item">
            <a class="nav-link"
href="#">Masuk</a>
          </li>
          <li class="nav-item">
            <a class="nav-link"
href="#">Daftar</a>
          </li>
        </ul>
      </div>
    </nav>
  </header>
```

Memulai Awal proyek

```
<main class="container mt-5">
    {% block content %}
    {% endblock %}
</main>
<footer class="text-center py-4 bg-dark
text-white">
    <p>&copy; 2024 Platform Bot. All rights
reserved.</p>
</footer>
<!-- Menyertakan JavaScript Bootstrap -->
<script
src="https://cdn.jsdelivr.net/npm/@popperjs/core
@2.11.8/dist/umd/popper.min.js"
integrity="sha384-I7E8VVD/ismYTF4hNIPjVp/Zjvgyol
6VFvRkX/vR+Vc4jQkC+hVqc2pM80Dewa9r"
crossorigin="anonymous"></script>
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.
3/dist/js/bootstrap.min.js" integrity="sha384-
0pUGZvbkm6XF6gxjEnlmuGrJXVbNuzT9qBBavbLwCsOGabYf
Zo0T0to5eqruptLy"
crossorigin="anonymous"></script>
</body>
</html>
```

Dengan menyusun navigasi dan elemen interaktif seperti ini, halaman home menjadi lebih fungsional dan mudah diakses oleh pengguna.

4.2.6 Mengonfigurasi URL untuk Halaman Home

Selanjutnya, kita perlu menambahkan konfigurasi URL untuk halaman home. Buka file `apps/authentication/urls.py` dan tambahkan URL baru untuk view home:

Memulai Awal proyek

```
# File: apps/authentication/urls.py

from django.urls import path
from .views import home

urlpatterns = [
    path('', home, name='home'),
    # Tambahkan URL untuk login dan registrasi
    di sini
]
```

Selanjutnya, kita perlu memastikan bahwa URL aplikasi `authentication` termasuk dalam routing utama proyek. Buka file `platform_bot/urls.py` dan tambahkan konfigurasi berikut:

```
# File: platform_bot/urls.py

from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('',
include('apps.authentication.urls')), #
Menyertakan URL aplikasi authentication
]
```

Dengan konfigurasi ini, halaman `home` akan ditampilkan ketika pengguna mengunjungi URL root dari aplikasi.

Dengan mengikuti langkah-langkah di atas, Anda telah berhasil membuat halaman `home` (landing page) untuk aplikasi `platform_bot`. Halaman ini akan menyambut pengguna dengan in-

Memulai Awal proyek

formasi dasar dan memberikan opsi untuk login atau registrasi. Selanjutnya, kita akan melanjutkan dengan menambahkan fungsionalitas lebih lanjut dan memastikan aplikasi berfungsi sesuai harapan.

4.2.7 Menguji Halaman Home

Setelah semua konfigurasi dilakukan, langkah terakhir adalah menguji apakah halaman home berfungsi dengan baik.

Menjalankan Server

Pertama, pastikan server Django berjalan. Jika belum, jalankan server menggunakan perintah berikut:

```
$ python manage.py runserver
```

Mengakses Halaman Home

Buka browser Anda dan akses halaman home dengan mengunjungi URL berikut:

```
http://127.0.0.1:8000/
```

Jika semua konfigurasi telah dilakukan dengan benar, Anda seharusnya melihat halaman home dengan gaya yang telah Anda sesuaikan dengan bootstrap. Pastikan semua elemen tampil dengan benar, termasuk header, hero section, features section, dan tombol.

Dengan ini, halaman home Anda sudah selesai dibuat, lengkap dengan view, template, URL routing, styling, dan asset statis. Jika ada masalah, pastikan untuk memeriksa kembali langkah-

Memulai Awal proyek

langkah di atas dan pastikan semua file dan konfigurasi telah dilakukan dengan benar.

Kesimpulan Bab

Pada Bab ini, kita telah mempelajari langkah-langkah untuk membangun halaman landing page (halaman home) pada proyek *Platform Bot*. Halaman ini berfungsi sebagai pintu utama yang menyambut pengguna baru dan memandu mereka untuk mendaftar atau login ke dalam platform.

Kita memulai dengan memahami tujuan dan peran halaman home dalam aplikasi. Kemudian, kita membahas bagaimana membuat *view* untuk halaman home menggunakan Django, dan mengintegrasikannya dengan template HTML yang memanfaatkan *template inheritance* untuk menjaga kode tetap terstruktur dan rapi.

Selanjutnya, kita mengelola layout responsif menggunakan *framework* CSS seperti Bootstrap, memastikan tampilan halaman dapat beradaptasi di berbagai perangkat. Terakhir, kita menambahkan navigasi dan elemen interaktif seperti tombol dan tautan menuju halaman login dan registrasi untuk meningkatkan pengalaman pengguna.

Dengan halaman home yang sudah terbangun, platform kita kini memiliki fondasi yang kuat sebagai antarmuka pengguna. Langkah selanjutnya adalah mengembangkan fitur-fitur lain seperti halaman dashboard dan manajemen bot, yang akan kita bahas pada bab-bab berikutnya.

Memulai Awal proyek

Semua langkah ini memastikan bahwa kita memiliki fondasi yang kuat untuk memulai pengembangan aplikasi web menggunakan Django.

BAB 5 - Fitur Registrasi, Login, dan Logout

Dalam bagian kita akan membahas dan membuat fitur untuk registrasi, login, dan logout untuk user. Form registrasi adalah bagian penting dari aplikasi yang memungkinkan pengguna untuk mendaftar dengan detail yang diperlukan. Kita juga akan menambahkan sistem login dan logout pengguna pada aplikasi Django. Proses ini melibatkan pembuatan form login, view untuk menangani login, serta template untuk menampilkan form login kepada pengguna.

5.1 Membuat Fitur Registrasi

Kita akan membahas bagaimana cara membuat fitur registrasi untuk pengguna di aplikasi Django Anda. Fitur registrasi ini memungkinkan pengguna baru untuk membuat akun di platform Anda. Kita juga akan membahas penggunaan model pengguna, baik model bawaan Django maupun model pengguna kustom jika diperlukan.

5.1.1 Membuat Model User

Django menyediakan model User bawaan yang dapat digunakan untuk autentikasi pengguna. Namun, jika Anda memerlukan informasi tambahan tentang pengguna, Anda bisa membuat model profil pengguna tambahan yang terhubung dengan model User.

Fitur Registrasi, Login, dan Logout

Dalam contoh ini, kita akan menggunakan model `User` bawaan Django dan membuat model `UserProfile` untuk menyimpan informasi tambahan tentang pengguna.

Model `UserProfile` ini akan berisi informasi tambahan seperti kota, alamat, negara, kode pos, deskripsi tentang pengguna, nama perusahaan, dan foto profil. Model ini menggunakan relasi satu-ke-satu dengan model `User`, yang memungkinkan setiap pengguna memiliki satu profil tambahan.

Berikut adalah kode untuk model `UserProfile`:

```
# File: apps/authentication/models.py

from django.contrib.auth.models import User
from django.db import models
from django.utils import timezone

class UserProfile(models.Model):
    user = models.OneToOneField(User,
on_delete=models.CASCADE)
    city = models.CharField(max_length=100,
blank=True)
    address = models.CharField(max_length=255,
blank=True)
    country = models.CharField(max_length=100,
blank=True)
    postal_code =
models.CharField(max_length=20, blank=True)
    about = models.TextField(blank=True)
    company = models.CharField(max_length=100,
blank=True)
```

Fitur Registrasi, Login, dan Logout

```
foto_profile =  
models.ImageField(upload_to='profile_pics/',  
blank=True)  
  
def __str__(self):  
    return self.user.username
```

Penjelasan Model

- **user**: Relasi satu-ke-satu dengan model `User`, yang berarti setiap entri `UserProfile` terkait dengan satu entri `User`.
- **city**: Menyimpan nama kota pengguna, bersifat opsional.
- **address**: Menyimpan alamat pengguna, bersifat opsional.
- **country**: Menyimpan nama negara pengguna, bersifat opsional.
- **postal_code**: Menyimpan kode pos pengguna, bersifat opsional.
- **about**: Menyimpan deskripsi atau informasi tambahan tentang pengguna, bersifat opsional.
- **company**: Menyimpan nama perusahaan pengguna, bersifat opsional.
- **foto_profile**: Menyimpan foto profil pengguna, diupload ke folder `profile_pics/`.

Langkah Selanjutnya

Setelah model `UserProfile` dibuat, kita perlu memberitahu Django untuk membuat tabel yang sesuai di database. Proses ini dilakukan dengan melakukan migrasi.

Fitur Registrasi, Login, dan Logout

Langkah pertama adalah membuat file migrasi dengan menjalankan perintah berikut di terminal:

```
$ python manage.py makemigrations
```

Output jika berhasil: Jika perintah berhasil, Anda akan melihat pesan seperti ini:

```
Migrations for 'users':  
users/migrations/0001_initial.py  
- Create model UserProfile
```

Perintah ini akan menghasilkan file migrasi untuk aplikasi `users`. Setelah file migrasi dibuat, jalankan perintah berikut untuk menerapkan migrasi dan membuat tabel di database:

```
$ python manage.py migrate
```

Output jika berhasil: Jika perintah berhasil, Anda akan melihat pesan seperti ini:

```
Operations to perform:  
  Apply all migrations: admin, auth, contenttypes, sessions, users  
Running migrations:  
  Applying contenttypes.0001_initial... OK  
  Applying auth.0001_initial... OK  
  Applying admin.0001_initial... OK
```

Fitur Registrasi, Login, dan Logout

```
Applying admin.0002_logentry_remove_auto_add... OK
Applying admin.0003_logentry_add_action_flag_choices... OK
Applying contenttypes.0002_remove_content_type_name... OK
Applying auth.0002_alter_permission_name_max_length... OK
Applying auth.0003_alter_user_email_max_length... OK
Applying auth.0004_alter_user_username_opts... OK
Applying auth.0005_alter_user_last_login_null... OK
Applying auth.0006_require_contenttypes_0002... OK
Applying auth.0007_alter_validators_add_error_messages...
OK
Applying auth.0008_alter_user_username_max_length... OK
Applying auth.0009_alter_user_last_name_max_length... OK
Applying auth.0010_alter_group_name_max_length... OK
Applying auth.0011_update_proxy_permissions... OK
Applying auth.0012_alter_user_first_name_max_length... OK
Applying sessions.0001_initial... OK
Applying users.0001_initial... OK
```

Setelah migrasi selesai, tabel baru untuk model `UserProfile` akan dibuat di database Anda.

Perintah `makemigrations` akan membuat file migrasi berdasarkan perubahan yang dilakukan pada model, sedangkan perintah `migrate` akan menerapkan migrasi ke database.

Fitur Registrasi, Login, dan Logout

Dengan model ini, Anda dapat menyimpan informasi tambahan tentang pengguna yang tidak disediakan oleh model `User` bawaan Django. Ini memberi Anda fleksibilitas untuk menyesuaikan profil pengguna sesuai kebutuhan aplikasi Anda.

Menambahkan Model `UserProfile` ke Django Admin

Agar model `UserProfile` dapat dikelola melalui Django admin site, kita perlu mendaftarkannya di `admin.py`. Dengan ini, admin dapat melihat, menambah, dan mengedit profil pengguna dari halaman admin.

Buka file `authentication/admin.py` dan tambahkan kode berikut:

```
# File: apps/authentication/admin.py

from django.contrib import admin
from .models import UserProfile

admin.site.register(UserProfile)
```

Kode di atas akan mendaftarkan model `UserProfile` di admin site Django, sehingga bisa dikelola dari antarmuka admin.

Sekarang, jika Anda menjalankan server Django dan membuka admin site, Anda akan melihat model `UserProfile` di sana. Anda bisa menambahkan, mengedit, atau menghapus profil pengguna melalui antarmuka ini.

Fitur Registrasi, Login, dan Logout

5.1.2 Membuat Form Registrasi

Dalam bagian ini, kita akan membahas cara membuat form registrasi pengguna dengan menggunakan Django. Form registrasi ini akan memungkinkan pengguna untuk mendaftar di aplikasi Anda dengan mengisi berbagai informasi yang diperlukan.

Untuk membuat form registrasi, kita akan menggunakan `forms.py` di dalam aplikasi `authentication`. Form ini akan mengumpulkan data dari pengguna dan melakukan validasi untuk memastikan bahwa data yang dimasukkan sesuai dengan yang diharapkan.

Berikut adalah kode untuk form registrasi yang menggunakan kelas `UserCreationForm` dari Django dan menambahkan beberapa field tambahan untuk informasi pengguna:

```
# File: apps/authentication/forms.py

from django import forms
from django.contrib.auth.forms import
UserCreationForm
from django.contrib.auth.models import User

class UserRegistrationForm(UserCreationForm):
    email =
forms.EmailField(widget=forms.EmailInput(attrs={
'class': 'form-control', 'placeholder':
'Email'}))
    username =
forms.CharField(widget=forms.TextInput(attrs={'c
lass': 'form-control', 'placeholder':
'Username'}))
```

Fitur Registrasi, Login, dan Logout

```
password1 =
forms.CharField(widget=forms.PasswordInput(attrs
={ 'class': 'form-control', 'placeholder':
'Password'}))
password2 =
forms.CharField(widget=forms.PasswordInput(attrs
={ 'class': 'form-control', 'placeholder':
'Confirm Password'}))
first_name = forms.CharField(max_length=100,
widget=forms.TextInput(attrs={ 'class': 'form-
control', 'placeholder': 'First Name'}))
last_name = forms.CharField(max_length=100,
widget=forms.TextInput(attrs={ 'class': 'form-
control', 'placeholder': 'Last Name'}))
company = forms.CharField(max_length=100,
required=False,
widget=forms.TextInput(attrs={ 'class': 'form-
control', 'placeholder': 'Company'}))
address = forms.CharField(max_length=255,
widget=forms.TextInput(attrs={ 'class': 'form-
control', 'placeholder': 'Address'}))
city = forms.CharField(max_length=100,
widget=forms.TextInput(attrs={ 'class': 'form-
control', 'placeholder': 'City'}))
country = forms.CharField(max_length=100,
widget=forms.TextInput(attrs={ 'class': 'form-
control', 'placeholder': 'Country'}))
postal_code = forms.CharField(max_length=20,
widget=forms.TextInput(attrs={ 'class': 'form-
control', 'placeholder': 'Postal Code'}))
about =
forms.CharField(widget=forms.Textarea(attrs={ 'cl
ass': 'form-control', 'placeholder': 'About
me'}), required=False)
foto_profile =
forms.ImageField(required=False,
widget=forms.FileInput(attrs={ 'class': 'form-
control-file'}))
```

Fitur Registrasi, Login, dan Logout

```
class Meta:
    model = User
    fields = ['username', 'email',
'password1', 'password2', 'first_name',
'last_name', 'company', 'address', 'city',
'country', 'postal_code', 'about',
'foto_profile']
```

Penjelasan Form:

- **email**: Field untuk email pengguna dengan widget EmailInput untuk memberikan placeholder dan kelas CSS.
- **username**: Field untuk nama pengguna dengan widget TextInput.
- **password1** dan **password2**: Field untuk kata sandi, dengan widget PasswordInput untuk menyembunyikan input pengguna.
- **first_name** dan **last_name**: Field untuk nama depan dan belakang pengguna.
- **company**: Field opsional untuk nama perusahaan pengguna.
- **address**: Field untuk alamat pengguna.
- **city**: Field untuk kota pengguna.
- **country**: Field untuk negara pengguna.
- **postal_code**: Field untuk kode pos pengguna.
- **about**: Field opsional untuk informasi tambahan tentang pengguna.
- **foto_profile**: Field opsional untuk mengunggah foto profil pengguna.

Validasi Input Pengguna

Fitur Registrasi, Login, dan Logout

Django menyediakan validasi bawaan untuk form, termasuk validasi untuk email dan kata sandi. Anda dapat menambahkan validasi tambahan jika diperlukan dengan mendefinisikan metode `clean` pada form. Misalnya, jika Anda ingin memvalidasi bahwa dua password yang dimasukkan sama, Anda dapat menggunakan metode ini untuk memeriksa kondisi tersebut.

Dengan form registrasi ini, pengguna dapat mendaftar dengan berbagai informasi yang diperlukan, dan form ini akan memastikan bahwa data yang dimasukkan sesuai dengan format yang diinginkan. Pastikan untuk menambahkan form ini ke dalam tampilan dan template yang sesuai untuk melengkapi fitur registrasi.

5.1.3 Membuat View untuk Registrasi

Setelah membuat form registrasi, langkah selanjutnya adalah mengatur cara menangani request registrasi di dalam view. Dalam bagian ini, kita akan menangani input dari form, melakukan validasi, dan menyimpan data pengguna ke dalam database. Selain itu, setelah registrasi berhasil, pengguna akan diarahkan ke halaman tertentu, seperti dashboard atau halaman utama.

View bertanggung jawab untuk menangani request yang datang dari pengguna. Dalam hal ini, kita akan menangani request POST dari form registrasi, memvalidasi input pengguna, menyimpan data pengguna ke dalam model `User`, serta membuat `User - Profile` yang terhubung dengan pengguna tersebut. Selain itu, kita juga akan secara otomatis melakukan login setelah registrasi berhasil dan mengarahkan pengguna ke halaman yang sesuai.

Fitur Registrasi, Login, dan Logout

Berikut adalah kode untuk view registrasi:

```
# File: apps/authentication/views.py

from django.shortcuts import render, redirect
from django.contrib.auth import login
from .forms import UserRegistrationForm
from .models import UserProfile

# View untuk menangani registrasi pengguna baru
def user_register(request):
    if request.method == 'POST':
        # Memeriksa apakah request adalah POST
        # dan memproses data form
        form =
        UserRegistrationForm(request.POST,
        request.FILES)
        if form.is_valid():
            # Menyimpan data pengguna ke dalam
            model User
            user = form.save()
            # Membuat dan menyimpan profil
            pengguna yang terhubung ke User
            UserProfile.objects.create(
                user=user,

            city=form.cleaned_data.get('city', ''),
            address=form.cleaned_data.get('address', ''),
            country=form.cleaned_data.get('country', ''),
            postal_code=form.cleaned_data.get('postal_code',
            ''),
            about=form.cleaned_data.get('about', ''),
```


Fitur Registrasi, Login, dan Logout

```
foto_profile=request.FILES.get('foto_profile',
None)
    )
    # Log in pengguna secara otomatis
    setelah registrasi
    login(request, user)
    # Redirect ke halaman home atau
    dashboard setelah registrasi berhasil
    return redirect('home')
else:
    # Jika request bukan POST, buat form
    kosong untuk ditampilkan di halaman
    form = UserRegistrationForm()

    # Menyiapkan konteks untuk template
    context = {
        'form': form,
        'ASSETS_ROOT': '/static/assets', #
Menyertakan path assets
    }

    # Render halaman register dengan form yang
    sudah disiapkan
    return render(request,
'account/register.html', context)
```

Penjelasan Kode:

1. **Request Handling:** View ini pertama-tama memeriksa apakah request yang diterima adalah POST. Jika POST, artinya pengguna telah mengirimkan form registrasi.
2. **Form Validation:** Form yang dikirim oleh pengguna divalidasi dengan `form.is_valid()`. Jika valid, data dari form akan disimpan ke dalam model User.

Fitur Registrasi, Login, dan Logout

3. **UserProfile**: Setelah pengguna berhasil didaftarkan, profil pengguna (model `UserProfile`) juga dibuat dan disimpan. Data tambahan seperti `city`, `address`, dan `foto_profile` diambil dari form dan dihubungkan dengan user yang baru terdaftar.
4. **Login Otomatis**: Setelah pengguna berhasil diregistrasi, kita secara otomatis melakukan login menggunakan fungsi `login()` dari Django, sehingga pengguna tidak perlu login secara manual setelah registrasi.
5. **Redirect Setelah Registrasi**: Setelah registrasi dan login berhasil, pengguna diarahkan ke halaman `home` atau halaman lain yang ditentukan dengan `redirect('home')`.
6. **Form Rendering**: Jika request bukan POST (misalnya, GET), maka form kosong akan dibuat dan ditampilkan di halaman registrasi (`register.html`).

Dengan membuat view ini, kita telah menyelesaikan penanganan request registrasi. Pengguna akan dapat mendaftar, sistem akan memvalidasi data yang dimasukkan, dan jika valid, pengguna akan langsung diarahkan ke halaman utama atau dashboard mereka.

5.1.4 Menyusun Template Registrasi

Setelah kita mengatur view untuk registrasi, langkah berikutnya adalah menyusun template HTML untuk form registrasi. Template ini akan menampilkan form kepada pengguna, memungkinkan mereka untuk memasukkan informasi yang diperlukan, serta menampilkan pesan validasi jika ada kesalahan input. Kami juga

Fitur Registrasi, Login, dan Logout

akan menambahkan styling untuk membuat tampilan form lebih menarik dan sesuai dengan desain yang diinginkan.

Untuk membuat template registrasi, kita akan membuat file HTML di folder `templates` yang ada di dalam folder aplikasi `authentication`. Berikut adalah langkah-langkah untuk menyusun template registrasi:

Membuat Template HTML

Buatlah file template HTML di folder `templates/account/` dengan nama `register.html`. Template ini akan menampilkan form registrasi yang telah kita buat sebelumnya.

```
<!-- File: apps/templates/account/register.html -->

{% extends 'home/base.html' %}

{% block title %}Register - Platform Bot{% endblock %}

{% block content %}
<div class="container mt-5">
  <h2 class="text-center">Registrasi Pengguna Baru</h2>
  <form method="POST" enctype="multipart/form-data">
    {% csrf_token %}
    <div class="form-group">
      {{ form.username.label_tag }}
      {{ form.username }}
      {% if form.username.errors %}
        <div class="text-danger">{{ form.username.errors }}</div>
      {% endif %}
    </div>
  </form>
</div>
{% endblock %}
```

Fitur Registrasi, Login, dan Logout

```
        {% endif %}
    </div>
    <div class="form-group">
        {{ form.email.label_tag }}
        {{ form.email }}
        {% if form.email.errors %}
            <div class="text-
danger">{{ form.email.errors }}</div>
        {% endif %}
    </div>
    <div class="form-group">
        {{ form.password1.label_tag }}
        {{ form.password1 }}
        {% if form.password1.errors %}
            <div class="text-
danger">{{ form.password1.errors }}</div>
        {% endif %}
    </div>
    <div class="form-group">
        {{ form.password2.label_tag }}
        {{ form.password2 }}
        {% if form.password2.errors %}
            <div class="text-
danger">{{ form.password2.errors }}</div>
        {% endif %}
    </div>
    <div class="form-group">
        {{ form.first_name.label_tag }}
        {{ form.first_name }}
        {% if form.first_name.errors %}
            <div class="text-
danger">{{ form.first_name.errors }}</div>
        {% endif %}
    </div>
    <div class="form-group">
        {{ form.last_name.label_tag }}
        {{ form.last_name }}
        {% if form.last_name.errors %}
```

Fitur Registrasi, Login, dan Logout

```

        <div class="text-
danger">{{ form.last_name.errors }}</div>
        {% endif %}
    </div>
    <div class="form-group">
        {{ form.company.label_tag }}
        {{ form.company }}
        {% if form.company.errors %}
            <div class="text-
danger">{{ form.company.errors }}</div>
            {% endif %}
        </div>
        <div class="form-group">
            {{ form.address.label_tag }}
            {{ form.address }}
            {% if form.address.errors %}
                <div class="text-
danger">{{ form.address.errors }}</div>
                {% endif %}
            </div>
            <div class="form-group">
                {{ form.city.label_tag }}
                {{ form.city }}
                {% if form.city.errors %}
                    <div class="text-
danger">{{ form.city.errors }}</div>
                    {% endif %}
                </div>
                <div class="form-group">
                    {{ form.country.label_tag }}
                    {{ form.country }}
                    {% if form.country.errors %}
                        <div class="text-
danger">{{ form.country.errors }}</div>
                        {% endif %}
                    </div>
                    <div class="form-group">
                        {{ form.postal_code.label_tag }}
                        {{ form.postal_code }}

```

Fitur Registrasi, Login, dan Logout

```
{% if form.postal_code.errors %}
    <div class="text-
danger">{{ form.postal_code.errors }}</div>
{% endif %}
</div>
<div class="form-group">
    {{ form.about.label_tag }}
    {{ form.about }}
    {% if form.about.errors %}
        <div class="text-
danger">{{ form.about.errors }}</div>
    {% endif %}
</div>
<div class="form-group">
    {{ form.foto_profile.label_tag }}
    {{ form.foto_profile }}
    {% if form.foto_profile.errors %}
        <div class="text-
danger">{{ form.foto_profile.errors }}</div>
    {% endif %}
</div>
<button type="submit" class="btn btn-
primary">Daftar</button>
</form>
</div>
{% endblock %}
```

Penjelasan Template:

1. **Template Inheritance:** Template `register.html` memperluas template dasar (`base.html`). Ini memastikan bahwa form registrasi akan menggunakan layout dan styling yang konsisten dengan halaman lainnya.
2. **Form Rendering:** Form ditampilkan menggunakan metode `form.field_name` yang dihasilkan dari `UserRegistrationForm`. Untuk setiap field, kita menam-

Fitur Registrasi, Login, dan Logout

pilkan label, input, dan jika ada error, kita menampilkan-nya dengan pesan error yang relevan.

3. **CSRF Token:** Token CSRF disertakan dengan `{% csrf_token %}` untuk melindungi form dari serangan CSRF (Cross-Site Request Forgery).
4. **Styling dan Validasi:** Form menggunakan kelas Bootstrap untuk styling. Pesan error untuk setiap field ditampilkan di bawah input field jika ada kesalahan validasi. Hal ini membantu pengguna memahami apa yang salah dengan input mereka dan bagaimana cara memperbaikinya.
5. **Enctype:** Atribut `enctype="multipart/form-data"` pada tag form diperlukan untuk upload file, seperti foto profil.

Dengan template ini, pengguna akan dapat melihat form registrasi dengan styling yang konsisten dan dapat langsung mengisi data mereka. Validasi input juga ditampilkan secara jelas, membantu pengguna memperbaiki kesalahan sebelum mengirimkan form.

5.1.5 Menambahkan Routing URLS Registrasi

Setelah menyelesaikan pengaturan model, form, view, dan template untuk registrasi, tahap selanjutnya adalah menambahkan url routing untuk halaman registrasi ini.

tambahkan path berikut di `urls.py` aplikasi `authentications`:

```
# File: apps/authentications/urls.py
```

Fitur Registrasi, Login, dan Logout

```
from django.urls import path
from .views import user_register

urlpatterns = [
    path('register/', user_register,
name='register'),
]
```

Jalankan server lokal Anda menggunakan perintah berikut:

```
$ python manage.py runserver
```

Setelah itu, buka browser dan akses halaman register. Coba login dengan kredensial yang benar dan salah untuk menguji validitas sistem autentikasi dan penanganan pesan kesalahan.

5.1.6 Testing Fitur Registrasi (opsional)

Setelah menyelesaikan pengaturan model, form, view, dan template untuk registrasi, tahap terakhir yang sangat penting adalah melakukan pengujian terhadap fungsionalitas registrasi ini. Pengujian memastikan bahwa fitur ini bekerja sesuai dengan yang diharapkan dan menghindari adanya bug atau kesalahan di masa mendatang. Langkah-langkah ini mencakup pengujian validasi form, pengiriman data, pembuatan pengguna baru, serta pengecekan apakah pengguna dapat login secara otomatis setelah registrasi.

Dalam tahap pengujian ini, kita akan menggunakan Django Testing Framework untuk menguji berbagai aspek dari fitur registrasi.

Fitur Registrasi, Login, dan Logout

Membuat Test Case untuk Registrasi

Pertama, kita akan membuat file pengujian untuk aplikasi authentication. Pengujian ini akan berada di file `tests.py` di dalam folder aplikasi tersebut.

Buat atau buka file `tests.py`:

```
# File: apps/authentication/tests.py

from django.test import TestCase
from django.urls import reverse
from django.contrib.auth.models import User
from apps.authentications.forms import
UserRegistrationForm

class UserRegistrationTest(TestCase):
    # Setup untuk test awal
    def setUp(self):
        self.register_url = reverse('register')
# URL untuk halaman registrasi
        self.user_data = {
            'username': 'testuser',
            'email': 'testuser@example.com',
            'password1': 'testpassword123',
            'password2': 'testpassword123',
            'first_name': 'Test',
            'last_name': 'User',
            'city': 'Test City',
            'country': 'Test Country',
            'postal_code': '12345',
            'about': 'This is a test user.',
            'company': 'Test Company',
            'address': '123 Test Street'
        }
```

Fitur Registrasi, Login, dan Logout

```
# Menguji apakah halaman registrasi dapat diakses
def test_register_page_accessible(self):
    response = self.client.get(self.register_url)
    self.assertEqual(response.status_code, 200)
    self.assertTemplateUsed(response, 'home/register.html')

# Menguji apakah form registrasi valid dengan data yang benar
def test_registration_form_valid(self):
    form = UserRegistrationForm(data=self.user_data)
    self.assertTrue(form.is_valid())

# Menguji pembuatan pengguna baru setelah registrasi
def test_user_created_after_registration(self):
    response = self.client.post(self.register_url, self.user_data)
    self.assertEqual(response.status_code, 302) # Harus redirect setelah registrasi berhasil

self.assertTrue(User.objects.filter(username='tester').exists())

# Menguji validasi jika password tidak cocok
def test_registration_password_mismatch(self):
    self.user_data['password2'] = 'wrongpassword123'
    form = UserRegistrationForm(data=self.user_data)
    self.assertFalse(form.is_valid())
```

Fitur Registrasi, Login, dan Logout

```
        self.assertIn('password2', form.errors)

        # Menguji apakah pengguna langsung login
        setelah registrasi
        def
        test_user_login_after_registration(self):
            response =
            self.client.post(self.register_url,
            self.user_data, follow=True)

            self.assertTrue(response.context['user'].is_auth
            enticated)
```

Menjalankan Pengujian

Setelah menulis test case, kita dapat menjalankan pengujian untuk memastikan bahwa fitur registrasi berfungsi dengan baik. Pengujian ini akan mencakup beberapa skenario: mengakses halaman registrasi, pengujian form, validasi password, dan pengecekan apakah pengguna bisa langsung login setelah registrasi berhasil.

Untuk menjalankan pengujian, buka terminal dan arahkan ke direktori proyek Django Anda, lalu jalankan perintah berikut:

```
$ python manage.py test apps.authentications
```

Jika pengujian berjalan dengan baik, Anda akan melihat output seperti berikut:

```
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
...
-----
-----
```

Fitur Registrasi, Login, dan Logout

```
Ran 4 tests in 0.543s
```

```
OK
```

Penjelasan Pengujian:

- ***test_register_page_accessible***: Menguji apakah halaman registrasi dapat diakses dengan kode status 200 dan apakah template yang benar digunakan.
- ***test_registration_form_valid***: Menguji validitas form registrasi dengan data yang valid. Form harus valid jika semua data yang diberikan benar.
- ***test_user_created_after_registration***: Menguji apakah pengguna baru dibuat setelah data dikirimkan ke server melalui form registrasi. Setelah berhasil, pengguna baru akan tersimpan di database, yang dicek dengan metode `filter`.
- ***test_registration_password_mismatch***: Menguji validasi kesalahan jika pengguna memasukkan dua password yang tidak cocok. Pengujian ini memastikan bahwa Django mengembalikan error pada form jika password tidak cocok.
- ***test_user_login_after_registration***: Menguji apakah pengguna langsung login secara otomatis setelah registrasi berhasil. Kita mengecek apakah konteks view mengandung objek pengguna yang telah terotentikasi.

Dengan melakukan pengujian-pengujian ini, kita memastikan bahwa fitur registrasi berfungsi dengan baik, mulai dari akses ha-

Fitur Registrasi, Login, dan Logout

laman, validasi input, hingga login otomatis. Pengujian adalah langkah penting dalam proses pengembangan aplikasi karena membantu mencegah masalah di masa depan dan memastikan aplikasi bekerja sesuai harapan.

5.2 Membuat Fitur Login

Setelah berhasil menyusun fitur registrasi, langkah selanjutnya adalah membuat fitur login. Fitur login ini penting untuk memungkinkan pengguna yang telah terdaftar untuk mengakses akun mereka. Kita akan menggunakan `forms.py` untuk membuat form login yang memungkinkan pengguna memasukkan kredensial mereka (username dan password) dan mengautentikasi mereka untuk masuk ke sistem.

5.2.1 Membuat Form Login

Untuk membuat form login, kita akan menggunakan kelas bawaan Django, yaitu `AuthenticationForm`. Kelas ini menyediakan form dasar untuk autentikasi pengguna, di mana kita dapat menambahkan kustomisasi seperti desain UI pada field-form yang ada, seperti menambahkan placeholder dan class CSS.

Buka file `forms.py` yang berada di dalam folder aplikasi `authentications` dan tambahkan kode berikut. Form ini terdiri dari dua field utama, yaitu `username` dan `password`, dengan atribut CSS yang telah ditambahkan untuk tampilan yang lebih baik di antarmuka.

```
# File: apps/authentications/forms.py
```

Fitur Registrasi, Login, dan Logout

```
from django.contrib.auth.forms import
UserCreationForm, AuthenticationForm

# Form untuk login pengguna
class UserLoginForm(AuthenticationForm):
    # Field untuk username
    username = forms.CharField(
        widget=forms.TextInput(attrs={'class':
'form-control', 'placeholder': 'Username'})
    )
    # Field untuk password
    password = forms.CharField(
        widget=forms.PasswordInput(attrs={'class':
'form-control', 'placeholder': 'Password'})
    )
```

Penjelasan forms:

- **username:** Field ini menggunakan widget `TextInput` dengan penambahan atribut `class` untuk styling dan `placeholder` untuk panduan pengguna.
- **password:** Field ini menggunakan widget `PasswordInput`, yang secara otomatis akan menyembunyikan teks yang dimasukkan pengguna.

Form ini akan digunakan untuk menangkap input dari pengguna saat mereka mencoba login ke dalam aplikasi.

Validasi dan Autentikasi Pengguna

Setelah membuat form, Django akan menangani proses autentikasi pengguna melalui `AuthenticationForm`. Form ini akan memastikan bahwa kombinasi `username` dan `password` yang

Fitur Registrasi, Login, dan Logout

dimasukkan pengguna cocok dengan data yang ada di database. Jika validasi berhasil, pengguna akan diizinkan masuk. Jika ada kesalahan (seperti password salah atau pengguna tidak ditemukan), maka pesan error akan dikembalikan ke form.

Kustomisasi lebih lanjut terhadap proses autentikasi bisa dilakukan dengan menambahkan logika tambahan di view atau middleware, namun untuk saat ini, kita akan menggunakan mekanisme bawaan Django yang sudah cukup mumpuni.

5.2.2 Membuat View untuk Login

Setelah kita membuat form login yang berfungsi untuk menangkap input pengguna, langkah berikutnya adalah membuat **view** yang akan menangani proses login itu sendiri. View ini berfungsi untuk memvalidasi kredensial pengguna dan mengatur logika autentikasi menggunakan form yang telah kita buat. Jika login berhasil, pengguna akan diarahkan ke halaman yang sesuai, misalnya dashboard. Namun, jika login gagal, kita juga perlu menampilkan pesan kesalahan yang informatif.

Menangani Login dan Redireksi Pengguna

Untuk menangani proses login, kita dapat memanfaatkan view di Django. Fungsi `user_login` akan bertanggung jawab untuk memproses input dari form login, memvalidasi kredensial, dan melakukan autentikasi pengguna.

Berikut adalah implementasi view untuk login pengguna yang perlu dimasukkan ke dalam file `views.py` di aplikasi `authentications`:

Fitur Registrasi, Login, dan Logout

```
# File: apps/authentications/views.py

from django.contrib.auth import authenticate,
login
from django.contrib import messages
from .forms import
UserRegistrationForm, UserLoginForm

# View untuk login pengguna
def user_login(request):
    if request.method == 'POST':
        # Mengambil data dari form dan
        memvalidasi
        form = UserLoginForm(data=request.POST)
        if form.is_valid():
            # Autentikasi pengguna
            user =
authenticate(username=form.cleaned_data['username'], password=form.cleaned_data['password'])
            if user is not None:
                # Login pengguna jika data valid
                login(request, user)
                messages.success(request, 'Login
berhasil.')
                return redirect('home') #
Mengarahkan pengguna setelah login berhasil
            else:
                # Menampilkan pesan error jika
                autentikasi gagal
                messages.error(request, 'Invalid
username or password.')
        else:
            form = UserLoginForm()

    context = {
        'ASSETS_ROOT': '/static/assets', #
        Tambahan path untuk asset static jika diperlukan
        'form': form,
```


Fitur Registrasi, Login, dan Logout

```
}  
return render(request, 'account/login.html',  
context)
```

Penjelasan kode:

- **Proses Request POST:** Ketika pengguna mengirimkan form login (dengan metode POST), view ini akan menerima data form tersebut dan memvalidasi input menggunakan `UserLoginForm`.
- **Autentikasi Pengguna:** Jika form valid, proses autentikasi dimulai dengan menggunakan `authenticate()` yang merupakan fungsi bawaan Django untuk memeriksa apakah kombinasi `username` dan `password` yang dimasukkan benar. Jika autentikasi berhasil, pengguna akan di-log in menggunakan fungsi `login()`, dan kemudian diarahkan ke halaman `home`.
- **Feedback untuk Pengguna:** Jika login berhasil, kita menampilkan pesan sukses menggunakan `messages.success()`. Namun, jika autentikasi gagal (misalnya karena kombinasi `username` dan `password` salah), kita akan menampilkan pesan error menggunakan `messages.error()`.
- **Request GET:** Jika halaman diakses menggunakan metode GET (misalnya saat pengguna pertama kali membuka halaman login), form login akan ditampilkan secara default.

Menambahkan Feedback Jika Login Gagal

Fitur Registrasi, Login, dan Logout

Django menyediakan sistem *messages framework* untuk memberikan umpan balik kepada pengguna dalam berbagai konteks. Pada view login ini, kita menggunakan fungsi *messages* untuk menampilkan pesan ketika login gagal. Jika kombinasi *user - name* dan *password* tidak cocok, pengguna akan melihat pesan kesalahan yang menjelaskan bahwa username atau password salah. Feedback ini penting untuk memastikan pengguna memahami apa yang salah dalam proses login.

Pada tahap ini, pengguna seharusnya dapat login dengan benar, dan jika terjadi kesalahan, mereka akan menerima feedback berupa pesan yang relevan.

5.2.3 Membuat Template Login

Setelah kita membuat form dan view untuk menangani login, langkah berikutnya adalah membuat *template HTML* untuk menampilkan form login. Template ini akan digunakan untuk menerima input dari pengguna, seperti *username* dan *password*. Kita juga akan menambahkan elemen interaktif dan styling agar tampilan form login terlihat menarik dan mudah digunakan.

Membuat Template HTML untuk Form Login

Untuk memulai, kita buat file `login.html` di dalam folder `templates/account/`. Template ini akan merender form login yang telah kita buat sebelumnya melalui view.

Berikut contoh template untuk halaman login:

```
<!-- File: apps/templates/account/login.html -->
```

Fitur Registrasi, Login, dan Logout

```
{% extends 'home/base.html' %}
{% block title %}Login - Platform Bot{% endblock %}

{% block content %}
    <div class="container">
        <div class="login-wrapper">
            <h2 class="login-title">Login</h2>

            <!-- Menampilkan pesan jika ada
error atau sukses -->
            {% if messages %}
                <div class="messages">
                    {% for message in messages %}
                        <div class="alert
{{ message.tags }}">{{ message }}</div>
                    {% endfor %}
                </div>
            {% endif %}

            <!-- Form login -->
            <form method="POST">
                {% csrf_token %}

                <!-- Field untuk username -->
                <div class="form-group">
                    {{ form.username.label_tag
}}
                    {{ form.username }}
                </div>

                <!-- Field untuk password -->
                <div class="form-group">
                    {{ form.password.label_tag
}}
                    {{ form.password }}
                </div>
            </form>
        </div>
    </div>
{% endblock %}
```

Fitur Registrasi, Login, dan Logout

```
        <!-- Tombol submit -->
        <div class="form-group">
            <button type="submit"
class="btn btn-primary">Login</button>
        </div>
    </form>

    <!-- Link untuk pengguna yang belum
memiliki akun -->
    <div class="register-link">
        <p>Belum punya akun? <a href="{%
url 'register' %}">Daftar sekarang</a>.</p>
    </div>
</div>
{% endblock %}
```

Penjelasan Template Login

- **Penggunaan CSS dan Asset Statis:** Pada bagian `<head>`, kita menghubungkan template dengan file CSS eksternal menggunakan `{{ ASSETS_ROOT }}`, yang sudah didefinisikan di view login untuk merujuk pada folder statis. Ini akan memastikan form login mendapatkan styling yang konsisten dengan bagian lain dari aplikasi.
- **Tampilan Pesan:** Kita menggunakan blok pesan `if messages` untuk menampilkan pesan kesalahan atau sukses yang dikirim dari view. Ini akan membantu memberikan feedback secara real-time kepada pengguna saat mereka mencoba login.
- **Form Login:** Template ini merender `form.username` dan `form.password` yang berasal dari `UserLogin-`

Fitur Registrasi, Login, dan Logout

Form di view. Masing-masing field ditempatkan dalam `div` dengan kelas `form-group` untuk memastikan elemen form tersusun dengan baik.

- **CSRF Token:** Jangan lupa untuk menyertakan `{% csrf_token %}` di dalam form. Ini adalah fitur keamanan yang disediakan oleh Django untuk mencegah serangan Cross-Site Request Forgery (CSRF).
- **Tombol Submit:** Tombol login dibuat menggunakan elemen `<button>` dengan kelas `btn btn-primary`. Kelas ini mengindikasikan bahwa kita menggunakan framework CSS seperti Bootstrap, namun Anda bisa menyesuaikannya sesuai dengan framework atau style manual yang Anda gunakan.
- **Link Daftar:** Jika pengguna belum memiliki akun, kita tambahkan link yang akan mengarahkan mereka ke halaman registrasi dengan menggunakan `{% url 'register' %}`.

5.2.4 Menambahkan Routing URLS Login

Setelah menyelesaikan pengaturan form, view, dan template untuk login, tahap selanjutnya adalah menambahkan url routing untuk halaman login ini.

Tambahkan path berikut di `urls.py` aplikasi `authentications`:

```
# File: apps/authentications/urls.py
from django.urls import path
```

Fitur Registrasi, Login, dan Logout

```
from .views import user_login

urlpatterns = [
    path('login/', user_login, name='login'),
]
```

Jalankan server lokal Anda menggunakan perintah berikut:

```
$ python manage.py runserver
```

Setelah itu, buka browser dan akses halaman login. Coba login dengan kredensial yang benar dan salah untuk menguji validitas sistem autentikasi dan penanganan pesan kesalahan.

5.2.5 Testing Fitur Login (opsional)

Setelah kita menyelesaikan pembuatan form login, view login, serta templatanya, langkah selanjutnya adalah melakukan ***pengujian fitur login***. Pengujian ini bertujuan untuk memastikan bahwa fitur login berfungsi sesuai dengan yang diharapkan, termasuk menangani berbagai skenario login yang mungkin terjadi. Kita akan menggunakan ***Django's test framework*** untuk menguji fitur login dengan pendekatan sistematis dan efisien.

Menguji Login dengan Berbagai Skenario

Langkah pertama dalam pengujian fitur login adalah memastikan bahwa pengguna bisa login dengan informasi yang valid dan mendapatkan pesan error jika mereka menggunakan informasi yang tidak valid. Beberapa skenario utama yang harus diuji meliputi:

- ***Skenario Login Berhasil:*** Pengguna memasukkan username dan password yang benar dan berhasil login.

Fitur Registrasi, Login, dan Logout

- **Skenario Login Gagal:** Pengguna memasukkan username atau password yang salah dan mendapatkan pesan error.
- **Skenario Tanpa Input:** Pengguna mencoba login tanpa mengisi field apapun dan mendapatkan pesan validasi yang sesuai.

Menyiapkan Pengujian dengan Django Test Framework

Django menyediakan test framework bawaan yang sangat berguna untuk menguji aplikasi secara otomatis. Untuk memulai, kita buat file baru di dalam direktori aplikasi untuk menyimpan pengujian ini.

Misalnya, untuk aplikasi **authentications**, kita buat file `test_views.py` di dalam folder `tests/`. Django biasanya sudah membuat direktori `tests/` secara otomatis ketika kita memulai project, tetapi jika belum ada, kita bisa membuatnya secara manual.

```
$ mkdir -p apps/authentications/tests
$ touch apps/authentications/tests/test_views.py
```

Menulis Test Case untuk Fitur Login

Setelah file untuk pengujian sudah siap, kita mulai menulis test case untuk menguji login. Pengujian ini akan menggunakan `TestCase` dari Django untuk melakukan simulasi HTTP request dan mengevaluasi respon yang diberikan oleh view login.

Berikut adalah contoh pengujian untuk fitur login:

Fitur Registrasi, Login, dan Logout

```
# File: apps/authentications/tests/test_views.py
from django.test import TestCase
from django.urls import reverse
from django.contrib.auth.models import User

class UserLoginTest(TestCase):

    def setUp(self):
        # Membuat pengguna untuk menguji login
        self.user =
User.objects.create_user(username='testuser',
password='password123')

    def test_login_page_renders_correctly(self):
        # Menguji apakah halaman login dirender
dengan benar
        response =
self.client.get(reverse('login'))
        self.assertEqual(response.status_code,
200)
        self.assertTemplateUsed(response,
'account/login.html')

    def test_login_successful(self):
        # Menguji apakah pengguna dapat login
dengan kredensial yang benar
        response =
self.client.post(reverse('login'), {
            'username': 'testuser',
            'password': 'password123'
        })
        self.assertRedirects(response,
reverse('dashboard')) # Menguji apakah pengguna
diarahkan ke dashboard

    def test_login_invalid_credentials(self):
        # Menguji apakah pengguna gagal login
dengan kredensial yang salah
```


Fitur Registrasi, Login, dan Logout

```
        response =
self.client.post(reverse('login'), {
    'username': 'testuser',
    'password': 'wrongpassword'
})
self.assertEqual(response.status_code,
200) # Tetap berada di halaman login
self.assertContains(response, 'Invalid
username or password') # Pesan error
ditampilkan

def test_login_empty_fields(self):
    # Menguji apakah pengguna gagal login
    tanpa memasukkan data apapun
    response =
self.client.post(reverse('login'), {
    'username': '',
    'password': ''
})
self.assertEqual(response.status_code,
200) # Tetap berada di halaman login
self.assertContains(response, 'This
field is required') # Pesan validasi
ditampilkan
```

Penjelasan Pengujian

- **setUp Method:** Method setUp dijalankan sebelum setiap test case. Di sini, kita membuat pengguna (testuser) untuk keperluan pengujian login.
- **Test Halaman Login:** Pengujian pertama (test_login_page_renders_correctly) memastikan bahwa halaman login dirender dengan benar dan menggunakan template yang sesuai (login.html).
- **Test Login Berhasil:** Test kedua (test_login_successful) menguji apakah pengguna dapat login de-

Fitur Registrasi, Login, dan Logout

ngan kredensial yang benar dan apakah mereka diarahkan ke halaman dashboard setelah berhasil login.

- **Test Login Gagal (Kredensial Salah):** Test ketiga (`test_login_invalid_credentials`) memastikan bahwa pengguna yang memasukkan username atau password yang salah mendapatkan pesan error dan tetap berada di halaman login.
- **Test Login Gagal (Field Kosong):** Test terakhir (`test_login_empty_fields`) memastikan bahwa pengguna yang mencoba login tanpa mengisi field apapun akan melihat pesan validasi yang sesuai.

Menjalankan Pengujian

Setelah kita selesai menulis test case, kita bisa menjalankan pengujian menggunakan perintah berikut di terminal:

```
$ python manage.py test authentications
```

Perintah ini akan menjalankan semua pengujian di dalam aplikasi `authentications`, termasuk pengujian untuk fitur login yang baru saja kita buat. Jika semua pengujian berhasil, kita akan melihat output yang mengindikasikan bahwa semua test lulus.

```
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
. . . .
-----
-----
Ran 4 tests in 0.123s

OK
```

Fitur Registrasi, Login, dan Logout

Mengatasi Error atau Kegagalan

Jika salah satu pengujian gagal, Django akan memberikan deskripsi tentang kesalahan yang terjadi, seperti HTTP status code yang tidak sesuai atau pesan error yang tidak ditampilkan sebagaimana mestinya. Kita bisa memperbaiki bagian yang salah di kode aplikasi atau test case, kemudian menjalankan ulang pengujian hingga semua test berhasil.

Testing sangat penting untuk memastikan bahwa fitur login berfungsi dengan baik dalam berbagai skenario dan kondisi. Dengan menggunakan Django test framework, kita bisa mengotomatiskan pengujian ini, sehingga lebih efisien dan akurat dalam memastikan fungsionalitas aplikasi.

5.3 Membuat Fitur Logout

Setelah fitur login berfungsi dengan baik dan telah diuji, kita akan melanjutkan untuk menambahkan fitur **logout** ke dalam aplikasi kita. Fitur ini penting agar pengguna dapat keluar dari akun mereka dengan aman, dan sesi mereka diakhiri dengan benar. Dalam Django, proses logout cukup sederhana karena Django sudah menyediakan fungsi `logout` yang dapat menangani berbagai aspek seperti membersihkan sesi pengguna.

5.3.1 Mengatur View untuk Logout

Untuk memulai, kita akan membuat **view untuk logout**. View ini akan memanggil fungsi `logout` dari Django, yang secara otomatis menghapus session dan cookies pengguna, sehingga pengguna tidak lagi dianggap sedang login. Setelah logout, kita akan

Fitur Registrasi, Login, dan Logout

mengarahkan pengguna kembali ke halaman login atau halaman lain yang sesuai.

View untuk logout sangatlah sederhana. Pertama, kita akan menambahkan view `user_logout` yang memanggil fungsi `logout` untuk mengakhiri sesi pengguna dan kemudian mengarahkan mereka kembali ke halaman login.

Kode view logout yang sudah kita siapkan adalah sebagai berikut:

```
# File: apps/authentications/views.py

from django.contrib.auth import logout

# View untuk logout pengguna
def user_logout(request):
    # Fungsi logout akan menghapus session
    # pengguna
    logout(request)
    # Setelah logout, pengguna diarahkan kembali
    # ke halaman login
    return redirect('login')
```

- **Fungsi `logout(request)`**: Fungsi ini akan membersihkan sesi pengguna, termasuk menghapus cookies yang terkait dengan login. Pengguna yang sudah logout tidak akan memiliki akses ke halaman yang memerlukan autentikasi.
- **Redireksi**: Setelah proses logout selesai, kita menggunakan `redirect('login')` untuk mengarahkan pengguna ke halaman login. Anda juga bisa mengganti halaman redireksi ini dengan halaman lain, misalnya hala-

Fitur Registrasi, Login, dan Logout

man utama atau halaman informasi logout yang memberikan pesan kepada pengguna bahwa mereka telah berhasil logout.

5.3.2 Mengatur URL Routing Logout

Setelah view logout selesai dibuat, kita perlu menambahkan URL yang akan memetakan view tersebut. Masukkan URL logout ke dalam file `urls.py` aplikasi `authentications`.

```
# File: apps/authentications/urls.py

from django.urls import path
from .views import user_logout

urlpatterns = [
    path('logout/', user_logout, name='logout'),
    # URL lainnya...
]
```

Dengan menambahkan path ini, pengguna dapat mengakses halaman logout melalui URL `/logout/`. View `user_logout` akan dijalankan setiap kali URL tersebut diakses, dan pengguna akan keluar dari sesi mereka.

Menangani Session dan Cookies (opsional)

Ketika pengguna melakukan logout, Django secara otomatis akan menangani penghapusan sesi dan cookies yang terkait dengan autentikasi. Session adalah mekanisme yang digunakan untuk menyimpan status login pengguna, dan ketika pengguna logout, session tersebut dihapus.

Fitur Registrasi, Login, dan Logout

Selain itu, jika aplikasi Anda menggunakan **cookies** untuk menyimpan informasi tambahan, seperti preferensi pengguna, cookie tersebut tidak akan dihapus oleh `logout`. Jadi, jika Anda memiliki cookies kustom yang ingin Anda hapus pada saat logout, Anda dapat menambahkannya secara manual dalam view `user_logout`.

Contoh menghapus cookies kustom:

```
# File: apps/authentications/views.py

from django.contrib.auth import logout
from django.shortcuts import redirect

def user_logout(request):
    # Logout pengguna
    logout(request)

    # Hapus cookies kustom (misalnya,
    # 'remember_me')
    if 'remember_me' in request.COOKIES:
        response = redirect('login')
        response.delete_cookie('remember_me')
        return response

    # Setelah logout, arahkan pengguna kembali
    # ke halaman login
    return redirect('login')
```

Dalam contoh di atas, selain menghapus sesi, kita juga menghapus cookie `remember_me` jika ada. Ini memastikan bahwa semua data yang terkait dengan sesi pengguna dihapus ketika mereka logout.

Fitur Registrasi, Login, dan Logout

Menambahkan Tombol Logout di Template

Agar pengguna dapat dengan mudah logout, kita perlu menambahkan tombol logout di template, misalnya di header atau menu navigasi, agar selalu terlihat oleh pengguna yang sedang login. Tombol ini akan mengarah ke URL `/logout/`.

Contoh kode HTML untuk menambahkan tombol logout:

```
<!-- File: templates/base.html -->

{% if user.is_authenticated %}
    <a href="{% url 'logout' %}" class="btn btn-
outline-danger">Logout</a>
{% endif %}
```

Pada template di atas, kita menggunakan `if user.is_authenticated` untuk memeriksa apakah pengguna sedang login. Jika pengguna sudah login, tombol logout akan ditampilkan. Ketika pengguna mengklik tombol ini, mereka akan diarahkan ke URL `/logout/` yang menjalankan view `user_logout`.

5.3.3 Mengatur Redirect Setelah Logout

Setelah berhasil mengimplementasikan fitur logout, kita dapat menyesuaikan halaman yang dituju setelah pengguna keluar dari sesi mereka. Secara default, dalam contoh sebelumnya, pengguna diarahkan ke halaman login setelah logout. Namun, tergantung pada kebutuhan aplikasi, kita dapat mengatur agar pengguna diarahkan ke halaman lain, seperti halaman utama atau halaman home.

Fitur Registrasi, Login, dan Logout

Menentukan halaman tujuan setelah logout cukup fleksibel, dan kita bisa melakukannya dengan mudah menggunakan fungsi `redirect`. Jika kita ingin mengarahkan pengguna ke halaman utama setelah logout, cukup mengganti `redirect('login')` dengan `redirect('home')` atau halaman lain yang sesuai.

Berikut ini adalah modifikasi sederhana pada view logout agar mengarahkan pengguna ke halaman utama:

```
# File: apps/authentications/views.py

from django.contrib.auth import logout
from django.shortcuts import redirect

def user_logout(request):
    # Logout pengguna
    logout(request)
    # Setelah logout, arahkan pengguna ke
    # halaman utama
    return redirect('login') # Ganti dengan
    'home' atau halaman lain yang diinginkan
```

Dalam contoh ini, setelah pengguna melakukan logout, mereka akan diarahkan ke halaman login (login). Halaman ini ditentukan melalui path URL `login`, yang sebelumnya harus sudah didefinisikan di `urls.py` aplikasi Anda. Jika halaman `login` belum ada, pastikan untuk membuatnya.

Menentukan halaman tujuan logout ini sangat penting untuk meningkatkan pengalaman pengguna. Mengarahkan mereka ke halaman yang relevan atau menyajikan pesan "Anda telah berhasil logout" dapat memberikan rasa interaksi yang lebih baik.

Fitur Registrasi, Login, dan Logout

Jika aplikasi Anda menggunakan halaman home atau dashboard yang banyak diakses, pengaturan ini bisa menjadi pilihan yang tepat untuk menjaga alur navigasi pengguna setelah mereka keluar dari akun.

5.3.4 Testing Fitur Logout (Opsional)

Setelah mengimplementasikan fitur logout, penting untuk mengujinya dengan berbagai skenario untuk memastikan bahwa fungsi logout bekerja dengan benar dan sesi pengguna benar-benar ditutup. Django menyediakan framework testing yang dapat kita gunakan untuk mengotomatisasi proses pengujian ini.

Menggunakan Django's Test Framework

Untuk menguji fitur logout, kita akan membuat test case yang memverifikasi apakah sesi pengguna dihapus dan apakah pengguna diarahkan ke halaman yang benar setelah logout. Pengujian ini akan dilakukan menggunakan framework test

Django yang ada di file `tests.py`.

Langkah-langkah pengujian logout:

- Pengguna login terlebih dahulu.
- Pengguna mengakses URL logout.
- Sistem memastikan sesi pengguna dihapus dan mereka tidak lagi dianggap sebagai pengguna yang terautentikasi.
- Memastikan pengguna diarahkan ke halaman yang sesuai (misalnya, home atau login).

Fitur Registrasi, Login, dan Logout

Contoh pengujian:

```
# File: apps/authentications/tests.py

from django.test import TestCase
from django.urls import reverse
from django.contrib.auth.models import User

class LogoutTestCase(TestCase):
    def setUp(self):
        # Buat pengguna baru untuk digunakan
        # dalam pengujian
        self.user =
        User.objects.create_user(username='testuser',
        password='password123')

    def test_logout(self):
        # Login pengguna sebelum melakukan
        # logout
        self.client.login(username='testuser',
        password='password123')

        # Pastikan pengguna terautentikasi
        response =
        self.client.get(reverse('home'))
        self.assertEqual(response.status_code,
        200)

        self.assertTrue(response.context['user'].is_auth
        enticated)

        # Akses URL logout
        response =
        self.client.get(reverse('logout'))

        # Pastikan pengguna diarahkan ke halaman
        # yang benar setelah logout
```

Fitur Registrasi, Login, dan Logout

```
self.assertRedirects(response,
reverse('home'))

# Pastikan pengguna tidak lagi
terautentikasi
response =
self.client.get(reverse('home'))

self.assertFalse(response.context['user'].is_authenticated)
```

- **SetUp Method:** Pada awal pengujian, kita membuat pengguna baru (`testuser`) dengan password tertentu. Pengguna ini digunakan dalam setiap pengujian logout.
- **Login Test:** Kita melakukan login menggunakan `self.client.login()` dan memastikan bahwa pengguna berhasil login dan terautentikasi.
- **Logout Test:** Setelah login, kita mengakses URL logout menggunakan `self.client.get(reverse('logout'))` untuk melakukan logout.
- **Redireksi dan Sesi:** Pengujian memastikan pengguna diarahkan ke halaman home setelah logout, dan juga mengecek apakah sesi pengguna sudah dihapus, yaitu memastikan bahwa pengguna tidak lagi terautentikasi.

Menggunakan Perintah Terminal

Setelah kode pengujian logout siap, jalankan pengujian menggunakan perintah terminal untuk memastikan semua skenario bekerja dengan benar. Jalankan perintah berikut dari direktori proyek Anda:

```
$ python manage.py test
```

Fitur Registrasi, Login, dan Logout

Perintah ini akan menjalankan semua pengujian yang ada di file `tests.py`, termasuk pengujian fitur logout. Jika semua pengujian berhasil, output terminal akan menunjukkan bahwa semua fitur berfungsi dengan baik, termasuk logout.

Skenario Pengujian Tambahan

Selain skenario dasar di atas, Anda juga dapat menguji logout dengan berbagai variasi, seperti:

- ***Logout tanpa login terlebih dahulu:*** Mengakses URL logout tanpa login dan memastikan sistem tidak mengalami error serta pengguna tetap diarahkan ke halaman yang benar.
- ***Memastikan tidak ada akses setelah logout:*** Mengakses halaman yang membutuhkan autentikasi setelah logout untuk memastikan pengguna tidak bisa lagi mengaksesnya.

Dengan melakukan pengujian ini, Anda dapat memastikan bahwa fitur logout berjalan dengan sempurna, menghapus sesi pengguna, dan mengarahkan mereka ke halaman yang tepat setelah logout. Hal ini penting untuk memastikan keamanan aplikasi, terutama dalam hal menangani sesi pengguna yang sensitif.

5.4 Kode Lengkap

Sebelumnya kita akan review kembali kode yang sudah kita buat di app authentication kita, agar tidak ada kesalahan dalam penulisan kode nya. Berikut adalah kode lengkap nya.

Fitur Registrasi, Login, dan Logout

5.4.1 Models.py

```
# File: apps/authentication/models.py

from django.contrib.auth.models import User
from django.db import models
from django.utils import timezone

class UserProfile(models.Model):
    user = models.OneToOneField(User,
on_delete=models.CASCADE)
    city = models.CharField(max_length=100,
blank=True)
    address = models.CharField(max_length=255,
blank=True)
    country = models.CharField(max_length=100,
blank=True)
    postal_code =
models.CharField(max_length=20, blank=True)
    about = models.TextField(blank=True)
    company = models.CharField(max_length=100,
blank=True)
    foto_profile =
models.ImageField(upload_to='profile_pics/',
blank=True)

    def __str__(self):
        return self.user.username
```

5.4.2 Forms.py

```
# File: apps/authentication/forms.py

from django import forms
from django.contrib.auth.forms import
UserCreationForm, AuthenticationForm
from django.contrib.auth.models import User
```

Fitur Registrasi, Login, dan Logout

```
class UserRegistrationForm(UserCreationForm):
    email =
forms.EmailField(widget=forms.EmailInput(attrs={
'class': 'form-control', 'placeholder':
'Email'}))
    username =
forms.CharField(widget=forms.TextInput(attrs={'c
lass': 'form-control', 'placeholder':
'Username'}))
    password1 =
forms.CharField(widget=forms.PasswordInput(attrs
={'class': 'form-control', 'placeholder':
'Password'}))
    password2 =
forms.CharField(widget=forms.PasswordInput(attrs
={'class': 'form-control', 'placeholder':
'Confirm Password'}))
    first_name = forms.CharField(max_length=100,
widget=forms.TextInput(attrs={'class': 'form-
control', 'placeholder': 'First Name'}))
    last_name = forms.CharField(max_length=100,
widget=forms.TextInput(attrs={'class': 'form-
control', 'placeholder': 'Last Name'}))
    company = forms.CharField(max_length=100,
required=False,
widget=forms.TextInput(attrs={'class': 'form-
control', 'placeholder': 'Company'}))
    address = forms.CharField(max_length=255,
widget=forms.TextInput(attrs={'class': 'form-
control', 'placeholder': 'Address'}))
    city = forms.CharField(max_length=100,
widget=forms.TextInput(attrs={'class': 'form-
control', 'placeholder': 'City'}))
    country = forms.CharField(max_length=100,
widget=forms.TextInput(attrs={'class': 'form-
control', 'placeholder': 'Country'}))
```

Fitur Registrasi, Login, dan Logout

```
        postal_code = forms.CharField(max_length=20,
widget=forms.TextInput(attrs={'class': 'form-
control', 'placeholder': 'Postal Code'}))
        about =
forms.CharField(widget=forms.Textarea(attrs={'cl
ass': 'form-control', 'placeholder': 'About
me'}), required=False)
        foto_profile =
forms.ImageField(required=False,
widget=forms.FileInput(attrs={'class': 'form-
control-file'}))

        class Meta:
            model = User
            fields = ['username', 'email',
'password1', 'password2', 'first_name',
'last_name', 'company', 'address', 'city',
'country', 'postal_code', 'about',
'foto_profile']

# Form untuk login pengguna
class UserLoginForm(AuthenticationForm):
    # Field untuk username
    username = forms.CharField(
        widget=forms.TextInput(attrs={'class':
'form-control', 'placeholder': 'Username'})
    )
    # Field untuk password
    password = forms.CharField(
        widget=forms.PasswordInput(attrs={'class':
'form-control', 'placeholder': 'Password'})
    )
```

5.4.3 Views.py

```
# File: apps/authentication/views.py

from django.shortcuts import render, redirect
from django.contrib.auth import authenticate, login, logout
from django.contrib import messages
from .forms import
UserRegistrationForm, UserLoginForm
from .models import UserProfile

# View untuk halaman home
def home(request):
    context = {
        'ASSETS_ROOT': '/static/assets', #
        # Tambahan path untuk asset static jika diperlukan
    }
    return render(request, 'home/home.html',
context)

# View untuk menangani registrasi pengguna baru
def user_register(request):
    if request.method == 'POST':
        # Memeriksa apakah request adalah POST
        # dan memproses data form
        form =
        UserRegistrationForm(request.POST,
request.FILES)
        if form.is_valid():
            # Menyimpan data pengguna ke dalam
            model User
            user = form.save()
            # Membuat dan menyimpan profil
            pengguna yang terhubung ke User
            UserProfile.objects.create(
                user=user,
```


Fitur Registrasi, Login, dan Logout

```
city=form.cleaned_data.get('city', ''),
address=form.cleaned_data.get('address', ''),
country=form.cleaned_data.get('country', ''),
postal_code=form.cleaned_data.get('postal_code',
''),

about=form.cleaned_data.get('about', ''),

foto_profile=request.FILES.get('foto_profile',
None)
    )
    # Log in pengguna secara otomatis
    setelah registrasi
    login(request, user)
    # Redirect ke halaman home atau
    dashboard setelah registrasi berhasil
    return redirect('home')
else:
    # Jika request bukan POST, buat form
    kosong untuk ditampilkan di halaman
    form = UserRegistrationForm()

    # Menyiapkan konteks untuk template
    context = {
        'form': form,
        'ASSETS_ROOT': '/static/assets', #
    Menyertakan path assets
    }

    # Render halaman register dengan form yang
    sudah disiapkan
    return render(request,
'account/register.html', context)
```

Fitur Registrasi, Login, dan Logout

```
# View untuk login pengguna
def user_login(request):
    if request.method == 'POST':
        # Mengambil data dari form dan memvalidasi
        form = UserLoginForm(data=request.POST)
        if form.is_valid():
            # Autentikasi pengguna
            user = authenticate(username=form.cleaned_data['username'], password=form.cleaned_data['password'])
            if user is not None:
                # Login pengguna jika data valid
                login(request, user)
                messages.success(request, 'Login berhasil.')
                return redirect('home') # Mengarahkan pengguna setelah login berhasil
            else:
                # Menampilkan pesan error jika autentikasi gagal
                messages.error(request, 'Invalid username or password.')
        else:
            form = UserLoginForm()

    context = {
        'ASSETS_ROOT': '/static/assets', # Tambahan path untuk asset static jika diperlukan
        'form': form,
    }
    return render(request, 'account/login.html', context)

# View untuk logout pengguna
def user_logout(request):
    # Logout pengguna
```

Fitur Registrasi, Login, dan Logout

```
logout(request)

# Hapus cookies kustom (misalnya,
'remember_me')
if 'remember_me' in request.COOKIES:
    response = redirect('login')
    response.delete_cookie('remember_me')
    return response

# Setelah logout, arahkan pengguna kembali
ke halaman login
return redirect('login')
```

5.4.4 URLS.py

```
# File: apps/authentication/urls.py

from django.urls import path
from .views import
home, user_register, user_login, user_logout

urlpatterns = [
    path('', home, name='home'),
    path('register/', user_register,
name='register'),
    path('login/', user_login, name='login'),
    path('logout/', user_logout, name='logout'),
]
```

5.5 Menghubungkan Halaman Home Dengan App Authentication

Pertama, kita perlu memastikan bahwa URL yang mengarah ke halaman **Login** dan **Registrasi** sudah didefinisikan dengan benar di file `urls.py` pada app **Authentication**. Mari kita tinjau bagaimana caranya.

Menautkan URL dari Halaman Home

Pada halaman **Home**, kita sudah memiliki tombol dengan label **Mulai Sekarang** yang akan mengarahkan pengguna ke halaman **Registrasi**. Kita juga bisa menambahkan tautan ke halaman **Login** untuk pengguna yang sudah memiliki akun.

Untuk itu, kita perlu memastikan bahwa URL untuk **Login** dan **Registrasi** telah ditentukan dengan benar di dalam file `urls.py` dari app **Authentication**.

Pada file template halaman **Home** yang berada di `apps/templates/authentication/home.html`, kita akan memperbarui atau menambahkan tautan ke halaman **Login**, dan **Registrasi**. , tambahkan tautan seperti berikut:

```
<!-- File:
apps/templates/authentication/home.html -->

{% extends 'home/base.html' %}

{% block title %}Home - Platform Bot{% endblock
%}
```

Fitur Registrasi, Login, dan Logout

```
{% block content %}
<div class="jumbotron text-center">
  <h1 class="display-4">Selamat Datang di
Platform Bot</h1>
  <p class="lead">Gerbang Anda untuk mengelola
dan berinteraksi dengan bot Anda.</p>
  <hr class="my-4">
  <p>Buat dan kelola bot Anda dengan mudah
melalui platform kami.</p>
  <a class="btn btn-primary btn-lg" href="{%
url 'register' %}" role="button">Mulai
Sekarang</a>
  <p class="mt-4">Sudah punya akun? <a
href="{% url 'login' %}">Masuk di sini</a></p>
<!-- Tautan ke halaman Login -->
</div>

<div class="row">
  <div class="col-md-4">
    <h3>Mudah Digunakan</h3>
    <p>Platform kami dirancang agar
sederhana dan intuitif, memungkinkan Anda
membuat dan mengelola bot dengan mudah.</p>
  </div>
  <div class="col-md-4">
    <h3>Fleksibel</h3>
    <p>Sesuaikan bot Anda sesuai kebutuhan
dengan pengaturan dan alat yang fleksibel dari
kami.</p>
  </div>
  <div class="col-md-4">
    <h3>Dapat Diskalakan</h3>
    <p>Apakah Anda mengelola satu bot atau
ratusan, platform kami dapat berkembang sesuai
kebutuhan bisnis Anda.</p>
  </div>
</div>
{% endblock %}
```

Fitur Registrasi, Login, dan Logout

Dalam template ini, kami menambahkan tautan `href="{% url 'login' %}"` yang mengarahkan pengguna ke halaman **Login** jika mereka sudah memiliki akun. Tautan ini menggunakan tag `url` dari Django untuk merujuk pada nama URL yang telah kita daftarkan di `urls.py`.

Terakhir, jika pengguna mengklik **Mulai Sekarang**, mereka akan diarahkan ke halaman **Registrasi**, dan jika mengklik **Masuk di sini**, mereka akan diarahkan ke halaman **Login**.

Pada file **base.html** kita juga akan menambahkan tautan untuk **register**, **login**, dan **logout** pada navbar nya.

```
<!-- File: apps/templates/home/base.html -->

<!DOCTYPE html>
<html>
<head>
  <title>{% block title %}Platform Bot{%
endblock %}</title>
  <!-- Menambahkan Bootstrap dari CDN -->
  <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3
.3/dist/css/bootstrap.min.css" rel="stylesheet"
integrity="sha384-
QWTKZyjpPEjISv5WaRU90FeRpok6YctnYmDr5pNlyT2bRjXh
0JMhjY6hW+ALEwIH" crossorigin="anonymous">

</head>
<body>
  <!-- Bagian header untuk navigasi -->
  <header>
```

Fitur Registrasi, Login, dan Logout

```
<nav class="navbar navbar-expand-lg
navbar-light bg-light">
  <a class="navbar-brand" href="{% url
'home' %}">Platform Bot</a>
  <button class="navbar-toggler"
type="button" data-toggle="collapse" data-
target="#navbarNav" aria-controls="navbarNav"
aria-expanded="false" aria-label="Toggle
navigation">
    <span class="navbar-toggler-
icon"></span>
  </button>
  <div class="collapse navbar-
collapse" id="navbarNav">
    <ul class="navbar-nav ms-auto">
      <li class="nav-item">
        <a class="nav-link"
href="{% url 'login' %}">Masuk</a>
      </li>
      <li class="nav-item">
        <a class="nav-link"
href="{% url 'register' %}">Daftar</a>
      </li>
      <li class="nav-item">
        <a class="nav-link"
href="{% url 'logout' %}">Keluar</a>
      </li>
    </ul>
  </div>
</nav>
</header>

    <main class="container
mt-5">
      {% block content %}
      {% endblock %}
    </main>
    <footer class="text-center py-4 bg-dark
text-white">
```

Fitur Registrasi, Login, dan Logout

```
<p>&copy; 2024 Platform Bot. All rights reserved.</p>
</footer>
<!-- Menyertakan JavaScript
Bootstrap -->
<script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.8/dist/umd/popper.min.js"
integrity="sha384-I7E8VVD/ismYTF4hNIPjVp/Zjvgyol6VFvRkX/vR+Vc4jQkC+hVqc2pM80Dewa9r"
crossorigin="anonymous"></script>
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.min.js" integrity="sha384-0pUGZvbkm6XF6gxjEnlmuGrJXVbNuzT9qBBavbLwCs0GabYfZo0T0to5eqruptLy"
crossorigin="anonymous"></script>
</body>
</html>
```

Dengan langkah-langkah ini, kita telah menghubungkan halaman **Home** dengan halaman **Login** dan **Registrasi** di app **Authentication**. Pengguna sekarang dapat dengan mudah melakukan navigasi dari halaman utama ke halaman yang sesuai dengan status mereka, apakah mereka ingin membuat akun baru atau masuk ke akun yang sudah ada.

Kesimpulan Bab

Pada Bab ini, kita telah membahas secara menyeluruh proses pembuatan fitur autentikasi pengguna yang meliputi registrasi, login, dan logout dalam proyek *platform_bot*. Dimulai dari pembuatan form dan template registrasi yang memungkinkan pengguna untuk membuat akun, kita juga mengimplementasikan validasi dan pengujian agar fitur berjalan sesuai harapan.

Selanjutnya, fitur login dikembangkan untuk memungkinkan pengguna masuk ke akun mereka dengan autentikasi yang kuat. Proses login ini dilengkapi dengan validasi yang ketat serta pengujian untuk memastikan bahwa login berjalan lancar, baik saat data benar maupun ketika terjadi kesalahan dalam input.

Fitur logout kemudian diimplementasikan untuk memberikan pengguna kemampuan keluar dari akun mereka dengan aman. Kami juga menambahkan pengaturan redirect setelah logout untuk meningkatkan pengalaman pengguna, dengan memastikan mereka diarahkan ke halaman yang relevan. Tidak hanya itu, pengujian terhadap fitur logout dilakukan untuk memastikan bahwa sesi pengguna dihapus secara benar setelah logout.

Pada **Sub-bab 5.1**, kita memulai dengan membuat model pengguna yang memungkinkan kita menyimpan dan mengelola data pengguna dengan baik. Sub-bab ini meliputi penjelasan tentang bagaimana mendefinisikan model user, menambahkan field yang relevan, dan menyesuaikan model dengan kebutuhan aplikasi.

Fitur Registrasi, Login, dan Logout

Sub-bab 5.2 berfokus pada pembuatan fitur login. Dengan membahas pembuatan form login, template yang sesuai, serta konfigurasi URL, kita dapat memahami bagaimana proses autentikasi pengguna dilakukan dan bagaimana memastikan tampilan login yang user-friendly.

Di **Sub-bab 5.3**, kami membahas fitur logout yang penting untuk memastikan pengguna dapat keluar dari sistem dengan aman. Langkah-langkah yang dijelaskan mencakup pengaturan view untuk logout, konfigurasi URL, serta penanganan redirect setelah logout untuk meningkatkan pengalaman pengguna.

Secara keseluruhan, bab ini memberikan panduan komprehensif untuk implementasi fitur autentikasi dasar dalam aplikasi Django, serta mengintegrasikannya dengan baik di dalam sistem yang lebih luas. Dengan memahami dan menerapkan teknik-teknik ini, Anda dapat membangun aplikasi yang lebih aman dan mudah digunakan, serta siap untuk integrasi dengan fitur-fitur tambahan yang mungkin diperlukan di masa depan.

Setiap langkah telah diiringi dengan praktik pengujian menggunakan Django's test framework, sehingga seluruh proses autentikasi ini dapat berjalan dengan baik tanpa kendala. Dengan fitur autentikasi yang lengkap ini, aplikasi *platform_bot* telah memiliki pondasi yang kuat dalam menangani keamanan dan kenyamanan pengguna.

BAB 6 - Membuat Halaman Dashboard User

Dalam pengembangan aplikasi *platform_bot*, dashboard berperan penting sebagai pusat kendali bagi pengguna yang telah terdaftar. Dashboard memungkinkan pengguna untuk mengelola akun, mengakses fitur bot, dan memantau kinerja serta statistik. Pada bagian ini, kita akan membahas bagaimana mendesain layout dashboard yang user-friendly serta memudahkan navigasi dan aksesibilitas pengguna.

6.1 Memahami Tujuan Dan Manfaat Dashboard

Dashboard adalah halaman utama yang akan diakses oleh pengguna setelah mereka berhasil login. Tujuan utama dari dashboard adalah memberikan gambaran umum mengenai data atau aktivitas penting yang relevan dengan pengguna. Dalam konteks pembuatan bot menggunakan Django, dashboard dapat digunakan untuk menampilkan informasi seperti statistik penggunaan bot, status bot yang sedang berjalan, log aktivitas terbaru, dan menu untuk mengelola bot.

Manfaat dari dashboard adalah:

1. **Pengelolaan yang Terpusat:** Dashboard memungkinkan pengguna untuk mengelola seluruh aspek dari bot mere-

Membuat Halaman Dashboard User

ka di satu tempat, tanpa harus berpindah-pindah halaman.

2. **Akses Informasi Cepat:** Pengguna dapat dengan cepat mengakses informasi penting mengenai bot, seperti status bot, statistik penggunaan, dan lainnya.
3. **Navigasi yang Mudah:** Dengan adanya sidebar atau menu navigasi, pengguna dapat dengan mudah berpindah ke fitur-fitur lain yang tersedia di dalam aplikasi.
4. **User Experience yang Lebih Baik:** Dashboard yang dirancang dengan baik dapat meningkatkan pengalaman pengguna, membuat aplikasi terasa lebih profesional dan mudah digunakan.

Kita akan mulai dengan langkah-langkah praktis untuk membuat halaman dashboard ini, dimulai dari model (jika diperlukan), view, template, hingga routing dan styling. Semua komponen ini akan membentuk halaman dashboard yang fungsional dan menarik.

Selanjutnya, kita akan membahas pembuatan model tambahan untuk informasi yang akan ditampilkan di dashboard, jika diperlukan.

6.2 Membuat Model Tambahan Untuk Informasi Dashboard

Pada tahap ini, kita akan membuat model tambahan untuk menyimpan informasi yang akan ditampilkan di dashboard. Dalam banyak kasus, dashboard akan menampilkan data yang sudah ada

Membuat Halaman Dashboard User

di dalam aplikasi, seperti profil pengguna, aktifitas penggunaan , dan lain-lain. Namun, jika ada informasi khusus yang perlu ditampilkan di dashboard dan tidak tercakup oleh model yang sudah ada, kita mungkin perlu menambahkan model baru.

Menentukan Kebutuhan Model Tambahan

Pertama-tama, kita perlu menentukan informasi apa saja yang akan ditampilkan di dashboard. Jika informasi tersebut sudah ada di model yang sudah kita buat sebelumnya, kita tidak perlu membuat model tambahan. Namun, jika ada data khusus yang tidak tercakup, seperti statistik harian, log aktivitas, atau notifikasi, kita perlu membuat model baru.

Contoh kasus:

- Kita akan menampilkan **log aktivitas** yang dilakukan oleh pengguna di dashboard, Kita perlu membuat model baru untuk menyimpan log tersebut.

Membuat Model log aktifitas

Disini kita ingin menyimpan log aktivitas pengguna, kita bisa membuat model baru seperti berikut:

```
# File: apps/dashboard/models.py
from django.contrib.auth.models import User
from django.db import models
from django.utils import timezone

class ActivityLog(models.Model):
```

Membuat Halaman Dashboard User

```
user = models.ForeignKey(User,
on_delete=models.CASCADE) # Relasi ke model
User
    action = models.CharField(max_length=255) #
Deskripsi aktivitas
    timestamp =
models.DateTimeField(default=timezone.now) #
Waktu aktivitas terjadi

    def __str__(self):
        return f"{self.user.username} -
{self.action} at {self.timestamp}"
```

Komentar di atas memberikan penjelasan tentang setiap elemen model:

- **user:** Menyimpan referensi ke pengguna yang melakukan aktivitas.
- **action:** Menyimpan deskripsi singkat tentang aktivitas yang dilakukan.
- **timestamp:** Menyimpan waktu kapan aktivitas tersebut terjadi.

Model ini dapat digunakan untuk mencatat semua aktivitas pengguna di dalam aplikasi, yang kemudian bisa ditampilkan di dashboard.

Menerapkan Migrasi untuk Model Baru

Setelah kita menambahkan model baru di `users/models.py`, kita perlu membuat dan menerapkan migrasi untuk memperbarui skema database.

Membuat Halaman Dashboard User

Di terminal, jalankan perintah berikut untuk membuat migrasi:

```
$ python manage.py makemigrations
```

Perintah ini akan membuat file migrasi untuk model `ActivityLog`.

Selanjutnya, terapkan migrasi tersebut ke database dengan perintah:

```
$ python manage.py migrate
```

Setelah migrasi selesai, skema database akan diperbarui, dan kita siap menggunakan model `ActivityLog` untuk mencatat aktivitas pengguna.

Mengintegrasikan Model dengan Dashboard

Model `ActivityLog` yang baru saja kita buat dapat diintegrasikan dengan dashboard untuk menampilkan aktivitas terbaru pengguna. Di bagian berikutnya, kita akan membuat view dan template untuk menampilkan informasi ini di halaman dashboard.

Dengan langkah-langkah ini, kita telah berhasil membuat model tambahan yang diperlukan untuk menampilkan informasi di dashboard. Sekarang lanjut ke pembuatan view dan template untuk dashboard.

6.3 Membuat View Dashboard

Setelah mendesain layout dashboard yang mencakup navigasi dan aksesibilitas, langkah berikutnya adalah menampilkan data yang relevan di dalamnya. View untuk dashboard bertanggung jawab dalam mengambil dan menampilkan data yang sesuai bagi pengguna. Selain itu, kita juga perlu memastikan bahwa hanya pengguna yang memiliki hak akses yang dapat melihat dan mengelola informasi di dashboard.

View adalah komponen penting dalam arsitektur aplikasi berbasis Django, yang berfungsi sebagai penghubung antara logika bisnis dan tampilan (template). Dalam konteks dashboard pengguna, view bertugas untuk mengambil data pengguna yang login, seperti profil, aktivitas, dan statistik bot, lalu merender template yang menampilkannya.

Berikut ini adalah cara membuat view untuk dashboard yang menampilkan data bot dan mengatur hak akses user:

```
# File: apps/dashboard/views.py
from django.shortcuts import render
from django.contrib.auth.decorators import login_required
from apps.authentication.models import UserProfile
from apps.dashboard.models import ActivityLog

@login_required
def dashboard(request):
    # Ambil profil pengguna yang sedang login
```


Membuat Halaman Dashboard User

```
    user_profile =
UserProfile.objects.filter(user=request.user).fi
rst()

    # Ambil aktivitas log pengguna yang sedang
login
    activity_logs =
ActivityLog.objects.filter(user=request.user).or
der_by('-timestamp')

    # Data yang akan diteruskan ke template
context = {
        'ASSETS_ROOT': '/static/assets',
        'user_profile': user_profile,
        'activity_logs': activity_logs,
    }

    # Merender template dashboard.html dengan
data yang diambil
    return render(request,
'dashboard/dashboard.html', context)
```

Pada view dashboard, kita menggunakan dekorator `@login_required` untuk memastikan hanya pengguna yang telah login yang dapat mengakses halaman dashboard. Jika pengguna yang tidak login mencoba mengakses halaman ini, mereka akan diarahkan ke halaman login.

Selanjutnya, kita mengambil data dari model `UserProfile` dan `ActivityLog`. `UserProfile` berisi informasi pribadi pengguna yang akan ditampilkan di dashboard, seperti nama, foto profil, dan detail lainnya. Sedangkan `ActivityLog` menyimpan catatan aktivitas pengguna, misalnya interaksi dengan bot yang sudah dibuat. Kita menggunakan metode `filter()`

Membuat Halaman Dashboard User

pada model `ActivityLog` untuk mendapatkan semua aktivitas yang terkait dengan pengguna yang sedang login, dan mengurutkannya berdasarkan `timestamp` dari yang terbaru.

Penjelasan Hak Akses

Hak akses diatur dengan menggunakan dekorator `@login_required` yang memastikan hanya pengguna yang login yang bisa melihat halaman dashboard. Ini sangat penting untuk menjaga privasi dan keamanan data pengguna. Dengan menerapkan login guard seperti ini, aplikasi tidak akan mengizinkan akses tanpa autentikasi yang valid.

Setelah data diambil, kita meneruskannya ke template dengan membuat konteks yang berisi `user_profile` dan `activity_logs`. Template `dashboard.html` kemudian akan bertanggung jawab untuk menampilkan data tersebut dalam bentuk yang mudah dimengerti oleh pengguna.

Testing Hak Akses

Kita bisa menguji apakah fitur ini berfungsi dengan benar dengan mencoba mengakses URL dashboard saat tidak login. Jika pengguna diarahkan ke halaman login, berarti hak akses telah diterapkan dengan baik.

6.4 Membuat Template Halaman Dashboard

Langkah selanjutnya dalam membuat dashboard adalah menentukan elemen-elemen utama yang akan ditampilkan. Pada aplikasi *platform_bot*.

beberapa fitur yang akan tersedia di dashboard antara lain:

- **Log aktifitas:** Menampilkan informasi log aktifitas user seperti user mengedit profile mereka.
- **Navigasi Bot:** Menu untuk mengelola bot, seperti membuat bot baru, mengedit bot yang sudah ada, atau memantau bot yang berjalan.
- **Pengaturan:** Akses ke pengaturan akun dan preferensi pengguna.
- **Logout:** Tombol untuk keluar dari sistem.

Setelah elemen-elemen ini ditentukan, langkah selanjutnya adalah merancang layout yang jelas dan mudah dinavigasi. Desain dashboard harus intuitif, memastikan bahwa pengguna dapat dengan cepat memahami fungsionalitas setiap fitur yang ada.

Misalnya, di bagian atas (header) dashboard dapat ditampilkan nama pengguna beserta tombol dropdown untuk pengaturan dan logout. Di bagian kiri, dapat diletakkan menu navigasi vertikal untuk fitur-fitur utama seperti **Dashboard Utama**, **Pengelolaan Bot**, dan **Statistik**. Sementara di bagian kanan (content area) akan menjadi tempat untuk menampilkan konten spesifik dari fitur yang dipilih.

Membuat Halaman Dashboard User

6.4.1 Membuat Template base.html

Pertama-tama, kita buat template `base.html` yang akan menjadi dasar dari semua halaman di dashboard. Template ini akan mengatur struktur dasar HTML, termasuk header, footer, dan area konten utama.

Buat file `base.html` di direktori `apps/templates/dashboard/` dengan isi sebagai berikut:

```
<!-- File: apps/templates/dashboard/base.html -->
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Dashboard | Platform Bot</title>
  <!-- Menambahkan Bootstrap dari CDN -->
  <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3
.3/dist/css/bootstrap.min.css" rel="stylesheet"
integrity="sha384-
QWTKZyjpPEjISv5WaRU90FeRpok6YctnYmDr5pNlyT2bRjXh
0JMHjY6hw+ALEwIH" crossorigin="anonymous">
  <link
href="https://cdn.jsdelivr.net/npm/bootstrap-
icons@1.10.5/font/bootstrap-icons.css"
rel="stylesheet">
  <link
href="https://stackpath.bootstrapcdn.com/bootstr
ap/5.3.0/css/bootstrap.min.css"
rel="stylesheet">
```

Membuat Halaman Dashboard User

```
</head>
<body>
  <nav class="navbar navbar-expand-lg navbar-
light bg-light">
    <a class="navbar-brand" href="{% url
'home' %}">Platform Bot</a>
    <div class="collapse navbar-collapse"
id="navbarNav">
      <ul class="navbar-nav ms-auto">
        <li class="nav-item">
          <a class="nav-link" href="{%
url 'logout' %}">Logout</a>
        </li>
      </ul>
    </div>
  </nav>
  <div class="container-fluid">
    <div class="row">
      <div class="col-md-3">
        <!-- Sidebar -->
        <div class="list-group">
          <a href="{% url 'dashboard'
%}" class="list-group-item list-group-item-
action">Dashboard</a>
          <a href="#" class="list-
group-item list-group-item-action">Profile</a>
          <a href="#" class="list-
group-item list-group-item-action">Create
bot</a>
          <a href="#" class="list-
group-item list-group-item-action">Manage
Bot</a>
        </div>
      </div>
      <div class="col-md-9">
        <!-- Main Content -->
        {% block content %}
        {% endblock %}
      </div>
    </div>
  </div>
</body>
</html>
```

Membuat Halaman Dashboard User

```
        </div>
    </div>
</div>

<!-- Menyertakan JavaScript Bootstrap -->
<script
src="https://cdn.jsdelivr.net/npm/@popperjs/core
@2.11.8/dist/umd/popper.min.js"
integrity="sha384-I7E8VVD/ismYTF4hNIPjVp/Zjvgyol
6VFvRkX/vR+Vc4jQkC+hVqc2pM80Dewa9r"
crossorigin="anonymous"></script>
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.
3/dist/js/bootstrap.min.js" integrity="sha384-
0pUGZvbkm6XF6gxjEnlmuGrJXVbNuzT9qBBavbLwCsOGabYf
Zo0T0to5eqruptLy"
crossorigin="anonymous"></script>
</body>
</html>
```

Desain ini memanfaatkan struktur standar yang memisahkan header, sidebar untuk navigasi, dan konten utama di area tengah. Template ini dapat digunakan sebagai basis untuk semua halaman dalam dashboard, memanfaatkan konsep *Django Template Inheritance* untuk menjaga konsistensi desain antar halaman.

Selain itu, penting untuk memastikan bahwa layout dashboard responsif, sehingga dapat diakses dengan baik dari perangkat seluler maupun desktop.

Membuat Halaman Dashboard User

6.4.2 Membuat halaman dashboard.html

Selanjutnya, kita buat template `dashboard.html` yang akan menggunakan `base.html` sebagai dasar dan menampilkan data yang telah kita siapkan di view `dashboard`.

Kita juga akan menambahkan kode untuk menampilkan pesan error dan berhasil jika user login.

```
<!-- Menampilkan pesan jika ada error atau sukses -->
    {% if messages %}
        <div class="messages">
            {% for message in messages %}
                <div class="alert
{{ message.tags }}">{{ message }}</div>
            {% endfor %}
        </div>
    {% endif %}
```

Dengan kode tersebut maka di dashboard ada ada pesan untuk user jika berhasil login.

Buat file `dashboard.html` di direktori `apps/templates/dashboard/` dengan isi sebagai berikut:

```
<!-- File:
apps/templates/dashboard/dashboard.html -->
{% extends 'dashboard/base.html' %}

{% block content %}
<h1 class="fw=bold">Dashboard</h1>
```

Membuat Halaman Dashboard User

```
<h2>Welcome to Your Dashboard,
{{ user_profile.user.username }}</h2>
<p>Di Dashboard ini kita sudah menampilkan log
aktivitas dan user profile.Selanjutnya kita akan
menambahkan lagi fitur lainnya.</p>

<!-- Menampilkan pesan jika ada error atau
sukses -->
    {% if messages %}
        <div class="messages">
            {% for message in messages
%}
                <div class="alert
{{ message.tags }}">{{ message }}</div>
            {% endfor %}
        </div>
    {% endif %}

<h3>Log Aktivitas</h3>
<table class="table table-bordered">
    <thead>
        <tr>
            <th>Timestamp</th>
            <th>Action</th>
        </tr>
    </thead>
    <tbody>
        {% for log in activity_logs %}
            <tr>
                <td>{{ log.timestamp }}</td>
                <td>{{ log.action }}</td>
            </tr>
            {% empty %}
                <tr>
                    <td colspan="2">Tidak ada aktivitas
yang tercatat.</td>
                </tr>
            {% endfor %}
```


Membuat Halaman Dashboard User

```
        </tbody>
    </table>

    <h2>Profil Pengguna</h2>
    <p>Nama Pengguna: {{ user_profile.user.username }}</p>
    <p>Kota: {{ user_profile.city }}</p>
    <p>Alamat: {{ user_profile.address }}</p>
    <p>Negara: {{ user_profile.country }}</p>
    <p>Kode Pos: {{ user_profile.postal_code }}</p>
    <p>Tentang: {{ user_profile.about }}</p>
    <p>Perusahaan: {{ user_profile.company }}</p>

    {% if user_profile.foto_profile %}
        
    {% else %}
        <p>Tidak ada foto profil.</p>
    {% endif %}

{% endblock %}
```

Penjelasan kode:

- Template ini mewarisi (extends) `base.html`, sehingga otomatis mendapatkan layout dari `base.html`.
- Bagian `{% block content %}` diisi dengan konten spesifik untuk halaman dashboard, termasuk salam selamat datang yang menampilkan nama pengguna dan tabel yang berisi log aktivitas pengguna.

6.4.3 Membuat Sidebar dengan Ikon dan Responsif Menggunakan Bootstrap

Selanjutnya kita akan memodifikasi sidebar pada halaman dashboard agar lebih menarik dengan menambahkan ikon dari **Boots-**

Membuat Halaman Dashboard User

trap Icons. Selain itu, kita juga akan memastikan sidebar tersebut responsif agar dapat diakses dengan baik pada perangkat mobile. Sidebar ini juga akan memiliki tombol toggle untuk perangkat kecil, di mana pengguna bisa menampilkan atau menyembunyikan sidebar sesuai kebutuhan.

Struktur HTML Sidebar dengan Ikon

Untuk menambahkan ikon di sidebar, kita menggunakan **Bootstrap Icons**. Ikon ini ditempatkan di dalam setiap elemen link pada sidebar, diikuti oleh teks navigasi. Dengan pendekatan ini, setiap item di sidebar akan memiliki ikon yang relevan dengan fungsi dari item tersebut.

```
<nav id="sidebar" class="col-md-3 col-lg-2 d-md-block bg-light sidebar">
  <div class="position-sticky">
    <ul class="nav flex-column">
      <li class="nav-item">
        <a class="nav-link active"
href="{% url 'dashboard' %}">
          <i class="bi bi-house-
door"></i> Dashboard
        </a>
      </li>
      <li class="nav-item">
        <a class="nav-link" href="{% url
'profile' %}">
          <i class="bi bi-person"></i>
Profile
        </a>
      </li>
      <li class="nav-item">
        <a class="nav-link" href="#">
          <i class="bi bi-plus"></i>
Create Bot
```

Membuat Halaman Dashboard User

```
        </a>
      </li>
      <li class="nav-item">
        <a class="nav-link" href="#">
          <i class="bi bi-gear"></i>
        </a>
      </li>
    </ul>
  </div>
</nav>
```

Pada kode di atas, ikon-ikon dari **Bootstrap Icons** ditambahkan menggunakan elemen `<i>` dengan class `bi`. Misalnya, ikon rumah untuk dashboard (`bi-house-door`), ikon orang untuk profil (`bi-person`), dan sebagainya.

Menambahkan Sidebar Toggle untuk Responsivitas

Sidebar ini harus dapat berfungsi dengan baik pada berbagai ukuran layar, termasuk perangkat mobile. Untuk itu, kita tambahkan sebuah tombol toggle yang hanya muncul pada perangkat kecil (layar di bawah ukuran **md** atau medium). Tombol ini akan memungkinkan pengguna untuk menampilkan atau menyembunyikan sidebar dengan lebih mudah.

Berikut adalah penambahan tombol toggle:

```
<div class="col-md-3">
  <button class="btn btn-primary d-md-none"
    type="button" data-bs-toggle="collapse" data-bs-
    target="#sidebarMenu" aria-expanded="false"
    aria-controls="sidebarMenu">
    Menu
```

Membuat Halaman Dashboard User

```
        </button>
        <div class="collapse d-md-block"
id="sidebarMenu">
            <div class="list-group">
                <a href="{% url 'dashboard' %}"
class="list-group-item list-group-item-action">
                    <i class="bi bi-house-door"></i>
Dashboard
                </a>
                <a href="#" class="list-group-item
list-group-item-action">
                    <i class="bi bi-person"></i>
Profile
                </a>
                <a href="#" class="list-group-item
list-group-item-action">
                    <i class="bi bi-plus"></i>
Create Bot
                </a>
                <a href="#" class="list-group-item
list-group-item-action">
                    <i class="bi bi-gear"></i>
Manage Bots
            </div>
        </div>
    </div>
</div>
```

Pada kode ini, elemen `<button>` dengan class `d-md-none` digunakan untuk menampilkan tombol hanya pada layar yang lebih kecil. Tombol ini akan mengontrol **collapse** dari sidebar dengan target `#sidebarMenu`. Ketika tombol ditekan, sidebar akan muncul atau menghilang berdasarkan status collapse.

Memperbaiki Kode base.html untuk Responsivitas Penuh

Membuat Halaman Dashboard User

Sekarang, kita akan menggabungkan kedua bagian tersebut ke dalam sebuah kode lengkap untuk membuat sidebar dengan ikon yang responsif.

```
<!-- File: apps/templates/dashboard/base.html -->
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Dashboard | Platform Bot</title>
  <!-- Menambahkan Bootstrap dan Bootstrap Icons dari CDN -->
  <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet">
  <link
href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.10.5/font/bootstrap-icons.css"
rel="stylesheet">
</head>
<body>
  <!-- Navbar -->
  <nav class="navbar navbar-expand-lg navbar-light bg-light">
    <div class="container-fluid">
      <a class="navbar-brand" href="{% url 'home' %}">Platform Bot</a>
      <button class="navbar-toggler"
type="button" data-bs-toggle="collapse" data-bs-target="#sidebarMenu" aria-controls="sidebarMenu" aria-expanded="false"
aria-label="Toggle sidebar">
        <span class="navbar-toggler-icon"></span>
      </button>
    </div>
  </nav>
</body>
</html>
```

Membuat Halaman Dashboard User

```
        </div>
    </nav>

    <div class="container-fluid">
        <div class="row">
            <!-- Sidebar -->
            <div class="col-md-3">
                <div class="collapse d-md-block"
id="sidebarMenu">
                    <nav id="sidebar" class="bg-
light sidebar">
                        <div class="position-
sticky">
                            <ul class="nav flex-
column">
                                <li class="nav-
item">
                                    <a
class="nav-link active" href="{% url 'dashboard'
%}">
                                        <i
class="bi bi-house-door"></i> Dashboard
                                    </a>
                                </li>
                                <li class="nav-
item">
                                    <a
class="nav-link" href="">
                                        <i
class="bi bi-person"></i> Profile
                                    </a>
                                </li>
                                <li class="nav-
item">
                                    <a
class="nav-link" href="#">
                                        <i
class="bi bi-plus"></i> Create Bot
                                    </a>
```

Membuat Halaman Dashboard User

```

</li>
<li class="nav-
item">
    <a
class="nav-link" href="#">
        <i
class="bi bi-gear"></i> Manage Bots
    </a>
</li>
<!-- Tombol
Logout di Sidebar -->
<li class="nav-
item">
    <a
class="nav-link text-danger" href="{% url
'logout' %}">
        <i
class="bi bi-box-arrow-right"></i> Logout
    </a>
</li>
</ul>
</div>
</nav>
</div>
</div>
</div>

<!-- Konten Utama -->
<div class="col-md-9">
    {% block content %}
    {% endblock %}
</div>
</div>

<!-- Menyertakan JavaScript Bootstrap -->
<script
src="https://cdn.jsdelivr.net/npm/@popperjs/core
@2.11.8/dist/umd/popper.min.js"></script>

```

Membuat Halaman Dashboard User

```
<script  
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.  
3/dist/js/bootstrap.min.js"></script>  
</body>  
</html>
```

Penjelasan Kode

- **Navbar:** Tombol navbar-toggler dengan data-bs-toggle="collapse" dan data-bs-target="#sidebarMenu" untuk memicu sidebar di layar kecil.
- **Sidebar dengan Ikon:** Sidebar berisi beberapa item navigasi yang masing-masing memiliki ikon. Untuk itu, kita menggunakan elemen `<i>` dari **Bootstrap Icons** yang relevan.
- **Responsivitas Sidebar:** Sidebar ini memiliki tombol toggle yang hanya muncul di layar kecil. Dengan tombol ini, pengguna dapat menampilkan atau menyembunyikan sidebar sesuai kebutuhan pada perangkat mobile.
- **Collapse Sidebar:** Pada layar besar, sidebar akan tetap terlihat secara default. Namun, pada layar kecil, sidebar akan tersembunyi dan hanya muncul jika tombol toggle ditekan.

Dengan demikian, sidebar yang kita buat ini tidak hanya terlihat lebih interaktif dengan ikon, tetapi juga lebih ramah pengguna, khususnya pada perangkat kecil.

6.5 Menyiapkan URL Routing Untuk Dashboard

Sekarang kita perlu menyiapkan URL routing untuk mengakses halaman dashboard. Kita akan menambahkan URL pattern di `urls.py` pada aplikasi dashboard.

Buka atau buat file `dashboard/urls.py` dan tambahkan kode berikut:

```
# File: apps/dashboard/urls.py
from django.urls import path
from .views import dashboard

urlpatterns = [
    path('dashboard/', dashboard,
        name='dashboard'),
]
```

Penjelasan kode:

- URL pattern `dashboard/` akan memetakan ke view `dashboard`, sehingga ketika pengguna mengakses `http://localhost:8000/dashboard/`, mereka akan diarahkan ke halaman dashboard.

Dengan menambahkan path ini, pengguna dapat mengakses halaman dashboard melalui URL `/dashboard/`. View `dashboard` akan dijalankan setiap kali URL tersebut diakses, dan pengguna akan keluar dari sesi mereka.

Membuat Halaman Dashboard User

Selanjutnya, kita perlu memastikan bahwa URL aplikasi dashboard termasuk dalam routing utama proyek. Buka file `platform_bot/urls.py` dan tambahkan konfigurasi berikut:

```
# File: platform_bot/urls.py

from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('apps.dashboard.urls')), #
    Menyertakan URL aplikasi dashboard
]
```

Dengan konfigurasi ini, halaman dashboard akan ditampilkan ketika pengguna mengunjungi URL root dari aplikasi.

6.6 Mengatur Redirect Otomatis Ke Dashboard Setelah Login

Untuk mengatasi masalah ketika pengguna mencoba mengakses halaman yang memerlukan autentikasi (misalnya halaman dashboard) dalam keadaan belum login, kamu bisa mengarahkan mereka ke halaman login atau halaman lainnya dengan lebih baik, daripada menampilkan error 404. Kamu bisa menggunakan mekanisme Django yang otomatis mengarahkan pengguna yang tidak autentikasi ke halaman login yang benar.

Berikut adalah cara untuk memperbaikinya:

Atur URL Login di `settings.py`

Membuat Halaman Dashboard User

Django memiliki pengaturan khusus yang dapat mengarahkan pengguna yang belum login ke halaman login. Kamu dapat memastikan bahwa URL login kamu sudah diset dengan benar dalam file `settings.py`.

Buka file `settings.py` dan tambahkan atau sesuaikan baris berikut:

```
# settings.py
LOGIN_URL = '/login/'
```

Dengan menambahkan `LOGIN_URL`, Django akan mengarahkan pengguna yang tidak autentikasi ke halaman login yang benar ketika mereka mencoba mengakses halaman yang membutuhkan login.

Redirect Setelah Login

Setelah login berhasil, pengguna biasanya akan diarahkan kembali ke halaman yang sebelumnya mereka coba akses. Django secara otomatis mengelola ini melalui parameter `next`. Misalnya, jika pengguna mencoba mengakses `/dashboard/`, mereka akan diarahkan ke `/login/?next=/dashboard/`, dan setelah login berhasil, mereka akan kembali ke dashboard.

Jika kamu ingin mengatur halaman redirect setelah login, kamu bisa menggunakan pengaturan berikut di `settings.py`:

```
# settings.py
```

Membuat Halaman Dashboard User

```
LOGIN_REDIRECT_URL = '/dashboard/' # Halaman  
yang akan diarahkan setelah login berhasil
```

Dengan pengaturan ini, setiap kali pengguna berhasil login, mereka akan diarahkan ke halaman dashboard atau halaman lainnya yang telah kamu tentukan.

Langkah-langkah di atas akan memastikan bahwa pengguna yang belum login diarahkan ke halaman login yang benar saat mencoba mengakses halaman yang memerlukan autentikasi. Dengan menggunakan `login_required` dan mengatur URL login dengan benar, pengalaman pengguna menjadi lebih baik dan error seperti 404 dapat dihindari.

Perbarui views di `apps/authentication`

Kita juga harus memperbaiki kode di views agar ketika user sudah register dan login maka akan redirect ke halaman dashboard.

Buka file views di `apps/authentication` dan ubah kode rerurn redirect di fungsi `user_register` dan `user_login` nya menjadi dashboard.

```
# File: apps/authentication/views.py  
# View untuk menangani registrasi pengguna baru  
def user_register(request):  
  
    # ...  
  
    # Redirect ke halaman dashboard  
    setelah registrasi berhasil  
    return redirect('dashboard')
```

Membuat Halaman Dashboard User

```
# ...  
  
# View untuk login pengguna  
def user_login(request):  
    # ...  
    return redirect('dashboard') #  
    Mengarahkan pengguna setelah login berhasil  
  
# ...
```

Langkah-langkah di atas akan memastikan bahwa pengguna yang sudah melakukan registrasi atau login akan diarahkan ke halaman dashboard.

Kesimpulan Bab

Pada Bab ini telah menjelaskan secara mendetail proses pembuatan halaman dashboard untuk pengguna dalam aplikasi Django. Fokus utama bab ini adalah untuk memberikan panduan lengkap tentang bagaimana mengembangkan dan mengelola dashboard yang efektif dan informatif bagi pengguna.

Sub-bab 6.1 dimulai dengan pemahaman tujuan dan manfaat dari dashboard user, yang membantu kita menetapkan kerangka kerja dan fungsionalitas yang diperlukan. Memahami kebutuhan pengguna dan tujuan dari dashboard merupakan langkah penting untuk memastikan dashboard yang efektif dan bermanfaat.

Dalam **Sub-bab 6.1.2**, kita membahas pembuatan model tambahan untuk menyimpan informasi yang relevan dengan dashboard. Model ini memungkinkan kita untuk menyimpan data yang dibutuhkan untuk menampilkan informasi penting secara real-time kepada pengguna.

Sub-bab 6.1.3 menjelaskan pembuatan view dashboard, yang merupakan bagian penting dalam menghubungkan model data dengan tampilan frontend. View ini bertanggung jawab untuk mengolah data dan menyediakannya ke template dashboard.

Pada **Sub-bab 6.1.4**, kami membahas pembuatan template untuk halaman dashboard. Template ini dirancang untuk menyajikan data dengan cara yang intuitif dan menarik, memastikan pengalaman pengguna yang baik dan interaktif.

Membuat Halaman Dashboard User

Sub-bab 6.1.5 menguraikan penyiapan URL routing untuk dashboard, yang penting untuk mengakses halaman dashboard dengan mudah melalui URL yang sesuai. Pengaturan ini memastikan bahwa dashboard dapat diakses dengan cepat dan tanpa kesulitan.

Akhirnya, **Sub-bab 5.1.6** membahas pengaturan redirect otomatis ke dashboard setelah login. Fitur ini meningkatkan pengalaman pengguna dengan mengarahkan pengguna langsung ke dashboard mereka setelah berhasil masuk, membuat proses navigasi lebih lancar.

Secara keseluruhan, bab ini memberikan panduan yang lengkap untuk pembuatan dan pengelolaan halaman dashboard user dalam aplikasi Django, memfokuskan pada bagaimana menyajikan data secara efektif dan meningkatkan interaksi pengguna. Dengan penerapan teknik-teknik ini, Anda dapat membangun dashboard yang tidak hanya fungsional tetapi juga memberikan nilai tambah bagi pengguna aplikasi Anda.

BAB 7 - Manajemen Profil Pengguna

Dalam bab ini kita akan membuat manajemen profil pengguna yang memungkinkan pengguna untuk melihat dan mengedit informasi profil mereka. Halaman ini akan mencakup detail seperti nama pengguna, email, nama depan, nama belakang, alamat, kota, negara, kode pos, perusahaan, tentang saya, dan foto profil.

7.1 Membuat Halaman Profile Pengguna

Model `UserProfile` yang telah kita buat di bab sebelumnya sudah memadai untuk menyimpan informasi profil pengguna. Model ini menyimpan data penting seperti nama kota, alamat, negara, kode pos, tentang pengguna, perusahaan, dan foto profil.

Dibagian ini kita akan membuat halaman untuk menampilkan informasi user yang ada di models `UserProfile`

7.1.1 Membuat View untuk Halaman Profil Pengguna

Pertama, kita akan menyiapkan tampilan yang akan mengontrol logika halaman profil pengguna. Tampilan ini akan menangani

Manajemen Profil Pengguna

pengambilan data profil pengguna, serta menampilkan profile pengguna.

Buka file `apps/users/views.py` dan tambahkan kode berikut:

```
# File: apps/users/views.py

from django.shortcuts import render,
get_object_or_404
from django.contrib.auth.decorators import
login_required
from django.contrib.auth.models import User
from apps.authentication.models import
UserProfile

@login_required
def profile(request):
    # Ambil user yang sedang login
    user = request.user

    # Ambil UserProfile terkait dengan user
    user_profile =
get_object_or_404(UserProfile, user=user)

    # Kirim data user dan user_profile ke
template
    context = {
        'user': user,
        'user_profile': user_profile,
    }

    return render(request, 'users/profile.html',
context)
```

Manajemen Profil Pengguna

Pada kode di atas, fungsi `profile` mengelola tampilan halaman profil.

7.1.2 Membuat Template Halaman Profil

Sekarang, kita perlu membuat template HTML untuk halaman profil pengguna.

Template untuk halaman profil harus mencakup form yang menampilkan informasi pengguna, seperti username, email, first name, last name, dan informasi profil lainnya seperti alamat, perusahaan, dan gambar profil. Berikut adalah contoh template untuk halaman profil pengguna:

Buat file bernama `profile.html` di dalam direktori `apps/templates/dashboard/` dan tambahkan kode berikut:

```
<!-- apps/templates/users/profile.html -->

{% extends "dashboard/base.html" %}

{% block content %}
<div class="container">
  <h2>User Profile</h2>
  <div class="row">
    <div class="col-md-8">
      <p><strong>Username:</strong>
      {{ user.username }}</p>
      <p><strong>Email:</strong>
      {{ user.email }}</p>
    </div>
  </div>
</div>
{% endblock %}
```

Manajemen Profil Pengguna

```
        <p><strong>First Name:</strong>
{{ user.first_name }}</p>
        <p><strong>Last Name:</strong>
{{ user.last_name }}</p>
        <p><strong>Address:</strong>
{{ user_profile.address }}</p>
        <p><strong>City:</strong>
{{ user_profile.city }}</p>
        <p><strong>Country:</strong>
{{ user_profile.country }}</p>
        <p><strong>Postal Code:</strong>
{{ user_profile.postal_code }}</p>
        <p><strong>Company:</strong>
{{ user_profile.company }}</p>
        <p><strong>About:</strong>
{{ user_profile.about }}</p>
    </div>

<div
class="col-md-4">
    {% if user_profile.foto_profile %}
        
    {% else %}
        <img src="" alt="Profile
Picture" class="img-thumbnail">
    {% endif %}
    </div>
</div>
{% endblock %}
```

Penjelasan Kode:

- Template ini menggunakan Bootstrap untuk membuat form yang responsif dan tata letak yang bersih.

Manajemen Profil Pengguna

- Ada form untuk mengedit informasi pengguna seperti username, email, first name, last name, dan informasi profil lainnya, termasuk foto profil.

7.1.3 Menambahkan URL Routing

Terakhir, tambahkan URL untuk mengakses halaman profil pengguna. Buka file `users/urls.py` dan tambahkan URL berikut:

```
# File: apps/users/urls.py

from django.urls import path
from . import views

urlpatterns = [
    path('profile/', views.profile,
         name='profile'),
]
```

Sekarang, pengguna dapat mengakses halaman profil mereka di `/profile/` dan melihat informasi profil mereka.

Dengan menyelesaikan langkah-langkah di atas, Anda telah berhasil membuat halaman profil pengguna yang memungkinkan pengguna untuk melihat informasi mereka.

Selanjutnya, kita perlu memastikan bahwa URL aplikasi `users` termasuk dalam routing utama proyek. Buka file `platform_bot/urls.py` dan tambahkan konfigurasi berikut:

Manajemen Profil Pengguna

```
# File: platform_bot/urls.py

from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('apps.users.urls')), #
    Menyertakan URL aplikasi users
]
```

Dengan konfigurasi ini, halaman users akan ditampilkan ketika pengguna mengunjungi URL root dari aplikasi.

Halaman ini meliputi detail seperti nama pengguna, email, nama depan, nama belakang, alamat, kota, negara, kode pos, perusahaan, tentang saya, dan foto profil.

7.2 Membuat Fitur Pengeditan Profile

Selanjutnya kita akan membuat fitur yang memungkinkan pengguna bisa mengedit informasi mereka langsung di halaman profile.

7.2.1 Membuat Form untuk mengedit Profil Pengguna

Kita sudah membuat view profile di `views.py` yang menampilkan informasi user, selanjutnya kita akan memperbaiki views ini agar bisa menangani pengeditan dan penampilan profil pengguna.

Manajemen Profil Pengguna

Untuk memungkinkan pengguna mengedit informasi mereka langsung di halaman profil tanpa mengarahkan ke halaman terpisah, kita sudah berada di jalur yang benar dengan menggunakan `EditProfileForm`. Form ini menggabungkan data dari model `User` dan `UserProfile` sehingga pengguna dapat memperbarui informasi akun dan profil mereka di halaman yang sama.

Ada beberapa penyesuaian yang bisa dilakukan pada `EditProfileForm` untuk memastikan semua data diproses dengan benar. Misalnya, saat menyimpan data, pastikan bahwa perubahan pada model `User` juga disimpan.

Berikut adalah penyesuaian untuk `EditProfileForm`:

```
# File: apps/users/forms.py

from django import forms
from django.contrib.auth.models import User
from apps.authentication.models import UserProfile

class EditProfileForm(forms.ModelForm):
    # Field dari model User
    email = forms.EmailField(required=True,
        widget=forms.EmailInput(attrs={'class': 'form-control', 'placeholder': 'Email'}))
    first_name = forms.CharField(max_length=100,
        widget=forms.TextInput(attrs={'class': 'form-control', 'placeholder': 'First Name'}))
    last_name = forms.CharField(max_length=100,
        widget=forms.TextInput(attrs={'class': 'form-control', 'placeholder': 'Last Name'}))
```

Manajemen Profil Pengguna

```
# Field dari model UserProfile
address = forms.CharField(max_length=255,
required=False,
widget=forms.TextInput(attrs={'class': 'form-
control', 'placeholder': 'Address'}))
city = forms.CharField(max_length=100,
required=False,
widget=forms.TextInput(attrs={'class': 'form-
control', 'placeholder': 'City'}))
country = forms.CharField(max_length=100,
required=False,
widget=forms.TextInput(attrs={'class': 'form-
control', 'placeholder': 'Country'}))
postal_code = forms.CharField(max_length=20,
required=False,
widget=forms.TextInput(attrs={'class': 'form-
control', 'placeholder': 'Postal Code'}))
company = forms.CharField(max_length=255,
required=False,
widget=forms.TextInput(attrs={'class': 'form-
control', 'placeholder': 'Company'}))
about = forms.CharField(max_length=255,
required=False,
widget=forms.Textarea(attrs={'class': 'form-
control', 'placeholder': 'About me', 'rows':
4}))
foto_profile =
forms.ImageField(required=False,
widget=forms.FileInput(attrs={'class': 'form-
control-file'}))

class Meta:
    model = UserProfile
    fields = ['city', 'address', 'country',
'postal_code', 'about', 'company',
'foto_profile']

def __init__(self, *args, **kwargs):
```

Manajemen Profil Pengguna

```
        user = kwargs.pop('user', None) # Ambil
data user
        super(EditProfileForm,
self).__init__(*args, **kwargs)

        if user:
            # Inisialisasi field dari model User
            self.fields['username'] =
forms.CharField(
                initial=user.username,
widget=forms.TextInput(attrs={'class': 'form-
control', 'readonly': 'readonly'})
            )
            self.fields['email'].initial =
user.email
            self.fields['first_name'].initial =
user.first_name
            self.fields['last_name'].initial =
user.last_name

        def save(self, commit=True):
            user = self.instance.user
            user.email = self.cleaned_data['email']
            user.first_name =
self.cleaned_data['first_name']
            user.last_name =
self.cleaned_data['last_name']

            if commit:
                user.save()
                super(EditProfileForm,
self).save(commit=True)
            return user
```


Manajemen Profil Pengguna

Dengan konfigurasi ini, pengguna bisa langsung mengedit informasi mereka di halaman profil tanpa perlu navigasi ke halaman lain.

Form yang kita buat, `EditProfileForm`, sudah cukup lengkap. Ini menggabungkan field dari model `User` dan `UserProfile`, dan menginisialisasi field `User` di dalam form. Namun, kita perlu memastikan bahwa field `username` tidak bisa diedit oleh pengguna, karena ini adalah field yang biasanya tetap konstan.

7.2.2 Memperbarui Views Profile

Selanjutnya, kita akan memperbaiki tampilan yang akan mengontrol logika halaman profil pengguna. Tampilan ini akan menangani pengambilan data, pengeditan profil pengguna, serta menampilkan profile pengguna.

Buka file `apps/users/views.py` dan tambahkan kode berikut:

```
# File: apps/users/views.py
from django.shortcuts import render, redirect
from django.contrib.auth.decorators import login_required
from apps.users.forms import EditProfileForm

@login_required
def profile(request):
    user = request.user
    profile = user.userprofile # Dapatkan
    profil user yang sedang login
```

Manajemen Profil Pengguna

```
if request.method == 'POST':
    form = EditProfileForm(request.POST,
request.FILES, instance=profile, user=user)
    if form.is_valid():
        form.save()
        return redirect('profile') #
Redirect ke halaman profil setelah berhasil
    else:
        form = EditProfileForm(instance=profile,
user=user)

# Definisikan context yang akan diberikan ke
template
context = {
    'ASSETS_ROOT': '/static/assets', # Aset
statis
    'form': form, # Form untuk mengedit
profil
    'profile': profile, # Profil user yang
sedang login
}

# Kembalikan context bersama dengan render
halaman
return render(request, 'users/profile.html',
context)
```

Pada kode di atas, fungsi `profile` mengelola tampilan halaman profil.

7.2.3 Memperbarui Template Halaman Profil

Berikut adalah beberapa perbaikan pada kode template `profile.html` agar lebih sesuai dengan penggunaan form Django dan memastikan semua field diambil dengan benar dari form Django:

Manajemen Profil Pengguna

Berikut adalah perbaikan kode kita:

```
<!-- apps/templates/users/profile.html -->
{% extends "dashboard/base.html" %}

{% block content %}
<div class="content">
  <div class="row">
    <div class="col-md-8">
      <div class="card">
        <div class="card-header">
          <h5 class="title">Edit
Profile</h5>
        </div>
        <div class="card-body ">
          <form method="post"
enctype="multipart/form-data">
            {% csrf_token %}
            <div class="row">
              <div class="col-md-
6">
                <div
class="form-group">
<label>Username</label>
{{ form.username }}
                </div>
              </div>
              <div class="col-md-
6">
                <div
class="form-group">
<label>Email</label>
```

Manajemen Profil Pengguna

```
{{ form.email }}
        </div>
    </div>
</div>
<div class="row">
    <div class="col-md-
6">
        <div
class="form-group">
            <label>First
Name</label>
{{ form.first_name }}
        </div>
    </div>
    <div class="col-md-
6">
        <div
class="form-group">
            <label>Last
Name</label>
{{ form.last_name }}
        </div>
    </div>
</div>
<div class="row">
    <div class="col-md-
12">
        <div
class="form-group">
<label>Company</label>
{{ form.company }}
        </div>
    </div>
</div>
```

Manajemen Profil Pengguna

```


<label>Address</label>
{{ form.address }}



<label>City</label>
{{ form.city }}



<label>Country</label>
{{ form.country }}



<label>Postal Code</label>


```

Manajemen Profil Pengguna

```
{{ form.postal_code }}
        </div>
    </div>
</div>
<div class="row">
    <div class="col-md-
12">
        <div
class="form-group">
            <label>About
Me</label>
        </div>
        </div>
        </div>
        <!-- Field untuk foto
profile -->
        <div class="form-group">
            <label>Profile
Picture</label>
            {{ form.foto_profile
}}
        </div>
        <button type="submit"
class="btn btn-fill btn-primary">Save</button>
    </form>
</div>
</div>
<div class="col-md-4">
    <div class="card card-user text-
center">
        <div class="card-body">
            <div class="author">
                
        <h5 class="title text-
center">{{ form.username.value }}</h5>
        </div>
        <div class="card-description
text-center">
                {{ form.about.value }}
            </div>
        </div>
    </div>
</div>
{% endblock %}
```

Perbaikan yang dilakukan:

- Penggunaan langsung `{{ form.field }}` agar Django menangani elemen-elemen form secara otomatis.
- Menghilangkan pemisahan antara `profile_form` dan `form`. Semua field kini ditangani oleh `form` dari `EditProfileForm`.
- Field `foto_profile` langsung dirender menggunakan `{% if form.foto_profile.value %}` untuk memastikan penanganan gambar profile tetap valid.

Dengan perbaikan ini, tampilan form lebih konsisten dengan Django dan proses edit bisa dilakukan pada halaman yang sama.

7.2.4 Menguji Halaman Profil Pengguna

Langkah pertama adalah memastikan bahwa halaman profil pengguna ditampilkan dengan benar dan dapat digunakan untuk mengedit informasi profil.

Akses Halaman Profil

Buka browser web dan masuk ke aplikasi Anda dengan akun pengguna yang sudah terdaftar. Navigasikan ke halaman profil pengguna. Biasanya, URL untuk halaman profil adalah `/profile/`. Pastikan Anda diarahkan ke halaman yang sesuai.

Verifikasi Tampilan Halaman Profil

Periksa bahwa semua elemen yang diharapkan ada pada halaman, termasuk formulir untuk mengedit profil, foto profil, dan informasi pengguna. Verifikasi bahwa data yang ditampilkan pada formulir sesuai dengan data yang tersimpan di database.

Uji Fungsi Pengeditan Profil

Coba mengedit beberapa informasi pada profil seperti nama depan, nama belakang, dan alamat email. Masukkan perubahan yang diinginkan dan klik tombol "Save".

```
<!-- users/templates/dashboard/profile.html -->
<input type="text" class="form-control"
name="first_name"
value="{{ form.first_name.value }}">
<input type="email" class="form-control"
name="email" value="{{ form.email.value }}">
```


Manajemen Profil Pengguna

Upload Foto Profil

Uji pengunggahan foto profil dengan memilih gambar baru dan menyimpannya. Pastikan gambar baru ditampilkan dengan benar setelah penyimpanan.

Dengan mengikuti langkah-langkah ini, Anda dapat memastikan bahwa halaman profil pengguna berfungsi dengan baik dan memberikan pengalaman pengguna yang optimal.

7.3 Integrasi Dengan Model Log Aktivitas

Selanjutnya, kita perlu memperbaiki untuk menampilkan log aktivitas ini di dashboard pengguna. Untuk melakukan hal ini, kita harus menambahkan logika di view dashboard serta memperbaiki template dashboard untuk menampilkan log aktivitas.

Kita sudah membuat model `ActivityLog` di bab sebelumnya, langkah berikutnya adalah mengintegrasikan fungsi pencatatan log aktivitas di view yang mengelola pengeditan profil pengguna. Setiap kali pengguna mengedit profilnya, tindakan tersebut akan dicatat.

Buka file `views.py` pada aplikasi `users` dan sesuaikan kode untuk view pengeditan profil seperti berikut:

```
# File: apps/users/views.py
from django.shortcuts import render, redirect
```

Manajemen Profil Pengguna

```
from django.contrib.auth.decorators import login_required
from apps.users.forms import EditProfileForm
from apps.dashboard.models import ActivityLog
from django.utils import timezone

@login_required
def profile(request):
    user = request.user
    profile = user.userprofile # Dapatkan profil user yang sedang login

    if request.method == 'POST':
        form = EditProfileForm(request.POST, request.FILES, instance=profile, user=user)
        if form.is_valid():
            form.save()

            # Membuat log aktivitas ketika profil diperbarui
            ActivityLog.objects.create(
                user=user,
                action="Mengedit profil",
                timestamp=timezone.now()
            )

            return redirect('profile') # Redirect ke halaman profil setelah berhasil
        else:
            form = EditProfileForm(instance=profile, user=user)

    # Definisikan context yang akan diberikan ke template
    context = {
        'ASSETS_ROOT': '/static/assets', # Aset statis
        'form': form, # Form untuk mengedit profil
    }
```

Manajemen Profil Pengguna

```
        'profile': profile, # Profil user yang
sedang login
    }

    return render(request, 'users/profile.html',
context)
```

Kode di atas menambahkan pencatatan log aktivitas setiap kali pengguna berhasil mengubah profilnya. Log ini akan disimpan dalam model `ActivityLog` dengan mencatat pengguna yang melakukan perubahan, jenis tindakan (dalam hal ini "Mengedit profil"), dan waktu kejadian.

7.3.1 Memperbaiki Template untuk Menampilkan Log Aktivitas di Dashboard

Selanjutnya, kita perlu memperbaiki untuk menampilkan log aktivitas ini di dashboard pengguna. Untuk melakukan hal ini, kita harus memperbaiki template dashboard untuk menampilkan log aktivitas.

View `activity_log` ini akan menampilkan semua log aktivitas pengguna yang sedang login dengan urutan berdasarkan waktu, di mana log terbaru akan muncul di bagian atas.

Selanjutnya, perbaiki file template `dashboard.html` di dalam folder `apps/users/templates/dashboard/`:

Mari kita mulai dengan meninjau kembali kode HTML di kedua file (`base.html` dan `dashboard.html`). Pertama, kita akan memastikan bahwa komponen Bootstrap yang digunakan telah disusun dengan benar untuk mendukung responsivitas.

Manajemen Profil Pengguna

Memperbaiki dashboard.html

File `dashboard.html` ini merupakan halaman khusus yang ditampilkan kepada pengguna saat mereka masuk ke dashboard. Mari kita perbaiki agar tampilannya lebih responsif dan menarik.

```
<!-- File:
apps/templates/dashboard/dashboard.html -->
{% extends 'dashboard/base.html' %}

{% block content %}
<div class="container mt-4">
  <div class="row">
    <div class="col-lg-12">
      <div class="jumbotron text-center
bg-primary text-white py-5">
        <h2>Selamat Datang di Dashboard,
{{ user_profile.user.username }}!</h2>
        <p>Di sini Anda dapat mengelola
bot, melihat statistik, dan memeriksa log
aktivitas terbaru Anda.</p>
      </div>
    </div>
  </div>
</div>

<!-- Menampilkan pesan jika ada error atau
sukses -->
{% if messages %}
  <div class="messages">
    {% for message in messages
%}
      <div class="alert
{{ message.tags }}">{{ message }}</div>
    {% endfor %}
  </div>
{% endif %}
```

Manajemen Profil Pengguna

```
<div class="row mt-5">
  <div class="col-lg-12">
    <h3 class="text-center">Log
Aktivitas</h3>
    <table class="table table-striped
table-hover">
      <thead class="thead-dark">
        <tr>
          <th
scope="col">Waktu</th>
          <th
scope="col">Aktivitas</th>
        </tr>
      </thead>
      <tbody>
        {% for log in activity_logs
%}
          <tr>
            <td>{{ log.timestamp|
date:"d-m-Y H:i" }}</td>
            <td>{{ log.action
}}</td>
          </tr>
          {% empty %}
            <tr>
              <td colspan="2"
class="text-center">Tidak ada log aktivitas
ditemukan.</td>
            </tr>
          {% endfor %}
        </tbody>
      </table>
    </div>
  </div>
</div>
{% endblock %}
```

Perbaikan:

Manajemen Profil Pengguna

- **Penggunaan `py - 5` pada Jumbotron:** Saya menambahkan padding vertikal (`py - 5`) untuk membuat teks pada jumbotron terlihat lebih baik dan tidak terlalu padat.
- **Table Responsif:** Bootstrap secara otomatis membuat tabel lebih responsif dan mudah dibaca pada berbagai ukuran layar.

Dengan mengikuti langkah di atas kita sudah berhasil menampilkan log aktivitas user di dashboard ketika mereka mengedit profile.

Dengan menyusun ulang dan memperbaiki struktur `base.html` dan `dashboard.html`, kita telah memastikan bahwa dashboard yang dibangun menjadi lebih responsif dan dapat diakses dengan baik di perangkat mobile.

7.4 Kode Lengkap

Kita akan mereview kembali semua kode untuk aplikasi users ini. Berikut adalah kode lengkapnya.

7.4.1 Forms.py

```
# File: apps/users/forms.py

from django import forms
from django.contrib.auth.models import User
from apps.authentication.models import
UserProfile
```

```
class EditProfileForm(forms.ModelForm):
    # Field dari model User
    email = forms.EmailField(required=True,
widget=forms.EmailInput(attrs={'class': 'form-
control', 'placeholder': 'Email'}))
    first_name = forms.CharField(max_length=100,
widget=forms.TextInput(attrs={'class': 'form-
control', 'placeholder': 'First Name'}))
    last_name = forms.CharField(max_length=100,
widget=forms.TextInput(attrs={'class': 'form-
control', 'placeholder': 'Last Name'}))

    # Field dari model UserProfile
    address = forms.CharField(max_length=255,
required=False,
widget=forms.TextInput(attrs={'class': 'form-
control', 'placeholder': 'Address'}))
    city = forms.CharField(max_length=100,
required=False,
widget=forms.TextInput(attrs={'class': 'form-
control', 'placeholder': 'City'}))
    country = forms.CharField(max_length=100,
required=False,
widget=forms.TextInput(attrs={'class': 'form-
control', 'placeholder': 'Country'}))
    postal_code = forms.CharField(max_length=20,
required=False,
widget=forms.TextInput(attrs={'class': 'form-
control', 'placeholder': 'Postal Code'}))
    company = forms.CharField(max_length=255,
required=False,
widget=forms.TextInput(attrs={'class': 'form-
control', 'placeholder': 'Company'}))
    about = forms.CharField(max_length=255,
required=False,
widget=forms.Textarea(attrs={'class': 'form-
control', 'placeholder': 'About me', 'rows':
4}))
```

Manajemen Profil Pengguna

```
foto_profile =
forms.ImageField(required=False,
widget=forms.FileInput(attrs={'class': 'form-
control-file'}))

class Meta:
    model = UserProfile
    fields = ['city', 'address', 'country',
'postal_code', 'about', 'company',
'foto_profile']

def __init__(self, *args, **kwargs):
    user = kwargs.pop('user', None) # Ambil
data user
    super(EditProfileForm,
self).__init__(*args, **kwargs)

    if user:
        # Inisialisasi field dari model User
        self.fields['username'] =
forms.CharField(
            initial=user.username,
            widget=forms.TextInput(attrs={'class': 'form-
control', 'readonly': 'readonly'})
        )
        self.fields['email'].initial =
user.email
        self.fields['first_name'].initial =
user.first_name
        self.fields['last_name'].initial =
user.last_name

def save(self, commit=True):
    user = self.instance.user
    user.email = self.cleaned_data['email']
    user.first_name =
self.cleaned_data['first_name']
```


Manajemen Profil Pengguna

```
        user.last_name =
self.cleaned_data['last_name']

        if commit:
            user.save()
            super(EditProfileForm,
self).save(commit=True)
        return user
```

7.4.2 Views.py

```
# File: apps/users/views.py

from django.shortcuts import render, redirect
from django.contrib.auth.decorators import
login_required
from apps.users.forms import EditProfileForm
from apps.dashboard.models import ActivityLog
from django.utils import timezone

@login_required
def profile(request):
    user = request.user
    profile = user.userprofile # Dapatkan
profil user yang sedang login

    if request.method == 'POST':
        form = EditProfileForm(request.POST,
request.FILES, instance=profile, user=user)
        if form.is_valid():
            form.save()

            # Membuat log aktivitas ketika
profil diperbarui
            ActivityLog.objects.create(
                user=user,
                action="Mengedit profil",
                timestamp=timezone.now())
```

Manajemen Profil Pengguna

```
)  
  
        return redirect('profile') #  
Redirect ke halaman profil setelah berhasil  
    else:  
        form = EditProfileForm(instance=profile,  
user=user)  
  
        # Definisikan context yang akan diberikan ke  
template  
        context = {  
            'ASSETS_ROOT': '/static/assets', # Aset  
statis  
            'form': form, # Form untuk mengedit  
profil  
            'profile': profile, # Profil user yang  
sedang login  
        }  
  
        return render(request, 'users/profile.html',  
context)
```

7.4.3 Profile.html

```
<!-- apps/templates/users/profile.html -->  
{% extends "dashboard/base.html" %}  
  
{% block content %}  
<div class="content">  
    <div class="row">  
        <div class="col-md-8">  
            <div class="card">  
                <div class="card-header">  
                    <h5 class="title">Edit  
Profile</h5>  
                </div>  
                <div class="card-body ">
```

Manajemen Profil Pengguna

```
<form method="post"
enctype="multipart/form-data">
    {% csrf_token %}
    <div class="row">
        <div class="col-md-
6">
            <div
class="form-group">
<label>Username</label>
{{ form.username }}
            </div>
        </div>
        <div class="col-md-
6">
            <div
class="form-group">
<label>Email</label>
{{ form.email }}
            </div>
        </div>
    </div>
    <div class="row">
        <div class="col-md-
6">
            <div
class="form-group">
                <label>First
Name</label>
                {{ form.first_name }}
            </div>
        </div>
        <div class="col-md-
6">
```

Manajemen Profil Pengguna

```

                                <div
class="form-group">
                                <label>Last
Name</label>
                                </div>
                                </div>
                                </div>
                                <div class="row">
                                <div class="col-md-
12">
                                <div
class="form-group">
                                <label>Company</label>
                                {{ form.company }}
                                </div>
                                </div>
                                </div>
                                <div class="row">
                                <div class="col-md-
12">
                                <div
class="form-group">
                                <label>Address</label>
                                {{ form.address }}
                                </div>
                                </div>
                                </div>
                                <div class="row">
                                <div class="col-md-
4">
                                <div
class="form-group">

```

Manajemen Profil Pengguna

```
<label>City</label>
                                {{ form.city
}}
                                </div>
                                </div>
                                <div class="col-md-
4">
                                <div
class="form-group">
<label>Country</label>
{{ form.country }}
                                </div>
                                </div>
                                <div class="col-md-
4">
                                <div
class="form-group">
<label>Postal Code</label>
{{ form.postal_code }}
                                </div>
                                </div>
                                </div>
                                <div class="row">
                                <div class="col-md-
12">
                                <div
class="form-group">
                                <label>About
Me</label>
                                {{ form.about }}
                                </div>
                                </div>
```

Manajemen Profil Pengguna

```

        </div>
        <!-- Field untuk foto
profile -->
        <div class="form-group">
            <label>Profile
Picture</label>
            {{ form.foto_profile
        }}
        </div>
        <button type="submit"
class="btn btn-fill btn-primary">Save</button>
        </form>
    </div>
</div>
<div class="col-md-4">
    <div class="card card-user text-
center">
        <div class="card-body">
            <div class="author">
                
                <h5 class="title text-
center">{{ form.username.value }}</h5>
            </div>
            <div class="card-description
text-center">
                {{ form.about.value }}
            </div>
        </div>
    </div>
</div>
</div>
</div>
{% endblock %}
```

7.4.4 URLs.py

```
# File: apps/users/urls.py

from django.urls import path
from . import views

urlpatterns = [
    path('profile/', views.profile,
name='profile'),
]
```

Kesimpulan Bab

Pada Bab ini telah membahas secara menyeluruh bagaimana mengelola profil pengguna dalam aplikasi Django, dengan fokus pada pembuatan halaman profil, fitur pengeditan informasi, dan integrasi dengan log aktivitas.

Sub-bab 7.1 dimulai dengan pembuatan halaman profil pengguna, yang merupakan elemen penting untuk memberikan pengguna akses ke informasi pribadi mereka. Kami menjelaskan cara membuat view untuk halaman profil, merancang template yang sesuai, dan menyiapkan URL untuk mengakses halaman tersebut. Proses ini memastikan bahwa pengguna dapat melihat dan mengelola informasi pribadi mereka dengan mudah.

Pada **Sub-bab 7.2**, kami membahas fitur pengeditan informasi pengguna. Bagian ini mencakup pembuatan form untuk mengedit profil, memperbarui views profil untuk menangani data yang diperbarui, dan menyesuaikan template halaman profil agar sesuai dengan perubahan. Pengujian halaman profil juga disertakan untuk memastikan bahwa fitur pengeditan berfungsi dengan baik dan memberikan pengalaman pengguna yang mulus.

Sub-bab 7.3 berfokus pada integrasi dengan model log aktivitas, yang penting untuk melacak dan menampilkan aktivitas pengguna dalam aplikasi. Kami membahas bagaimana memperbaiki

Manajemen Profil Pengguna

template untuk menampilkan log aktivitas di dashboard dan bagaimana membuat sidebar responsif menggunakan Bootstrap untuk navigasi yang lebih baik.

Secara keseluruhan, bab ini memberikan panduan komprehensif tentang manajemen profil pengguna, dari pembuatan halaman profil dasar hingga penambahan fitur pengeditan dan integrasi log aktivitas. Dengan penerapan langkah-langkah ini, Anda dapat membangun sistem manajemen profil yang robust dan user-friendly, yang memungkinkan pengguna untuk mengelola informasi mereka dengan efektif dan melihat riwayat aktivitas mereka secara jelas.

BAB 8 - Pengenalan Bot dan Menyiapkan Halaman Dasar

Pada Bagin ini, kita akan mulai membuat halaman untuk *manajemen bot*. Namun, perlu diingat bahwa pada tahap ini halaman tersebut belum memiliki fungsionalitas penuh, melainkan kita akan menyiapkan fondasi dasarnya. Fondasi ini mencakup model dan formulir yang akan kita gunakan untuk mengelola bot, seperti membuat bot baru, serta melihat daftar bot yang sudah dibuat.

Model yang akan kita buat mencerminkan struktur data bot yang akan kita simpan di database. Di sini kita juga akan menyiapkan formulir (form) yang akan digunakan untuk menginput data saat pengguna membuat atau mengelola bot.

8.1 Menyiapkan Model Dan Formulir Untuk Bot

Mari kita mulai dengan menyiapkan model *Bot* yang akan digunakan untuk menyimpan data bot di dalam database. Model ini diletakkan di dalam file `apps/bots/models.py`. Setelah itu, kita akan menyiapkan formulir sederhana untuk mengelola data tersebut.

Pengenalan Bot dan Menyiapkan Halaman Dasar

8.1.1 Menyiapkan Model Bot

Pada tahap ini, kita akan membahas model yang digunakan untuk menyimpan data bot dalam aplikasi Django. Model ini penting karena menyimpan informasi kunci seperti jenis bot, token unik, URL webhook, pesan-pesan otomatis, dan status bot. Dengan memahami komponen dari model ini, kita dapat mengelola berbagai bot yang dibuat oleh pengguna dalam platform kita, baik itu bot Telegram maupun WhatsApp.

Model Bot akan dibuat dalam file `apps/bots/models.py`. Di dalam file ini, kita mendefinisikan model Bot dengan atribut-atribut yang diperlukan, seperti jenis bot, token, status, dan pesan otomatis.

Langkah pertama adalah membuka file `models.py` di dalam folder `apps/bots/`. Kita mulai dengan mengimpor modul yang dibutuhkan:

```
# File: apps/bots/models.py
from django.db import models
from django.contrib.auth.models import User
```

Pada baris ini, kita mengimpor `models` dari `django.db` untuk mendefinisikan model, serta `User` dari `django.contrib.auth.models` yang memungkinkan kita untuk menghubungkan bot yang dibuat dengan pengguna tertentu. Relasi antara bot dan pengguna akan sangat penting agar setiap bot yang dibuat dapat dihubungkan dengan akun pengguna yang valid.

Pengenalan Bot dan Menyiapkan Halaman Dasar

Selanjutnya, kita mendefinisikan kelas `Bot` yang akan menjadi representasi model dasar untuk setiap bot. Di dalam model ini, kita akan menyertakan atribut seperti nama, deskripsi, token, URL webhook, dan pesan otomatis.

```
# File: apps/bots/models.py

class Bot(models.Model):
    BOT_CHOICES = [
        ('telegram', 'Telegram'),
        ('whatsapp', 'WhatsApp'),
    ]
    user = models.ForeignKey(User,
on_delete=models.CASCADE) # Bot akan terkait
dengan pengguna
    bot_type = models.CharField(max_length=20,
choices=BOT_CHOICES) # Jenis bot: Telegram atau
WhatsApp

    token_telegram =
models.CharField(max_length=255, unique=True,
help_text="Masukkan token bot yang unik.") #
Token Telegram
    webhook_url =
models.URLField(max_length=255, blank=True,
help_text="URL webhook bot akan diatur secara
otomatis.") # URL webhook

    is_active =
models.BooleanField(default=True,
help_text="Status bot aktif atau tidak.") #
Status aktif atau tidaknya bot

    # Pesan otomatis
```

Pengenalan Bot dan Menyiapkan Halaman Dasar

```
start_message = models.TextField(blank=True,
default="Selamat datang di bot!",
help_text="Pesan yang dikirim saat pengguna
mengetik /start.")
help_message = models.TextField(blank=True,
default="Berikut cara menggunakan bot.",
help_text="Pesan yang dikirim saat pengguna
mengetik /help.")

messages = models.JSONField(default=list,
blank=True, help_text="Daftar pesan yang
diterima dan responsnya dalam format JSON.") #
Format JSON

# Timestamp
created_at =
models.DateTimeField(auto_now_add=True) #
Tanggal bot dibuat
updated_at =
models.DateTimeField(auto_now=True) # Tanggal
terakhir bot diperbarui

name = models.CharField(max_length=255,
blank=True, default="Bot Saya", help_text="Nama
bot.") # Nama bot
description = models.TextField(blank=True,
default="Ini bot pertama saya",
help_text="Deskripsi bot.") # Deskripsi bot

def __str__(self):
    return f'{self.bot_type.capitalize()}
Bot - {self.user.username}'

class Meta:
    verbose_name = "Bot"
    verbose_name_plural = "Bots"
```

Pengenalan Bot dan Menyiapkan Halaman Dasar

Kode Lengkap

```
# File: apps/bots/models.py

from django.db import models
from django.contrib.auth.models import User

class Bot(models.Model):
    BOT_CHOICES = [
        ('telegram', 'Telegram'),
        ('whatsapp', 'WhatsApp'),
    ]
    user = models.ForeignKey(User,
on_delete=models.CASCADE) # Bot akan terkait
dengan pengguna
    bot_type = models.CharField(max_length=20,
choices=BOT_CHOICES) # Jenis bot: Telegram atau
WhatsApp

    token_telegram =
models.CharField(max_length=255, unique=True,
help_text="Masukkan token bot yang unik.") #
Token Telegram
    webhook_url =
models.URLField(max_length=255, blank=True,
help_text="URL webhook bot akan diatur secara
otomatis.") # URL webhook

    is_active =
models.BooleanField(default=True,
help_text="Status bot aktif atau tidak.") #
Status aktif atau tidaknya bot

    # Pesan otomatis
    start_message = models.TextField(blank=True,
default="Selamat datang di bot!",
```

Pengenalan Bot dan Menyiapkan Halaman Dasar

```
help_text="Pesan yang dikirim saat pengguna
mengetik /start.")
    help_message = models.TextField(blank=True,
default="Berikut cara menggunakan bot.",
help_text="Pesan yang dikirim saat pengguna
mengetik /help.")

    messages = models.JSONField(default=list,
blank=True, help_text="Daftar pesan yang
diterima dan responsnya dalam format JSON.") #
Format JSON

    # Timestamp
    created_at =
models.DateTimeField(auto_now_add=True) #
Tanggal bot dibuat
    updated_at =
models.DateTimeField(auto_now=True) # Tanggal
terakhir bot diperbarui

    name = models.CharField(max_length=255,
blank=True, default="Bot Saya", help_text="Nama
bot.") # Nama bot
    description = models.TextField(blank=True,
default="Ini bot pertama saya",
help_text="Deskripsi bot.") # Deskripsi bot

    def __str__(self):
        return f'{self.bot_type.capitalize()}
Bot - {self.user.username}'

    class Meta:
        verbose_name = "Bot"
        verbose_name_plural = "Bots"
```

Penjelasan kode:

Pengenalan Bot dan Menyiapkan Halaman Dasar

- **user:** Atribut ini adalah relasi ForeignKey antara model Bot dan model User. Setiap bot terkait dengan satu pengguna, dan dengan parameter `on_delete=models.CASCADE`, jika pengguna dihapus, semua bot milik pengguna tersebut akan ikut dihapus.
- **bot_type:** Kolom ini menggunakan opsi yang ditentukan oleh BOT_CHOICES untuk menentukan apakah bot tersebut adalah bot Telegram atau WhatsApp.
- **token_telegram:** Untuk bot Telegram, kita membutuhkan token yang unik. Token ini penting karena digunakan untuk mengidentifikasi dan mengontrol bot di platform Telegram.
- **webhook_url:** URL webhook adalah endpoint yang digunakan oleh bot untuk menerima pesan dari platform (Telegram atau WhatsApp). URL ini dapat dikosongkan karena akan diatur secara otomatis berdasarkan konfigurasi.
- **is_active:** Atribut ini menandai apakah bot dalam status aktif atau tidak. Jika bernilai `True`, bot akan berfungsi normal. Jika `False`, bot dianggap nonaktif.
- **start_message dan help_message:** Kedua kolom ini berfungsi sebagai pesan otomatis yang dikirimkan saat pengguna mengirimkan perintah `/start` dan `/help` ke bot. Pesan ini dapat diubah sesuai keinginan pengguna.
- **messages:** Kolom ini menyimpan daftar pesan dan responsnya dalam format JSON. Format JSON memudahkan kita untuk mengatur respons otomatis berdasarkan pesan yang diterima dari pengguna.
- **created_at dan updated_at:** Dua atribut ini mencatat kapan bot dibuat (`created_at`) dan kapan terakhir diper-

Pengenalan Bot dan Menyiapkan Halaman Dasar

baru (`updated_at`). Secara otomatis, nilai-nilai ini akan diatur oleh Django.

Setelah memahami model ini, kita telah menyiapkan struktur dasar untuk menyimpan dan mengelola bot di dalam aplikasi kita. Model ini juga mempersiapkan kita untuk langkah berikutnya, yaitu membuat form dan tampilan yang diperlukan untuk menambah dan mengelola bot melalui antarmuka pengguna.

Langkah Selanjutnya: Membuat dan Menerapkan Migrasi

Setelah model `Bot` selesai didefinisikan, langkah berikutnya adalah membuat dan menerapkan migrasi agar perubahan ini tercermin di dalam database. Menerapkan migrasi memastikan bahwa semua kolom dan relasi yang telah kita definisikan akan dibuat di tabel database yang sesuai, sehingga data bot dapat disimpan dengan baik.

```
$ python manage.py makemigrations
$ python manage.py migrate
```

Dengan langkah ini, model `Bot` telah siap digunakan dalam aplikasi kita. Kita sekarang dapat melanjutkan ke pembuatan form dan tampilan untuk menambah dan mengelola bot.

8.1.2 Menyiapkan Formulir untuk Bot

Setelah model bot selesai didefinisikan, langkah selanjutnya dalam proses pengembangan aplikasi bot adalah menyiapkan formulir yang digunakan untuk mengelola data bot. Formulir ini

Pengenalan Bot dan Menyiapkan Halaman Dasar

akan memungkinkan pengguna untuk memasukkan informasi yang diperlukan untuk membuat atau mengedit bot.

Pada tahap ini, kita akan membuka file `apps/bots/form-s.py` dan mulai mendefinisikan formulir yang akan membantu kita memfasilitasi proses input data dari pengguna. Dengan menggunakan `ModelForm` yang disediakan oleh Django, kita bisa dengan mudah membuat formulir yang terhubung langsung dengan model yang sudah kita buat sebelumnya.

Pertama, kita perlu mengimpor modul `forms` dari `django` serta model `Bot` yang sudah kita definisikan sebelumnya:

```
# File: apps/bots/forms.py

from django import forms # Mengimpor modul
forms dari Django
from .models import Bot # Mengimpor model Bot
yang sudah dibuat sebelumnya
```

Pada kode di atas, kita memulai dengan mengimpor `forms` dari Django yang akan membantu kita dalam mendefinisikan formulir. Selain itu, kita juga mengimpor model `Bot` dari file `models` yang sudah kita buat pada tahap sebelumnya.

Membuat Kelas `BotForm`

Langkah berikutnya adalah membuat kelas `BotForm` yang berfungsi untuk menangani pembuatan dan pembaruan bot. Kelas ini akan menggunakan `ModelForm` dari Django untuk menghu-

Pengenalan Bot dan Menyiapkan Halaman Dasar

bungkan formulir dengan model Bot. Formulir ini akan mencakup input informasi dasar seperti jenis bot (Telegram atau WhatsApp). Berikut adalah kode lengkapnya:

```
# File: apps/bots/forms.py

from django import forms
from .models import Bot # Mengimpor model Bot

class BotForm(forms.ModelForm):
    class Meta:
        model = Bot # Menghubungkan formulir
        dengan model Bot
        fields = ['bot_type', 'token_telegram',
        'name', 'start_message', 'help_message',
        'description', 'is_active'] # Field yang
        ditampilkan dalam formulir
        widgets = {
            'name':
forms.TextInput(attrs={'class': 'form-control',
            'placeholder': 'Masukkan nama bot'}),
            'description':
forms.Textarea(attrs={'class': 'form-control',
            'rows': 3, 'placeholder': 'Deskripsi bot'}),
        }
        help_texts = {
            'token_telegram': 'Masukkan token
            unik yang diterima dari platform Telegram.',
            'name': 'Nama bot dapat diubah
            sesuai keinginan.',
            'description': 'Deskripsi singkat
            tentang bot dan fungsinya.',
        }
```

Pengenalan Bot dan Menyiapkan Halaman Dasar

Pada kode di atas, kita menggunakan beberapa fitur penting dari `ModelForm` untuk membuat proses pengisian data lebih mudah bagi pengguna:

- **Meta:** Bagian ini menghubungkan formulir dengan model `Bot`, serta mendefinisikan field yang akan ditampilkan di dalam formulir, seperti jenis bot (`bot_type`), token, nama, deskripsi, dan status bot (`is_active`).
- **Widgets:** Dengan `widgets`, kita dapat mengontrol tampilan dari setiap field pada formulir. Misalnya, kita menambahkan `class='form-control'` untuk elemen `name` dan `description`, sehingga tampilan formulir akan lebih rapi dengan memanfaatkan kelas bawaan `Bootstrap`.
- **Help Texts:** Untuk memandu pengguna, kita juga menambahkan `help_texts` yang memberikan penjelasan tambahan pada field tertentu, seperti token Telegram dan deskripsi bot, sehingga pengguna lebih paham mengenai data yang harus mereka masukkan.

Penjelasan Kode

Kode di atas menyiapkan formulir yang membantu pengguna dalam mengelola bot mereka. Penjelasan singkat mengenai masing-masing komponen dari formulir ini adalah sebagai berikut:

- **Meta:** Menghubungkan formulir dengan model `Bot` dan menentukan field mana saja yang akan diikutsertakan dalam formulir. Dalam hal ini, field yang ditampilkan adalah jenis bot, token, nama, deskripsi, dan status bot.

Pengenalan Bot dan Menyiapkan Halaman Dasar

- **Widgets:** Bagian ini mengubah tampilan input field. Sebagai contoh, `name` menggunakan `TextInput` dengan tambahan atribut `class='form-control'` agar lebih terlihat seperti komponen Bootstrap, sehingga tampilannya lebih konsisten dan ramah pengguna. Field `description` menggunakan `Textarea` dengan beberapa baris tambahan.
- **Help Texts:** Memberikan informasi tambahan kepada pengguna untuk lebih memahami setiap field yang ada pada formulir. Ini membantu agar pengguna tidak bingung ketika harus mengisi field yang mungkin kurang familiar, seperti `token_telegram`.

Menghubungkan Formulir dengan View

Setelah formulir berhasil dibuat, langkah berikutnya adalah menghubungkan formulir ini dengan tampilan (view) kita, sehingga data yang diisi oleh pengguna dapat disimpan ke dalam database. Namun, untuk saat ini, kita fokus pada persiapan formulir, dan implementasi view akan dijelaskan pada sub-bab berikutnya.

Dengan formulir yang telah dibuat, pengguna dapat dengan mudah mengisi informasi yang diperlukan untuk membuat atau mengedit bot. Hal ini memberikan fondasi kuat untuk pengelolaan bot yang lebih kompleks ke depannya, seperti menghubungkan bot dengan webhook atau menambahkan fitur manajemen lebih lanjut.

Pengenalan Bot dan Menyiapkan Halaman Dasar

Pada tahap berikutnya, kita akan mengimplementasikan logika di balik formulir ini pada tampilan, sehingga data yang diisi oleh pengguna dapat diproses dan disimpan dengan benar di dalam database.

8.2 Menyiapkan Handle Bot

Setelah kita menyiapkan formulir untuk mengelola bot, langkah berikutnya adalah menyiapkan fungsi-fungsi yang akan menangani interaksi antara bot dan pengguna. Pada bagian ini, kita akan mengimplementasikan handle untuk Telegram dan WhatsApp bot. Handle ini berfungsi sebagai penghubung antara pesan yang diterima bot dan respons yang akan dikirim kembali ke pengguna. Dengan kata lain, handle bertugas memproses permintaan pengguna dan memberikan jawaban yang sesuai.

Buat folder `services` di dalam folder `bots`, dan buat file `telegram.py` dan `whatsapp.py` untuk di letakkan fungsi `handle_update` di dalamnya.

```
bots/  
  views.py  
  services/  
    telegram.py  
    whatsapp.py
```

Pengenalan Bot dan Menyiapkan Halaman Dasar

Mari kita mulai dengan membuat handler untuk Telegram dan WhatsApp di file `apps/bots/services/telegram.py` dan `apps/bots/services/whatsapp.py`.

8.2.1 Handle Telegram

Setelah kita menyiapkan model dan formulir untuk bot, langkah selanjutnya adalah menyiapkan layanan yang akan menangani interaksi antara pengguna dan bot. Dalam hal ini, untuk bot Telegram, kita perlu membuat sebuah mekanisme yang dapat menerima dan memproses pembaruan (update) dari pengguna.

Ketika pengguna mengirimkan pesan ke bot, Telegram akan mengirimkan data berupa JSON yang berisi informasi mengenai pesan tersebut. Tugas kita adalah memproses data ini dan mengirimkan respons yang sesuai kembali ke pengguna.

Untuk mencapai hal ini, kita akan memanfaatkan API Telegram yang memungkinkan kita untuk mengambil pesan, memprosesnya, dan mengirimkan respons yang diperlukan. Di sini, kita akan membahas kode yang menangani update untuk bot Telegram. Kode ini akan ditempatkan dalam file **`apps/bots/services/telegram.py`**, yang akan menangani semua logika pemrosesan untuk bot Telegram.

Mengimpor Dependensi

Kita mulai dengan mengimpor dependensi yang dibutuhkan untuk menghubungkan aplikasi kita dengan API Telegram dan juga untuk mengelola bot yang terdaftar dalam aplikasi. Pada file ini,

Pengenalan Bot dan Menyiapkan Halaman Dasar

kita akan menggunakan **requests** untuk melakukan HTTP request, serta modul **logging** untuk mencatat error atau informasi selama bot berjalan.

```
# File: apps/bots/services/telegram.py

from apps.bots.models import Bot # Mengimpor
model Bot dari aplikasi bots
import requests # Untuk mengirim permintaan
HTTP ke API Telegram
import logging # Untuk logging error dan
informasi terkait bot

logger = logging.getLogger(__name__) #
Menginisialisasi logger untuk mencatat error
```

Pada bagian ini, kita menginisialisasi sebuah logger dengan nama yang sesuai dengan modul saat ini. Logger ini akan sangat berguna untuk memantau apakah ada kesalahan dalam mengirim pesan atau ketika API Telegram tidak merespons sesuai harapan.

Fungsi `handle_update(update)`

Fungsi **handle_update** adalah fungsi utama yang akan menangani update dari bot Telegram. Setiap kali pengguna mengirimkan pesan ke bot, Telegram akan mengirimkan data update dalam format JSON, yang kemudian diproses oleh fungsi ini. Di sini, kita akan memproses data pesan dan menentukan tindakan apa yang harus diambil oleh bot, berdasarkan perintah yang diberikan oleh pengguna.

```
def handle_update(update):
```


Pengenalan Bot dan Menyiapkan Halaman Dasar

```
    chat_id = update['message']['chat']['id'] #  
Mengambil chat_id dari pesan yang diterima  
    text = update['message']['text'].lower() #  
Mengubah teks pesan ke huruf kecil untuk  
memudahkan perbandingan  
  
    bot =  
Bot.objects.filter(bot_type='telegram').first()  
# Mengambil bot Telegram pertama dari database  
  
    if bot:  
        # Memproses perintah yang diterima dari  
pengguna  
        if text == "/start":  
            response_text = bot.start_message #  
Mengambil pesan sambutan dari model Bot  
        elif text == "/help":  
            response_text = bot.help_message #  
Mengambil pesan bantuan dari model Bot  
        else:  
            response_text = 'Perintah tidak  
dikenal. Ketik /help untuk daftar perintah.' #  
Pesan default untuk perintah yang tidak dikenali  
  
        # Mengirim pesan respons ke pengguna  
        send_message(bot.token_telegram,  
"sendMessage", {  
            'chat_id': chat_id,  
            'text': response_text  
        })
```

Dalam fungsi ini, kita mengambil **chat_id** dari pesan yang diterima, yang merupakan identifikasi unik dari percakapan antara bot dan pengguna. Kemudian, kita mengambil teks dari pesan yang diterima dan mengubahnya menjadi huruf kecil agar konsisten ketika dibandingkan dengan perintah-perintah yang ada.

Pengenalan Bot dan Menyiapkan Halaman Dasar

Setelah itu, kita melakukan query ke database untuk mengambil bot Telegram yang terdaftar, yang diwakili oleh model **Bot**. Jika bot ditemukan, kita memeriksa isi dari pesan pengguna dan menentukan respons yang sesuai berdasarkan perintah seperti **/start** atau **/help**. Jika perintah yang diberikan tidak dikenali, bot akan merespons dengan pesan default.

Fungsi `send_message(token, method, data)`

Fungsi **`send_message`** digunakan untuk mengirimkan pesan balasan ke pengguna melalui API Telegram. Fungsi ini memerlukan token bot yang disimpan dalam model **Bot**, serta data tambahan seperti **`chat_id`** dan teks pesan yang akan dikirimkan.

```
def send_message(token_telegram, method, data):
    telegram_api_url =
f'https://api.telegram.org/bot{token_telegram}/{method}' # Membentuk URL untuk API Telegram
    response = requests.post(telegram_api_url,
data=data) # Mengirim permintaan POST ke API Telegram

    # Memeriksa apakah permintaan berhasil
    if response.status_code != 200:
        logger.error(f"Failed to send message:
{response.text}") # Mencatat error jika gagal
mengirim pesan
    return response # Mengembalikan respons
dari API
```

Fungsi ini pertama-tama membangun URL API Telegram menggunakan token bot dan metode yang diinginkan (dalam kasus ini,

Pengenalan Bot dan Menyiapkan Halaman Dasar

sendMessage). Setelah URL terbentuk, kita mengirim permintaan POST menggunakan **requests.post**, dengan data yang diperlukan seperti **chat_id** dan teks pesan. Jika API Telegram merespons dengan status selain **200 OK**, kita mencatat error menggunakan **logger.error**, yang memudahkan kita dalam memantau masalah yang mungkin terjadi saat bot mengirim pesan.

Kode Lengkap

```
# File: apps/bots/services/telegram.py

from apps.bots.models import Bot # Mengimpor
model Bot dari aplikasi bots
import requests # Untuk mengirim permintaan
HTTP ke API Telegram
import logging # Untuk logging error dan
informasi terkait bot

logger = logging.getLogger(__name__) #
Menginisialisasi logger untuk mencatat error
def handle_update(update):
    chat_id = update['message']['chat']['id'] #
Mengambil chat_id dari pesan yang diterima
    text = update['message']['text'].lower() #
Mengubah teks pesan ke huruf kecil untuk
memudahkan perbandingan

    bot =
Bot.objects.filter(bot_type='telegram').first()
# Mengambil bot Telegram pertama dari database

    if bot:
```

Pengenalan Bot dan Menyiapkan Halaman Dasar

```
# Memproses perintah yang diterima dari
pengguna
if text == "/start":
    response_text = bot.start_message #
    Mengambil pesan sambutan dari model Bot
elif text == "/help":
    response_text = bot.help_message #
    Mengambil pesan bantuan dari model Bot
else:
    response_text = 'Perintah tidak
    dikenal. Ketik /help untuk daftar perintah.' #
    Pesan default untuk perintah yang tidak dikenali

    # Mengirim pesan respons ke pengguna
    send_message(bot.token_telegram,
    "sendMessage", {
        'chat_id': chat_id,
        'text': response_text
    })

def send_message(token_telegram, method, data):
    telegram_api_url =
    f'https://api.telegram.org/bot{token_telegram}/{
    method}' # Membentuk URL untuk API Telegram
    response = requests.post(telegram_api_url,
    data=data) # Mengirim permintaan POST ke API
    Telegram

    # Memeriksa apakah permintaan berhasil
    if response.status_code != 200:
        logger.error(f"Failed to send message:
        {response.text}") # Mencatat error jika gagal
        mengirim pesan
    return response # Mengembalikan respons
    dari API
```

Pengenalan Bot dan Menyiapkan Halaman Dasar

Pada bagian ini, kita telah menyiapkan dasar yang kuat untuk menangani interaksi antara pengguna dan bot Telegram. Dengan menggunakan fungsi **`handle_update`**, kita dapat memproses pesan dari pengguna dan meresponsnya dengan tepat. Fungsi **`send_message`** membantu kita dalam mengirim pesan balasan melalui API Telegram, serta mencatat error jika terjadi kegagalan.

Langkah selanjutnya adalah mengintegrasikan layanan ini ke dalam alur aplikasi secara lebih mendalam, seperti menghubungkan dengan webhook, sehingga setiap pembaruan yang diterima oleh bot dapat diproses secara otomatis oleh fungsi yang telah kita buat. Dengan fondasi ini, kita sudah selangkah lebih dekat menuju implementasi penuh bot yang dapat dioperasikan oleh pengguna melalui platform Telegram.

8.2.2 Handle Whatsapp

Setelah kita menyiapkan fungsi penanganan untuk bot Telegram, langkah selanjutnya adalah mengimplementasikan fungsi serupa untuk WhatsApp. Dalam hal ini, kita akan menggunakan layanan Twilio untuk mengirim dan menerima pesan melalui WhatsApp. Twilio menyediakan API yang memungkinkan kita mengelola komunikasi antara pengguna dan bot dengan mudah. Seperti pada bot Telegram, fungsi ini akan menangani pesan yang diterima dari pengguna, memprosesnya, dan meresponsnya dengan sesuai.

Kita akan menggunakan file `apps/bots/services/whatsapp.py` untuk menangani pembaruan yang datang dari What-

Pengenalan Bot dan Menyiapkan Halaman Dasar

sApp. Berikut adalah implementasi kode untuk menangani pesan masuk:

```
# File: apps/bots/services/whatsapp.py

from django.http import HttpResponse # Untuk
mengembalikan respon HTTP
from django.views.decorators.csrf import
csrf_exempt # Agar view dapat diakses tanpa
validasi CSRF
from twilio.twiml.messaging_response import
MessagingResponse # Untuk membuat balasan ke
Twilio API
from apps.bots.models import Bot # Mengimpor
model Bot dari aplikasi bots
import logging # Untuk logging aktivitas dan
error

logger = logging.getLogger(__name__) #
Inisialisasi logger

@csrf_exempt # Dekorator untuk menonaktifkan
validasi CSRF pada view ini
def handle_update(update):
    # Mendapatkan pengirim pesan dan isi pesan
    user = update['From']
    message = update['Body'].strip().lower() #
Mengubah pesan ke lowercase untuk perbandingan

    # Logging pesan yang diterima
    logger.info(f'{user} says {message}')

    # Mengambil bot pertama yang bertipe
    WhatsApp dari database
    bot =
Bot.objects.filter(bot_type='whatsapp').first()
```

Pengenalan Bot dan Menyiapkan Halaman Dasar

```
# Membuat respon menggunakan Twilio's
MessagingResponse
response = MessagingResponse()

if bot: # Jika bot ditemukan
    # Memeriksa perintah yang dikirim oleh
    pengguna
    if message == "/start":
        response_text = bot.start_message #
        Pesan sambutan
    elif message == "/help":
        response_text = bot.help_message #
        Pesan bantuan
    else:
        response_text = 'Perintah tidak
        dikenal. Ketik /help untuk melihat daftar
        perintah.' # Pesan default

    response.message(response_text) #
    Mengirim balasan pesan
else:
    response.message('Bot tidak ditemukan.')
# Jika bot tidak ditemukan di database

# Mengembalikan respon sebagai HTTP response
return HttpResponse(str(response))
```

Penjelasan Kode:

- **Mengimpor Dependensi**

Di awal kode, kita mengimpor beberapa modul penting. `HttpResponse` digunakan untuk mengirimkan respons HTTP kembali ke Twilio, sementara `csrf_exempt` memungkinkan kita untuk menonaktifkan validasi CSRF pada view ini, karena

Pengenalan Bot dan Menyiapkan Halaman Dasar

Twilio tidak mengirimkan token CSRF dalam webhook-nya. Kita juga mengimpor `MessagingResponse` dari Twilio untuk membangun balasan pesan ke pengguna, serta mengimpor model `Bot` untuk mendapatkan informasi terkait bot dari database.

- **`handle_update(update)`**

Fungsi `handle_update` adalah inti dari mekanisme penanganan pesan di WhatsApp. Fungsi ini menerima parameter `update`, yang berisi data JSON yang dikirim oleh Twilio setiap kali ada interaksi dari pengguna. Data tersebut mencakup informasi seperti nomor pengirim (`From`) dan pesan (`Body`). Kita mengubah pesan yang diterima menjadi huruf kecil menggunakan `message.lower()` untuk memudahkan perbandingan.

- **Mengambil Bot dari Database**

Dalam kode ini, kita melakukan query terhadap database untuk mendapatkan bot pertama yang bertipe WhatsApp. Logika ini mirip dengan yang kita gunakan pada `handle_telegram`, di mana kita hanya mengambil satu bot terlebih dahulu. Namun, di pengembangan berikutnya, kita dapat memperluas logika ini untuk mendukung beberapa bot WhatsApp dalam satu aplikasi.

- **Memproses Pesan**

Pesan yang diterima kemudian diperiksa apakah berisi perintah tertentu, seperti `/start` atau `/help`. Jika pengguna mengirimkan perintah `/start`, bot akan merespons dengan pesan sambutan yang disimpan dalam kolom `start_message` pada model `Bot`. Begitu pula

Pengenalan Bot dan Menyiapkan Halaman Dasar

jika perintah `/help` diterima, bot akan mengirimkan pesan bantuan yang telah disiapkan. Jika perintah yang dikirimkan tidak dikenal, bot akan memberikan respons default yang menyarankan pengguna untuk mengetik `/help`.

- **Mengirim Respon Melalui Twilio API**

Untuk mengirimkan balasan kepada pengguna, kita menggunakan `MessagingResponse` dari Twilio. Fungsi ini membangun pesan yang dikirimkan kembali ke pengguna WhatsApp melalui Twilio. Setelah semua proses selesai, kita mengembalikan respons sebagai `HttpResponse` agar Twilio dapat menerima pesan dan menampilkannya kepada pengguna.

8.2.3 Fungsi Penanganan Platform

Setiap platform bot, baik itu Telegram maupun WhatsApp, memiliki format data dan event yang berbeda dalam cara mengirimkan pesan melalui webhook. Oleh karena itu, sangat penting untuk menyesuaikan fungsi penanganan masing-masing platform sesuai dengan format data yang mereka kirimkan.

Dalam hal ini, kita menggunakan API Telegram untuk bot Telegram, dan API Twilio untuk bot WhatsApp.

Beberapa poin yang perlu diperhatikan ketika menangani platform yang berbeda:

- **Telegram:** Telegram mengirimkan data dalam bentuk JSON yang mencakup informasi tentang pesan, pengguna, dan chat. Kita harus memastikan bahwa semua data

Pengenalan Bot dan Menyiapkan Halaman Dasar

ini diproses dengan benar melalui fungsi `handle_telegram_webhook`.

- **WhatsApp:** Data dari WhatsApp melalui Twilio juga mencakup pesan teks, media, serta informasi tentang pengiriman. Oleh karena itu, kita harus menangani format ini dengan baik dalam fungsi `handle_what_sapp_webhook`.

Dengan cara ini, kita dapat menangani data yang dikirim oleh setiap platform dengan baik dan memberikan respons yang sesuai kepada pengguna. Pastikan untuk selalu menguji setiap implementasi platform bot secara menyeluruh untuk memastikan bahwa bot dapat merespons pesan dengan benar dan sesuai dengan fungsionalitas yang diinginkan.

8.3 Menyiapkan Halaman Untuk Membuat Bot

Pada bagian ini, kita akan membangun halaman yang memungkinkan pengguna untuk membuat bot baru. Langkah-langkah ini akan mencakup penyiapan view, template, dan pengaturan URL routing agar proses pembuatan bot dapat diakses oleh pengguna melalui halaman web.

Sebelumnya, kita telah menyiapkan model dan formulir untuk bot. Kini, kita akan melanjutkan dengan implementasi view, template, dan URL routing untuk halaman pembuatan bot.

Pengenalan Bot dan Menyiapkan Halaman Dasar

8.3.1 Membuat View untuk Membuat Bot

Pada tahap ini, kita akan membuat tampilan (view) yang menangani logika di balik halaman pembuatan bot. View bertanggung jawab untuk memproses permintaan (request) dari pengguna, baik itu untuk menampilkan form pembuatan bot maupun untuk memproses data yang dikirimkan pengguna saat form disubmit. Tampilan ini akan memberikan fondasi awal dari proses pembuatan bot, meskipun pada tahap ini kita belum menambahkan fitur webhook atau pencatatan aktivitas (activity log). Fitur-fitur tersebut akan kita bahas lebih lanjut di bab mendatang.

Kita akan menyiapkan sebuah tampilan sederhana yang memungkinkan pengguna untuk membuat bot tanpa adanya integrasi dengan webhook atau activity log. Meskipun fungsionalitasnya terbatas, halaman ini sudah dapat diakses dan digunakan oleh pengguna. Kode berikut akan ditempatkan pada file `apps/bots/views.py`:

```
# File: apps/bots/views.py

from django.shortcuts import render, redirect,
get_object_or_404 # Untuk rendering template
dan pengalihan halaman
from django.contrib.auth.decorators import
login_required # Untuk memastikan hanya
pengguna yang login yang dapat mengakses view
ini
from .forms import BotForm # Mengimpor form
BotForm untuk pembuatan bot
from .models import Bot # Mengimpor model Bot
dari models.py
```

Pengenalan Bot dan Menyiapkan Halaman Dasar

```
@login_required # Dekorator untuk memastikan
hanya pengguna terdaftar yang dapat mengakses
halaman ini
def create_bot(request):
    """
    Tampilan untuk membuat bot baru. Fitur
    webhook dan activity log belum diintegrasikan.
    """
    if request.method == 'POST': # Mengecek
    apakah request berupa POST, artinya pengguna
    telah mengirimkan form
        form = BotForm(request.POST) # Membuat
    instance dari BotForm dengan data yang
    dikirimkan
        if form.is_valid(): # Memvalidasi data
    yang dikirimkan melalui form
            bot = form.save(commit=False) #
    Membuat instance Bot tanpa langsung menyimpannya
    ke database
            bot.user = request.user # Menyimpan
    informasi pengguna yang membuat bot
            bot.save() # Menyimpan bot ke dalam
    database

    # Pada tahap ini, tidak ada
    integrasi dengan webhook atau pencatatan
    aktivitas.

    # Fitur-fitur tersebut akan dibahas
    pada bab selanjutnya.

    return redirect('manage_bots') #
    Setelah berhasil menyimpan bot, pengguna
    dialihkan ke halaman manajemen bot
    else:
        form = BotForm() # Jika request adalah
    GET, tampilkan form kosong untuk diisi pengguna
```

Pengenalan Bot dan Menyiapkan Halaman Dasar

```
return render(request,  
'bots/create_bot.html', {'form': form}) #  
Render halaman create_bot dengan form
```

Penjelasan Kode:

- **Pengimporan Modul**

Pada bagian awal kode, kita mengimpor beberapa modul yang diperlukan. Fungsi `render` digunakan untuk menampilkan template HTML, sementara `redirect` mengalihkan pengguna ke halaman lain setelah bot berhasil dibuat. Fungsi `get_object_or_404` akan berguna jika kita perlu mengambil objek yang ada di database berdasarkan parameter tertentu, namun pada contoh ini belum digunakan. Selanjutnya, `login_required` memastikan bahwa hanya pengguna yang sudah login yang dapat mengakses view ini. Kita juga mengimpor `BotForm` yang digunakan sebagai form untuk membuat bot, serta model `Bot` yang menyimpan data bot.

- **Fungsi `create_bot(request)`**

Fungsi `create_bot` merupakan inti dari logika pembuatan bot. Fungsi ini menangani dua jenis permintaan utama: permintaan GET untuk menampilkan form kosong, dan permintaan POST untuk memproses data yang diisi pengguna dalam form.

- **Penanganan Metode POST**

Ketika pengguna mengisi form dan mengirimkannya, metode request berubah menjadi POST. Pada kondisi ini,

Pengenalan Bot dan Menyiapkan Halaman Dasar

view akan memproses data yang dikirimkan menggunakan `BotForm(request.POST)`. Setelah itu, data tersebut divalidasi menggunakan `form.is_valid()`. Jika validasi berhasil, objek bot dibuat dari form tersebut dengan `commit=False`, yang artinya bot belum disimpan ke database. Selanjutnya, informasi pengguna yang membuat bot disimpan pada properti `user` dari objek bot. Setelah itu, bot disimpan ke database menggunakan `bot.save()`.

- **Pengalihan ke Halaman Manajemen Bot**

Setelah bot berhasil disimpan, pengguna akan dialihkan ke halaman manajemen bot melalui `redirect('manage_bots')`. Pengalihan ini memberikan alur yang mulus dan intuitif bagi pengguna, karena mereka akan langsung dibawa ke halaman yang menampilkan bot-bot yang sudah mereka buat.

- **Menampilkan Form Kosong (Metode GET)**

Jika pengguna membuka halaman ini tanpa mengirimkan form (melalui permintaan GET), tampilan akan menampilkan form kosong menggunakan `form = BotForm()`. Hal ini memungkinkan pengguna untuk mengisi form dan memulai proses pembuatan bot.

- **Template Rendering**

Fungsi `render` digunakan untuk menampilkan halaman `create_bot.html` dengan form yang sudah disiapkan. Form ini nantinya dapat diisi oleh pengguna untuk membuat bot baru. Pada tahap ini, meskipun fungsionalitas webhook dan pencatatan aktivitas belum

Pengenalan Bot dan Menyiapkan Halaman Dasar

terintegrasi, struktur dasar halaman pembuatan bot sudah siap dan dapat digunakan.

Dalam bagian ini, kita telah menyiapkan halaman pembuatan bot yang berfungsi untuk menampilkan form dan memproses data yang dikirim oleh pengguna. Meskipun halaman ini belum memiliki fitur webhook dan activity log, kita telah membangun fondasi dasar untuk menampilkan form dan menyimpan bot yang baru dibuat ke dalam database. Integrasi dengan webhook dan pencatatan aktivitas akan kita bahas lebih lanjut pada bab selanjutnya.

8.3.2 Membuat Template untuk Membuat Bot

Setelah menyiapkan view yang menangani logika pembuatan bot, langkah selanjutnya adalah membuat template yang akan menjadi antarmuka bagi pengguna. Template ini bertanggung jawab untuk menampilkan formulir yang telah kita buat di view sebelumnya dan menyediakan elemen-elemen yang dapat diisi oleh pengguna sebelum data tersebut dikirimkan ke server. Template di Django tidak hanya menampilkan informasi statis, tetapi juga dapat memanfaatkan variabel dan komponen dinamis yang dikirimkan oleh view.

Dalam hal ini, kita akan membuat template `create_bot.html` yang akan menjadi halaman pembuatan bot. Pengguna dapat memilih jenis bot yang akan mereka buat (Telegram atau WhatsApp), mengisi informasi yang diperlukan sesuai dengan plat-

Pengenalan Bot dan Menyiapkan Halaman Dasar

form yang dipilih, dan kemudian mengirimkan data tersebut. Meskipun pada tahap ini template belum memiliki fungsionalitas webhook atau pencatatan aktivitas, template ini akan berfungsi dengan baik untuk proses pembuatan bot dasar.

Template ini berperan sebagai antarmuka pengguna yang memungkinkan interaksi langsung dengan form. Template akan menampilkan form pembuatan bot, menyediakan validasi dasar melalui penggunaan token CSRF, dan menggunakan beberapa fungsi JavaScript untuk meningkatkan pengalaman pengguna, seperti menyembunyikan atau menampilkan field tergantung pada platform yang dipilih.

Berikut adalah kode untuk template `create_bot.html` yang akan ditempatkan di direktori `templates/bots/create_bot.html`:

```
<!-- File: apps/templates/bots/create_bot.html -->

{% extends 'dashboard/base.html' %}

{% block content %}
<div class="container mt-4">
  <!-- Card untuk Formulir -->
  <div class="card">
    <div class="card-header">
      <h5 class="card-title">Buat Bot
Baru</h5>
    </div>
    <div class="card-body">
      <!-- Form untuk Bot -->
```


Pengenalan Bot dan Menyiapkan Halaman Dasar

```
<form method="post">
  {% csrf_token %}

  <!-- Form Group untuk Platform
(muncul pertama) -->
  <div class="form-group">
    <label class="form-label">
      Platform
    </label>
    <div class="btn-group"
role="group" aria-label="Platform">
      <input type="radio"
name="bot_type" id="telegram" value="telegram"
{% if form.bot_type.value == 'telegram'
%}checked{% endif %} onclick="toggleFields()">
      <label
for="telegram">Telegram</label>

      <input type="radio"
name="bot_type" id="whatsapp" value="whatsapp"
{% if form.bot_type.value == 'whatsapp'
%}checked{% endif %} onclick="toggleFields()">
      <label
for="whatsapp">WhatsApp</label>
    </div>
  </div>

  <!-- Form Group untuk Nama -->
  <div class="form-group">
    <label
for="{{ form.name.id_for_label }}">
      Nama
    </label>
    <input type="text"
name="name" id="{{ form.name.id_for_label }}"
value="{{ form.name.value }}" class="form-
control {% if form.name.errors %}is-invalid{%
endif %}">
```

Pengenalan Bot dan Menyiapkan Halaman Dasar

```
        {% for error in
form.name.errors %}
            <div class="invalid-
feedback">{{ error }}</div>
        {% endfor %}
    </div>

    <!-- Form Group untuk Token
(hanya untuk Telegram) -->
    <div class="form-group"
id="token-field">
        <label
for="{{ form.token_telegram.id_for_label }}">
            Token
        </label>
        <input type="text"
name="token"
id="{{ form.token_telegram.id_for_label }}"
value="{{ form.token_telegram.value }}"
class="form-control {% if
form.token_telegram.errors %}is-invalid{% endif
%}">
            {% for error in
form.token_telegram.errors %}
                <div class="invalid-
feedback">{{ error }}</div>
            {% endfor %}
        </div>

        <!-- Button Submit -->
        <button type="submit" class="btn
btn-primary">
            Buat Bot
        </button>
    </form>
</div>
</div>
```

Pengenalan Bot dan Menyiapkan Halaman Dasar

```
<script>

    // Fungsi untuk
    menyembunyikan/menampilkan elemen sesuai
    platform
    function toggleFields() {
        var tokenField =
document.getElementById('token-field');
        var whatsappFields =
document.getElementById('whatsapp-fields');

        if
(document.getElementById('telegram').checked) {
            tokenField.classList.remove('d-
none');
            whatsappFields.classList.add('d-
none');
        } else if
(document.getElementById('whatsapp').checked) {
            tokenField.classList.add('d-
none');
            whatsappFields.classList.remove('d-none');
        }
    }

    // Panggil toggleFields() saat halaman
    dimuat
    toggleFields();
</script>
{% endblock %}
```

Penjelasan Kode:

- Penggunaan Base Template

Pengenalan Bot dan Menyiapkan Halaman Dasar

Template ini memperluas (`extends`) template dasar `base.html`, yang berarti template ini mewarisi struktur halaman yang sudah ditentukan sebelumnya. Ini mempermudah pengelolaan tata letak yang konsisten di seluruh halaman aplikasi kita. Semua konten spesifik untuk halaman pembuatan bot akan dimasukkan ke dalam blok `{% block content %}`.

- **Formulir Pembuatan Bot**

Formulir ini dirancang menggunakan elemen-elemen HTML sederhana yang disesuaikan dengan framework Bootstrap agar tampil lebih profesional dan responsif. Pengguna dapat memilih platform bot yang akan mereka buat (Telegram atau WhatsApp) dari dropdown `select`. Setelah platform dipilih, field yang relevan akan muncul. Misalnya, jika pengguna memilih Telegram, field untuk memasukkan token Telegram akan ditampilkan, sedangkan jika WhatsApp dipilih, field yang relevan dengan WhatsApp (seperti Account SID, Auth Token, dan nomor WhatsApp) akan muncul.

- **Penggunaan Token CSRF**

Token CSRF (`{% csrf_token %}`) adalah elemen keamanan penting dalam setiap form di Django. Token ini mencegah serangan CSRF (Cross-Site Request Forgery) dengan memastikan bahwa hanya form yang valid yang dapat mengirimkan data.

- **Validasi Sederhana dengan JavaScript**

Template ini menggunakan fungsi JavaScript sederhana, `togglePlatformFields()`, untuk menentukan

Pengenalan Bot dan Menyiapkan Halaman Dasar

field mana yang akan ditampilkan berdasarkan platform yang dipilih pengguna. Ketika pengguna memilih Telegram, field token Telegram akan muncul, dan ketika mereka memilih WhatsApp, field yang relevan dengan WhatsApp akan muncul. Logika ini diimplementasikan dengan menampilkan atau menyembunyikan elemen HTML menggunakan properti `display`.

- **Tombol Submit**

Setelah pengguna mengisi semua informasi yang diperlukan, mereka dapat mengirimkan form dengan menekan tombol "Buat Bot". Form ini akan dikirim ke server, dan data yang diisi pengguna akan diproses oleh view yang sudah kita siapkan sebelumnya.

Dalam bagian ini, kita telah membuat template `create_bot.html` yang berfungsi sebagai antarmuka pembuatan bot. Template ini memungkinkan pengguna untuk memilih platform yang akan digunakan (Telegram atau WhatsApp) dan mengisi informasi yang dibutuhkan, seperti token atau kredensial lain. Dengan menggunakan JavaScript sederhana, kita dapat membuat antarmuka yang interaktif dan dinamis, sehingga pengguna hanya melihat field yang relevan dengan platform yang mereka pilih. Template ini juga sudah dilengkapi dengan mekanisme keamanan melalui token CSRF, yang memastikan form hanya dapat diakses dan digunakan oleh pengguna yang valid.

Pengenalan Bot dan Menyiapkan Halaman Dasar

Di tahap ini, kita sudah berhasil membangun fondasi antarmuka pembuatan bot yang aman dan intuitif. Fungsionalitas lebih lanjut seperti integrasi webhook dan pencatatan aktivitas akan kita lanjutkan pada bab berikutnya.

8.3.3 Mengatur URL Routing untuk Membuat Bot

Langkah terakhir adalah mengatur URL routing agar pengguna dapat mengakses halaman pembuatan bot melalui alamat yang sesuai di aplikasi web kita.

Buka file `apps/bots/urls.py` dan tambahkan rute berikut:

```
# File: apps/bots/urls.py
from django.urls import path
from . import views

urlpatterns = [
    path('create/', views.create_bot,
        name='create_bot'), # URL untuk halaman
        pembuatan bot
    ]
```

Penjelasan Kode:

- `path('create/', views.create_bot, name='create_bot')`: URL ini akan menghubungkan halaman pembuatan bot dengan view yang telah kita buat sebelumnya (`create_bot`). Pengguna dapat mengakses halaman ini melalui `/create/`.

Pengenalan Bot dan Menyiapkan Halaman Dasar

Selanjutnya, kita perlu memastikan bahwa URL aplikasi bots termasuk dalam routing utama proyek. Buka file `platform_bot/urls.py` dan tambahkan konfigurasi berikut:

```
# File: platform_bot/urls.py

from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('apps.bots.urls')), #
    Menyertakan URL aplikasi bots
]
```

Dengan konfigurasi ini, halaman bots akan ditampilkan ketika pengguna mengunjungi URL root dari aplikasi.

Dengan demikian, halaman untuk membuat bot sudah bisa diakses dan pengguna dapat mengisi formulir pembuatan bot yang telah disiapkan.

Pada bagian ini, kita telah berhasil membuat halaman untuk pembuatan bot, lengkap dengan view, template, dan URL routing. Meskipun fungsionalitas pembuatan bot belum sepenuhnya selesai (karena nanti akan ada penanganan lebih lanjut seperti web-hook), fondasi untuk mengelola pembuatan bot sudah siap.

8.4 Membuat Halaman Untuk Manage Bot

Pada bagian ini, kita akan membuat halaman untuk manajemen bot, di mana pengguna dapat melihat daftar bot yang telah mereka buat dan melakukan tindakan seperti menghapus bot. Sama seperti pada bagian sebelumnya, kita akan menyusun view, template, dan URL routing untuk halaman ini agar pengguna bisa mengelola bot mereka dengan mudah.

Halaman ini akan berfungsi sebagai pusat kontrol bagi pengguna untuk mengelola bot yang mereka buat. Di sini, mereka dapat melihat daftar bot, menghapus bot yang tidak diperlukan lagi, dan melakukan tindakan lain di masa mendatang.

8.4.1 Membuat View untuk Manage Bot

Setelah kita membuat template pembuatan bot, langkah berikutnya adalah menyediakan halaman bagi pengguna untuk mengelola bot yang telah mereka buat. Dalam proses ini, kita akan membuat sebuah view yang bertugas menampilkan daftar bot yang dimiliki pengguna, sekaligus menangani berbagai aksi seperti penghapusan bot.

View ini akan mengambil data bot yang dimiliki oleh pengguna yang sedang login, menampilkannya dalam bentuk tabel atau kartu, dan memungkinkan pengguna untuk melakukan tindakan tertentu seperti menghapus bot. Selain itu, view ini harus dilengkapi dengan mekanisme keamanan yang mencegah akses atau

Pengenalan Bot dan Menyiapkan Halaman Dasar

penghapusan data bot oleh pengguna yang tidak berwenang. Untuk itu, kita akan menggunakan beberapa fitur Django seperti `login_required` dan `get_object_or_404`.

Pertama, buka file `views.py` di dalam direktori `apps/bots/`. Di sinilah kita akan menulis view untuk halaman manajemen bot. Pastikan file ini telah diatur dengan baik, dan tambahkan kode berikut:

```
# File: apps/bots/views.py

@login_required # Pastikan hanya pengguna yang
login yang bisa mengakses view ini
def manage_bots(request):
    bots = Bot.objects.filter(user=request.user)
    # Ambil daftar bot milik pengguna yang sedang
    login

    if request.method == 'POST':
        if 'delete' in request.POST: # Cek
            apakah ada permintaan untuk menghapus bot
            bot_id = request.POST.get('delete')
            # Ambil ID bot yang ingin dihapus dari form
            bot = get_object_or_404(Bot,
            id=bot_id) # Cari bot berdasarkan ID, atau
            tampilkan 404 jika tidak ditemukan
            bot.delete() # Hapus bot dari
            database

        return redirect('manage_bots') #
        Setelah bot dihapus, kembali ke halaman
        manajemen bot
```

Pengenalan Bot dan Menyiapkan Halaman Dasar

```
return render(request,
'bots/manage_bots.html', {'bots': bots}) #
Render halaman dengan daftar bot yang dimiliki
pengguna
```

Penjelasan Kode:

- **login_required**

View `manage_bots` dilindungi oleh dekorator `login_required`, yang memastikan bahwa hanya pengguna yang sudah login yang bisa mengakses halaman ini. Jika ada pengguna yang mencoba mengakses halaman ini tanpa login, mereka akan diarahkan ke halaman login terlebih dahulu. Fitur ini sangat penting untuk menjaga keamanan aplikasi, terutama ketika mengelola data sensitif seperti bot yang telah dibuat oleh pengguna.

- **Mengambil Daftar Bot**

Kode `bots = Bot.objects.filter(user=request.user)` bertanggung jawab untuk mengambil semua bot yang dimiliki oleh pengguna yang sedang login. Dengan menggunakan filter berdasarkan atribut `user`, hanya bot yang relevan dengan pengguna tersebut yang akan ditampilkan di halaman manajemen. Hal ini penting karena kita tidak ingin pengguna melihat atau mengelola bot milik pengguna lain.

- **Penghapusan Bot**

Pengenalan Bot dan Menyiapkan Halaman Dasar

Pada bagian ini, kita menangani logika penghapusan bot. Jika pengguna menekan tombol hapus pada salah satu bot di halaman, permintaan POST akan dikirim ke view ini. View kemudian memeriksa apakah ada parameter `delete` dalam permintaan POST dengan menggunakan `if 'delete' in request.POST`. Jika parameter tersebut ada, artinya pengguna berniat untuk menghapus bot tertentu. ID bot yang ingin dihapus diambil menggunakan `request.POST.get('delete')`.

Fungsi `get_object_or_404` kemudian digunakan untuk mencari bot yang sesuai dengan ID tersebut di database. Jika bot ditemukan, maka bot tersebut dihapus dari database dengan perintah `bot.delete()`. Jika tidak, pengguna akan diarahkan ke halaman 404, yang memastikan bahwa hanya bot yang valid yang dapat dihapus. Setelah bot dihapus, pengguna akan di-redirect kembali ke halaman manajemen bot menggunakan `redirect('manage_bots')`.

- **Merender Template**

Setelah semua data bot berhasil diambil dari database, view akan merender template `manage_bots.html` menggunakan perintah `render(request, 'bots/manage_bots.html', {'bots': bots})`. Data yang telah diambil (dalam hal ini daftar bot) dikirim ke template sebagai konteks, sehingga daftar bot tersebut dapat ditampilkan kepada pengguna.

Pengenalan Bot dan Menyiapkan Halaman Dasar

Dengan menggunakan pendekatan ini, kita memastikan bahwa halaman manajemen bot dapat menampilkan daftar bot dengan aman dan memberikan pengguna kemampuan untuk menghapus bot yang sudah tidak diinginkan.

Dalam sub-bab ini, kita telah membuat view `manage_bots` yang memungkinkan pengguna untuk mengelola bot yang mereka miliki. View ini memuat daftar bot berdasarkan pengguna yang sedang login, dan memberikan fungsi penghapusan bot yang aman melalui mekanisme POST. Dengan menggunakan `login_required` dan `get_object_or_404`, kita memastikan bahwa hanya pengguna yang berhak yang dapat mengakses halaman ini dan bahwa bot yang dihapus adalah bot yang valid.

Langkah berikutnya adalah membuat template untuk menampilkan daftar bot tersebut, yang akan kita bahas di sub-bab selanjutnya. Template ini akan menampilkan data bot dalam format yang rapi dan mudah dibaca, serta menyediakan tombol untuk menghapus bot.

8.4.2 Membuat Template untuk Manage Bot

Setelah view untuk mengelola bot siap, langkah selanjutnya adalah membuat template yang akan menampilkan daftar bot milik pengguna. Template ini akan berfungsi sebagai antarmuka bagi pengguna untuk melihat informasi bot mereka, seperti nama, deskripsi, token, status, serta memberikan opsi untuk mengedit atau menghapus bot.

Pengenalan Bot dan Menyiapkan Halaman Dasar

Template ini dirancang agar mudah dibaca dan digunakan, dengan menggunakan elemen HTML seperti tabel untuk menampilkan data secara terstruktur. Kita juga akan memanfaatkan fitur-fitur Bootstrap, seperti tombol dan tabel yang distil dengan baik untuk tampilan yang lebih profesional.

Buka file `manage_bots.html` yang terletak di dalam direktori `apps/templates/bots/`. Di sinilah kita akan menulis kode HTML untuk menampilkan daftar bot. Berikut adalah kode untuk template ini:

```
<!-- File: apps/templates/bots/manage_bots.html -->

{% extends 'dashboard/base.html' %} <!--
Menggunakan template dasar dari dashboard -->

{% block content %}
  <div class="container">
    <h1>Manage Bots</h1> <!-- Judul halaman -->

    <!-- Tabel untuk menampilkan daftar bot -->
    <table class="table table-striped">
      <thead>
        <tr>
          <th>ID</th> <!-- Kolom ID bot -->
          <th>Name</th> <!-- Kolom nama bot -->
          <th>Platform</th> <!-- Kolom platform
bot -->
          <th>Token</th> <!-- Kolom token bot
-->
          <th>Status</th> <!-- Kolom status bot
-->
```

Pengenalan Bot dan Menyiapkan Halaman Dasar

```
        <th>Actions</th> <!-- Kolom untuk
tindakan seperti edit atau hapus -->
    </tr>
</thead>
<tbody>
    {% for bot in bots %} <!-- Looping
untuk menampilkan setiap bot yang dimiliki
pengguna -->
        <tr>
            <td>{{ bot.id }}</td> <!--
Menampilkan ID bot -->
            <td>{{ bot.name }}</td> <!--
Menampilkan nama bot -->
            <td>{{ bot.bot_type }}</td> <!--
Menampilkan tipe atau platform bot -->
            <td>{{ bot.token_telegram }}</td>
<!-- Menampilkan token bot -->
            <td>{{ bot.is_active|
yesno:"Active,Inactive" }}</td> <!--
Menampilkan status aktif atau tidaknya bot -->
            <td>
                <a href="" class="btn btn-
primary">Edit</a> <!-- Tombol untuk mengedit
bot -->

                <!-- Form untuk menghapus bot -->
                <form method="post"
style="display:inline;">
                    {% csrf_token %} <!-- Token
CSRF untuk keamanan form -->
                    <button type="submit"
name="delete" value="{{ bot.id }}"
onclick="return confirm('Are you sure you want
to delete this bot?');" class="btn btn-
danger">Delete</button> <!-- Tombol hapus
dengan konfirmasi -->
                </form>
            </td>
```

Pengenalan Bot dan Menyiapkan Halaman Dasar

```
        </tr>
        {% empty %} <!-- Jika tidak ada bot
yang tersedia -->
        <tr>
            <td colspan="6">No bots
available.</td> <!-- Pesan yang ditampilkan
jika tidak ada bot -->
        </tr>
        {% endfor %}
    </tbody>
</table>

    <!-- Tombol untuk membuat bot baru -->
    <a href="{% url 'create_bot' %}" class="btn
btn-primary">Buat Bot Baru</a>
</div>
{% endblock %}
```

Penjelasan Kode:

- **{% for bot in bots %}**

Loop ini digunakan untuk menampilkan daftar bot yang dimiliki oleh pengguna. Setiap bot yang diambil dari view `manage_bots` akan ditampilkan di dalam tabel dengan kolom-kolom yang relevan, seperti ID, nama, platform, token, dan status bot. Kita juga menyediakan opsi bagi pengguna untuk mengedit atau menghapus setiap bot.

- **Tabel Daftar Bot**

Tabel digunakan untuk menyusun informasi setiap bot secara rapi. Kolom-kolom dalam tabel ini menampilkan informasi penting seperti ID bot, nama bot, jenis platform (misalnya Telegram atau WhatsApp), token

Pengenalan Bot dan Menyiapkan Halaman Dasar

yang digunakan oleh bot, serta status bot (aktif atau tidak). Dengan menggunakan kelas Bootstrap seperti `table` dan `table-striped`, tabel terlihat lebih profesional dan rapi.

- **Formulir Hapus**

Setiap baris dalam tabel juga dilengkapi dengan tombol hapus yang diwakili oleh sebuah formulir POST. Saat pengguna mengklik tombol hapus, form ini akan mengirimkan ID bot yang ingin dihapus ke view `manage_bots`. Fitur `csrf_token` disertakan untuk keamanan, dan JavaScript sederhana digunakan untuk menampilkan pesan konfirmasi sebelum menghapus bot.

- **Pesan Jika Tidak Ada Bot**

Jika pengguna belum memiliki bot, bagian `{% empty %}` akan dieksekusi dan menampilkan pesan "No bots available." Ini memastikan halaman tetap informatif meskipun daftar bot kosong.

- **Link Buat Bot Baru**

Di bagian bawah halaman, terdapat tautan yang memungkinkan pengguna untuk membuat bot baru. Tautan ini mengarahkan pengguna ke halaman `create_bot.html`, di mana mereka dapat membuat bot baru dengan platform dan informasi yang diinginkan.

Template `manage_bots.html` ini memberikan tampilan yang jelas dan mudah digunakan bagi pengguna untuk mengelola bot mereka. Dengan adanya tabel yang terstruktur, pengguna dapat

Pengenalan Bot dan Menyiapkan Halaman Dasar

melihat informasi bot secara detail dan melakukan tindakan seperti mengedit atau menghapus bot dengan mudah. Selain itu, fitur keamanan seperti token CSRF dan konfirmasi JavaScript memastikan bahwa tindakan penghapusan bot dilakukan dengan aman.

Langkah selanjutnya adalah menghubungkan view dan template ini dengan URL routing agar halaman ini bisa diakses melalui browser. Sub-bab berikutnya akan membahas bagaimana kita menambahkan URL untuk halaman manajemen bot dalam aplikasi Django kita.

8.4.3 Mengatur URL Routing untuk Manage Bot

Terakhir, kita perlu mengatur routing agar halaman manajemen bot bisa diakses oleh pengguna. Kita akan menambahkan rute untuk halaman ini dalam file `apps/bots/urls.py`.

Buka file `apps/bots/urls.py` dan tambahkan rute berikut:

```
# File: apps/bots/urls.py
from django.urls import path
from . import views

urlpatterns = [
    # URL untuk halaman pembuatan bot
    path('manage/', views.manage_bots,
name='manage_bots'), # URL untuk halaman
manajemen bot
]
```

Pengenalan Bot dan Menyiapkan Halaman Dasar

Penjelasan Kode:

- `path('manage/', views.manage_bots, name='manage_bots')`: URL ini menghubungkan halaman manajemen bot dengan view `manage_bots`. Pengguna dapat mengakses halaman ini melalui `/manage/`.

Dengan routing ini, pengguna bisa mengakses halaman manajemen bot dengan mengetikkan URL `/manage/` di browser mereka.

Pada bagian ini, kita telah berhasil membuat halaman manajemen bot. Pengguna dapat melihat daftar bot yang telah mereka buat dan menghapus bot yang tidak diperlukan. Dengan view, template, dan URL routing yang sudah diatur, kita kini memiliki halaman dasar untuk mengelola bot dengan baik.

8.5 Kode Lengkap

Kita akan mereview semua kode yang telah kita buat agar tidak terjadi kesalahan dalam penulisan, berikut adalah kode lengkap yang sudah kita buat sebelumnya:

8.5.1 Models.py

```
# File: apps/bots/models.py
from django.db import models
```

Pengenalan Bot dan Menyiapkan Halaman Dasar

```
from django.contrib.auth.models import User

class Bot(models.Model):
    BOT_CHOICES = [
        ('telegram', 'Telegram'),
        ('whatsapp', 'WhatsApp'),
    ]
    user = models.ForeignKey(User,
on_delete=models.CASCADE) # Bot akan terkait
dengan pengguna
    bot_type = models.CharField(max_length=20,
choices=BOT_CHOICES) # Jenis bot: Telegram atau
WhatsApp

    token_telegram =
models.CharField(max_length=255, unique=True,
help_text="Masukkan token bot yang unik.") #
Token Telegram
    webhook_url =
models.URLField(max_length=255, blank=True,
help_text="URL webhook bot akan diatur secara
otomatis.") # URL webhook

    is_active =
models.BooleanField(default=True,
help_text="Status bot aktif atau tidak.") #
Status aktif atau tidaknya bot

    # Pesan otomatis
    start_message = models.TextField(blank=True,
default="Selamat datang di bot!",
help_text="Pesan yang dikirim saat pengguna
mengetik /start.")
    help_message = models.TextField(blank=True,
default="Berikut cara menggunakan bot.",
```

Pengenalan Bot dan Menyiapkan Halaman Dasar

```
help_text="Pesan yang dikirim saat pengguna
mengetik /help.")

    messages = models.JSONField(default=list,
blank=True, help_text="Daftar pesan yang
diterima dan responsnya dalam format JSON.") #
Format JSON

    # Timestamp
    created_at =
models.DateTimeField(auto_now_add=True) #
Tanggal bot dibuat
    updated_at =
models.DateTimeField(auto_now=True) # Tanggal
terakhir bot diperbarui

    name = models.CharField(max_length=255,
blank=True, default="Bot Saya", help_text="Nama
bot.") # Nama bot
    description = models.TextField(blank=True,
default="Ini bot pertama saya",
help_text="Deskripsi bot.") # Deskripsi bot

    def __str__(self):
        return f'{self.bot_type.capitalize()}
Bot - {self.user.username}'

    class Meta:
        verbose_name = "Bot"
        verbose_name_plural = "Bots"
```

8.5.2 Forms.py

```
# File: apps/bots/forms.py

from django import forms
from .models import Bot # Mengimpor model Bot
```

Pengenalan Bot dan Menyiapkan Halaman Dasar

```
class BotForm(forms.ModelForm):
    class Meta:
        model = Bot # Menghubungkan formulir
dengan model Bot
        fields = ['bot_type', 'token_telegram',
'name', 'description', 'is_active',
'start_message', 'help_message'] # Field yang
ditampilkan dalam formulir
        widgets = {
            'name':
forms.TextInput(attrs={'class': 'form-control',
'placeholder': 'Masukkan nama bot'}),
            'description':
forms.Textarea(attrs={'class': 'form-control',
'rows': 3, 'placeholder': 'Deskripsi bot'}),
        }
        help_texts = {
            'token_telegram': 'Masukkan token
unik yang diterima dari platform Telegram.',
            'name': 'Nama bot dapat diubah
sesuai keinginan.',
            'description': 'Deskripsi singkat
tentang bot dan fungsinya.',
        }
```

8.5.3 Views.py

```
# File: apps/bots/views.py

from django.shortcuts import render,
redirect, get_object_or_404
from django.contrib.auth.decorators import
login_required
from .forms import BotForm
from .models import Bot
```

Pengenalan Bot dan Menyiapkan Halaman Dasar

```
@login_required
def create_bot(request):
    """
    Tampilan untuk membuat bot baru. Belum
    terhubung ke webhook dan activity log.
    """
    if request.method == 'POST':
        form = BotForm(request.POST)
        if form.is_valid():
            bot = form.save(commit=False)
            bot.user = request.user # Menyimpan
            bot dengan informasi pengguna
            bot.save() # Simpan bot ke database

            # Sementara, tidak ada integrasi
            dengan webhook atau activity log.
            # Webhook dan logging akan
            diintegrasikan di bab selanjutnya.

            return redirect('manage_bots') #
            Setelah sukses, alihkan ke halaman manajemen bot
        else:
            form = BotForm()

            return render(request,
                'bots/create_bot.html', {'form': form})

@login_required # Pastikan hanya pengguna yang
login yang bisa mengakses view ini
def manage_bots(request):
    bots = Bot.objects.filter(user=request.user)
    # Ambil daftar bot milik pengguna yang sedang
    login

    if request.method == 'POST':
```

Pengenalan Bot dan Menyiapkan Halaman Dasar

```
        if 'delete' in request.POST: # Cek
            apakah ada permintaan untuk menghapus bot
            bot_id = request.POST.get('delete')
        # Ambil ID bot yang ingin dihapus dari form
        bot = get_object_or_404(Bot,
            id=bot_id) # Cari bot berdasarkan ID, atau
            tampilkan 404 jika tidak ditemukan
        bot.delete() # Hapus bot dari
        database
        return redirect('manage_bots') #
        Setelah bot dihapus, kembali ke halaman
        manajemen bot

    return render(request,
        'bots/manage_bots.html', {'bots': bots}) #
        Render halaman dengan daftar bot yang dimiliki
        pengguna
```

8.5.4 URLS.py

```
# File: apps/bots/urls.py
from django.urls import path
from . import views

urlpatterns = [
    path('create/', views.create_bot,
        name='create_bot'), # URL untuk halaman
        pembuatan bot
    path('manage/', views.manage_bots,
        name='manage_bots'), # URL untuk halaman
        manajemen bot
]
```

Pengenalan Bot dan Menyiapkan Halaman Dasar

Kesimpulan Bab

Bab ini telah menjelaskan langkah-langkah penting dalam menyiapkan halaman manajemen bot pada aplikasi Django. Kita mulai dengan menyiapkan model dan formulir untuk bot, di mana model berfungsi untuk mendefinisikan struktur data bot, dan formulir memungkinkan input data dari pengguna dengan mudah. Selanjutnya, kita membahas pembuatan halaman untuk membuat, mengedit, dan mengelola bot, yang melibatkan pembuatan view, template, serta pengaturan URL routing yang sesuai.

Di setiap bagian, dijelaskan bagaimana kita bisa menghubungkan frontend dan backend, serta bagaimana mengintegrasikan bot dengan webhook untuk memastikan komunikasi berjalan dengan baik. Pada akhir bab ini, seluruh kode yang telah dibahas dikumpulkan dalam satu bagian untuk memudahkan pemahaman secara keseluruhan. Dengan mengikuti panduan dalam bab ini, kita dapat membangun halaman manajemen bot yang fungsional dan terstruktur dengan baik dalam aplikasi Django.

Dengan semua komponen ini, kita telah membangun dasar yang kuat untuk halaman manajemen bot. Pengguna sekarang dapat melihat dan mengelola bot mereka dengan cara yang intuitif.

Langkah selanjutnya mungkin termasuk menambahkan fitur tambahan seperti menampilkan lebih banyak informasi tentang bot, atau memperluas fungsionalitas halaman ini untuk mencakup fitur lain yang relevan dengan kebutuhan pengguna.

Pengenalan Bot dan Menyiapkan Halaman Dasar

Secara keseluruhan, Di bagian ini telah memberikan panduan komprehensif tentang cara membuat dan mengelola halaman manajemen bot dalam aplikasi Django. Dengan fondasi ini, Anda kini siap untuk mengembangkan lebih lanjut dan menambahkan fitur-fitur tambahan sesuai dengan kebutuhan proyek Anda.

BAB 9 - Validasi Token untuk Bot

Pada bagian ini, kita akan membahas secara mendalam mengenai proses validasi token yang digunakan untuk bot, serta bagaimana cara menerapkannya secara efektif dalam aplikasi. Validasi token merupakan langkah penting untuk memastikan bahwa hanya token yang sah dan sesuai dengan format yang ditentukan yang dapat digunakan untuk membuat dan mengelola bot. Dengan validasi yang baik, kita bisa mencegah kesalahan dan menjaga integritas sistem, sehingga bot yang dibuat dapat berfungsi dengan optimal dan aman. Mari kita mulai dengan memahami apa itu token, pentingnya validasi, dan bagaimana kita dapat mengimplementasikannya di aplikasi kita.

9.1 Pendahuluan

Pembuatan bot yang efektif dalam aplikasi web memerlukan pemahaman yang mendalam tentang token yang digunakan untuk otentikasi dan identifikasi. Token adalah string unik yang berfungsi sebagai kunci akses untuk menghubungkan bot dengan platform tertentu, seperti Telegram, Discord, atau platform lainnya. Dalam konteks ini, validasi token menjadi salah satu langkah penting untuk memastikan bahwa hanya token yang sah dan sesuai format yang diterima saat membuat bot.

Validasi token tidak hanya melindungi sistem dari input yang tidak valid, tetapi juga meningkatkan keamanan aplikasi dengan memastikan bahwa hanya pengguna yang memiliki token yang

Validasi Token untuk Bot

benar yang dapat membuat dan mengelola bot. Jika token yang dimasukkan oleh pengguna tidak sesuai atau tidak valid, proses pembuatan bot harus ditolak, dan pesan kesalahan yang jelas perlu disampaikan kepada pengguna.

Salah satu tantangan utama dalam validasi token adalah beragamnya format token yang mungkin digunakan oleh berbagai platform. Misalnya, token Telegram memiliki format tertentu yang berbeda dengan token dari platform lain. Oleh karena itu, penting bagi kita untuk memahami format masing-masing token dan cara mengimplementasikan validasi yang sesuai dalam aplikasi Django kita.

Pada sub-bab selanjutnya, kita akan mendalami lebih dalam mengenai format token yang umum digunakan, serta bagaimana implementasi validasi token dapat dilakukan di dalam aplikasi Django. Kita juga akan membahas tentang pembuatan model untuk token, serta bagaimana cara menangani kesalahan yang mungkin terjadi saat pengguna mencoba memasukkan token yang tidak valid. Melalui pembahasan ini, kita bertujuan untuk memastikan bahwa proses pembuatan bot berjalan dengan lancar dan aman, sehingga pengguna dapat berfokus pada fungsionalitas bot tanpa khawatir tentang kesalahan yang disebabkan oleh input yang tidak valid.

Dengan pemahaman yang baik tentang validasi token, kita dapat meningkatkan pengalaman pengguna dan memastikan bahwa aplikasi kita berfungsi dengan baik dalam berbagai kondisi. Selanjutnya, mari kita bahas format token yang umum digunakan un-

Validasi Token untuk Bot

tuk membantu kita memahami apa yang perlu divalidasi saat pengguna memasukkan token untuk bot mereka.

9.1.1 Pentingnya Validasi Token

Dalam pengembangan platform yang memungkinkan pengguna membuat bot, salah satu elemen terpenting yang harus diperhatikan adalah validasi token. Token berfungsi sebagai kunci otentikasi yang memungkinkan bot berkomunikasi dengan API dari platform yang digunakan, seperti Telegram atau WhatsApp. Tanpa validasi yang tepat, sistem akan rentan terhadap kesalahan input, yang tidak hanya dapat menyebabkan kegagalan operasional, tetapi juga dapat berpotensi mengganggu keamanan seluruh platform.

Validasi token adalah langkah yang sangat krusial dalam pengembangan bot, terutama ketika berinteraksi dengan berbagai platform. Token berfungsi sebagai jembatan yang menghubungkan bot kita dengan platform layanan, dan kesalahan dalam proses ini dapat berakibat fatal, mulai dari kebocoran data hingga penyalahgunaan akses. Oleh karena itu, penting bagi kita untuk memastikan bahwa token yang dimasukkan oleh pengguna memenuhi kriteria tertentu dan sah secara teknis.

Sebuah token dapat dianggap sebagai "identitas" yang dimiliki oleh bot untuk mengakses layanan eksternal. Misalnya, pada platform Telegram, setiap bot diidentifikasi dengan token unik yang diberikan oleh BotFather. Token ini harus dalam format yang sangat spesifik, menggabungkan ID bot dan kunci rahasia

Validasi Token untuk Bot

yang dipisahkan oleh tanda titik dua. Token seperti ini memastikan bahwa bot yang dihubungkan ke API adalah bot yang sah.

Namun, masalah muncul ketika pengguna salah memasukkan token atau bahkan mencoba memasukkan nilai yang tidak valid. Misalnya, jika pengguna memasukkan teks acak atau string yang tidak mengikuti format yang diharapkan, API Telegram tidak akan dapat mengenali permintaan dari bot tersebut. Sebagai akibatnya, koneksi antara bot dan API gagal, dan fungsi bot tidak akan berjalan sebagaimana mestinya. Hal yang sama berlaku untuk platform lain seperti WhatsApp melalui Twilio, di mana parameter seperti `account_sid`, `auth_token`, dan `whatsapp_number` harus sesuai dengan format yang ditentukan.

Selain itu, validasi token bukan hanya tentang memastikan format yang benar, tetapi juga tentang menjaga keamanan. Sebuah token yang salah atau sembarangan dapat menjadi celah keamanan bagi sistem. Misalnya, jika tidak ada validasi, pengguna bisa saja memasukkan token yang berpotensi merusak atau mengakses data yang tidak seharusnya. Ini akan membuka pintu bagi berbagai ancaman, termasuk penyalahgunaan API atau bahkan serangan denial of service (DoS) yang bisa memperlambat atau menghentikan seluruh sistem.

Di sinilah pentingnya validasi token menjadi sangat krusial. Dalam sistem yang kita bangun, validasi token dilakukan baik di sisi server maupun di sisi klien. Ini artinya, sebelum token disimpan ke dalam basis data, sistem harus memastikan bahwa token

Validasi Token untuk Bot

tersebut memenuhi syarat dari platform yang terkait, baik itu Telegram atau WhatsApp. Selain itu, sistem juga harus memberikan umpan balik yang jelas kepada pengguna jika token yang mereka masukkan tidak valid.

Misalnya, jika pengguna mencoba memasukkan token Telegram dalam format yang salah seperti `"ini_token_salah"`, sistem akan memberikan pesan error yang informatif, seperti "Token Telegram yang Anda masukkan tidak valid. Pastikan token berformat seperti ini: `5898177748:AAFBMw_gSELJWsFvvv0Fu-eRGQ459cHSaNCk`." Dengan demikian, pengguna dapat segera memperbaiki kesalahan mereka tanpa kebingungan.

Di Django, validasi token dapat diterapkan baik di level model maupun form. Validasi di model memastikan bahwa data yang dimasukkan ke dalam database sudah dalam format yang benar, sedangkan validasi di form membantu memberikan umpan balik lebih cepat kepada pengguna saat mereka sedang mengisi formulir. Validasi ini bisa dilakukan dengan menambahkan validasi kustom di `models.py` atau `forms.py`, tergantung pada pendekatan yang kita pilih.

Sebagai contoh, jika kita memilih untuk melakukan validasi di form, kita bisa mendefinisikan sebuah metode `clean` di `forms.py` untuk mengecek apakah token yang dimasukkan pengguna sesuai dengan format yang diperlukan. Misalnya, untuk token Telegram, kita bisa memeriksa apakah token terdiri dari dua bagian yang dipisahkan oleh tanda titik dua, dan apakah bagian kedua memiliki panjang yang tepat untuk sebuah kunci rahasia.

Validasi Token untuk Bot

```
# File: forms.py
from django import forms
import re

class TelegramTokenForm(forms.Form):
    token = forms.CharField(max_length=255,
                           required=True)

    def clean_token(self):
        token = self.cleaned_data.get('token')
        # Memeriksa apakah token memiliki format
        yang benar (ID:Secret Key)
        if not re.match(r'^\d{9}:[A-Za-z0-9_-]
{35}$', token):
            raise forms.ValidationError("Token
Telegram tidak valid. Pastikan token berformat
seperti ini:
5898177748:AAFBMw_gSELJWsFvvv0FueRGQ459cHSaNck")
        return token
```

Di dalam contoh di atas, kita menggunakan ekspresi reguler untuk memastikan bahwa token yang dimasukkan oleh pengguna mengikuti pola yang benar, yaitu terdiri dari ID numerik sembilan digit, diikuti oleh titik dua, lalu kunci rahasia yang memiliki 35 karakter alfanumerik. Jika token tidak sesuai dengan pola ini, Django akan memunculkan pesan kesalahan yang memberikan panduan jelas kepada pengguna.

Pendekatan validasi ini tidak hanya meningkatkan pengalaman pengguna, tetapi juga memberikan lapisan perlindungan tambahan terhadap potensi kesalahan atau ancaman keamanan yang berasal dari input yang tidak valid. Sistem validasi yang kuat sangat penting dalam menjaga keandalan dan keamanan platform,

Validasi Token untuk Bot

serta memastikan bahwa bot yang dibuat oleh pengguna dapat berjalan dengan baik dan aman.

Dengan menerapkan validasi yang baik dan menyeluruh, kita memastikan bahwa hanya token yang sah yang dapat digunakan untuk menjalankan bot di platform ini. Ini adalah langkah awal yang penting dalam membangun platform yang handal dan aman.

9.1.2 Jenis-jenis Token dan Penggunaannya

Dalam ekosistem pembuatan bot, setiap platform memiliki mekanisme otentikasi tersendiri yang umumnya diwujudkan dalam bentuk token. Token ini berperan sebagai kunci digital yang memungkinkan bot berkomunikasi dengan server platform, seperti Telegram atau WhatsApp. Setiap platform memerlukan token dengan format dan aturan yang berbeda, yang harus dipahami oleh developer untuk memastikan integrasi yang sukses.

Dalam konteks pengembangan bot, terdapat berbagai jenis token yang digunakan, masing-masing dengan fungsionalitas dan tujuan spesifik. Memahami jenis-jenis token ini sangat penting, karena akan membantu kita dalam proses validasi dan penggunaan yang tepat dalam aplikasi kita.

Salah satu jenis token yang paling umum adalah **API Token**. Token ini biasanya digunakan untuk otentikasi dalam komunikasi antara aplikasi dan server. Setiap kali bot kita melakukan permintaan ke API, seperti mengambil data atau mengirim pesan, API Token akan disertakan dalam permintaan tersebut untuk memastikan bahwa permintaan tersebut berasal dari sumber yang sah.

Validasi Token untuk Bot

Contohnya, token API Telegram yang digunakan untuk mengotentikasi bot kita saat berkomunikasi dengan server Telegram.

Token ini biasanya berupa string panjang yang mengandung kombinasi huruf dan angka. Penting untuk memastikan bahwa token ini tidak mudah ditebak dan tetap rahasia. Untuk melakukan validasi terhadap token API ini, kita perlu memeriksa format dan panjang token yang sesuai dengan spesifikasi dari platform yang bersangkutan.

Selain API Token, terdapat juga **OAuth Token**, yang sering digunakan dalam sistem yang memerlukan otorisasi pengguna. OAuth adalah protokol yang memungkinkan pengguna untuk memberikan akses ke aplikasi pihak ketiga tanpa membagikan informasi login mereka. Dalam konteks bot, setelah pengguna memberikan izin, aplikasi kita akan menerima OAuth Token yang dapat digunakan untuk mengakses data pengguna. Proses ini lebih aman dan mengurangi risiko kebocoran informasi pribadi.

Penggunaan OAuth Token membutuhkan pemahaman tentang flow otorisasi yang digunakan. Misalnya, dalam skenario aplikasi web, kita perlu mengarahkan pengguna ke halaman login platform untuk mendapatkan izin, lalu menerima token yang diperlukan. Dalam hal ini, validasi token juga menjadi krusial, karena kita perlu memastikan bahwa token tersebut masih berlaku dan memiliki izin yang tepat untuk mengakses data.

Ada juga **Bearer Token**, yang merupakan jenis token yang biasanya digunakan dalam otentikasi berbasis token. Bearer Token di-

Validasi Token untuk Bot

gunakan dalam header permintaan HTTP, memungkinkan server untuk mengidentifikasi pengguna tanpa memerlukan autentikasi tambahan. Saat menggunakan Bearer Token, penting untuk memvalidasi token tersebut di setiap permintaan untuk memastikan bahwa akses yang diberikan masih sah.

Selain itu, beberapa platform juga menggunakan **Refresh Token**. Token ini memungkinkan pengguna untuk mendapatkan token baru tanpa harus login ulang. Refresh Token biasanya memiliki masa berlaku yang lebih lama dibandingkan dengan token biasa. Dalam hal ini, validasi Refresh Token sangat penting untuk memastikan bahwa token tersebut belum kedaluwarsa dan masih dapat digunakan untuk mendapatkan akses baru.

Token Telegram

Pada platform Telegram, setiap bot diidentifikasi dan diberi izin untuk berinteraksi dengan API Telegram menggunakan token yang unik. Token ini diberikan oleh BotFather, sebuah bot khusus dari Telegram yang bertindak sebagai pengelola bot-bot lain. Saat pengguna membuat bot baru di Telegram melalui BotFather, mereka akan menerima token yang berbentuk seperti ini:

```
5898177748:AAFBMw_gSELJWsFvvv0FueRGQ459cHSaNck
```

Token ini terdiri dari dua bagian utama yang dipisahkan oleh tanda titik dua (:). Bagian pertama adalah ID bot, yang merupakan deretan angka yang mengidentifikasi bot secara unik di sistem Telegram. Bagian kedua adalah kunci rahasia yang memungkinkan bot untuk melakukan operasi seperti mengirim pesan atau menerima pembaruan dari Telegram API.

Validasi Token untuk Bot

Token Telegram digunakan dalam setiap permintaan API yang dilakukan bot. Saat bot mencoba mengirim pesan atau mendapatkan informasi dari Telegram, token ini dikirim sebagai bagian dari header permintaan untuk memastikan bahwa permintaan tersebut sah dan berasal dari bot yang diotorisasi. Sebagai contoh, jika kita ingin mengirim pesan melalui bot, kita perlu menggunakan token ini untuk mengakses endpoint API Telegram.

Sistem validasi Telegram sangat ketat, karena bot hanya akan menerima dan mengeksekusi permintaan yang dilengkapi dengan token yang valid. Jika token salah atau tidak sesuai format yang diharapkan, permintaan tersebut akan ditolak, dan bot tidak akan bisa beroperasi dengan baik.

Token WhatsApp melalui Twilio

Di sisi lain, WhatsApp tidak memiliki token otentikasi langsung seperti Telegram. Untuk mengintegrasikan bot WhatsApp, kita perlu menggunakan layanan pihak ketiga, seperti Twilio, yang bertindak sebagai penghubung antara bot dan API WhatsApp. Pada Twilio, otentikasi dilakukan dengan menggunakan tiga parameter utama: `account_sid`, `auth_token`, dan nomor WhatsApp yang terverifikasi.

- **Account SID** adalah identifier unik yang diberikan Twilio untuk setiap akun yang terdaftar. Ini berfungsi mirip dengan ID pengguna, mengidentifikasi secara unik siapa yang membuat permintaan API.
- **Auth Token** adalah kunci rahasia yang diberikan oleh Twilio untuk otorisasi. Sama halnya dengan kunci raha-

Validasi Token untuk Bot

sia pada Telegram, `auth_token` ini harus dijaga kerahasiaannya karena semua permintaan API yang dilakukan bot akan memerlukan token ini untuk verifikasi.

- **Nomor WhatsApp** adalah nomor telepon yang sudah terdaftar di Twilio dan diizinkan untuk digunakan dalam mengirim dan menerima pesan WhatsApp melalui API Twilio. Nomor ini harus dalam format internasional, misalnya:

```
+14155238886
```

Ketiga parameter ini (`account_sid`, `auth_token`, dan nomor WhatsApp) digunakan bersama-sama untuk melakukan otentikasi setiap permintaan yang dilakukan oleh bot ke API WhatsApp. Saat bot mencoba mengirim pesan melalui WhatsApp, Twilio akan memverifikasi apakah permintaan tersebut datang dari akun yang valid dan terautentikasi dengan benar menggunakan kombinasi `account_sid` dan `auth_token`.

Penggunaan Token dalam Otentikasi

Baik di Telegram maupun WhatsApp melalui Twilio, token memainkan peran krusial dalam menjaga keamanan komunikasi antara bot dan API. Setiap permintaan yang dikirim oleh bot harus dilengkapi dengan token atau parameter otentikasi yang valid. Jika token tidak valid atau sudah kadaluarsa, permintaan tersebut akan ditolak oleh server API.

Untuk Telegram, otentikasi dilakukan sepenuhnya melalui token yang dihasilkan oleh BotFather. Setiap kali bot mengirimkan permintaan API, seperti mengirim pesan atau mendapatkan pemba-

Validasi Token untuk Bot

ruan, token ini harus dimasukkan dalam setiap permintaan. Telegram akan memeriksa apakah token tersebut valid sebelum memproses permintaan.

Untuk WhatsApp melalui Twilio, otentikasi dilakukan dengan memeriksa kombinasi `account_sid`, `auth_token`, dan nomor WhatsApp. Ketiga parameter ini harus sesuai dengan yang terdaftar di akun Twilio, dan semua permintaan API yang diajukan oleh bot akan diverifikasi menggunakan data ini.

Penting untuk dicatat bahwa menjaga keamanan token adalah hal yang sangat vital. Jika token ini jatuh ke tangan yang salah, pihak yang tidak berwenang dapat menyalahgunakannya untuk mengakses API atau bahkan menjalankan bot atas nama pengguna. Oleh karena itu, dalam pengembangan platform bot, kita harus memastikan bahwa token selalu disimpan dengan aman, dan validasi yang kuat diterapkan untuk memeriksa keabsahan token tersebut.

Dalam implementasi Django, kita dapat memastikan validasi ini dilakukan baik di level model maupun form, untuk menjamin bahwa token yang dimasukkan oleh pengguna memiliki format yang benar dan sesuai dengan yang diharapkan oleh API dari platform yang digunakan. Misalnya, dengan menambahkan logika validasi di `models.py` atau `forms.py`, kita dapat memastikan bahwa token yang diterima sudah sesuai sebelum disimpan ke dalam database atau digunakan untuk memproses permintaan API.

Validasi Token untuk Bot

Validasi ini tidak hanya menjaga agar bot berfungsi dengan baik, tetapi juga melindungi sistem dari penyalahgunaan token yang tidak sah.

Memahami berbagai jenis token ini akan membantu kita dalam merancang sistem validasi yang lebih efektif dan aman. Setiap jenis token memiliki karakteristik dan aturan sendiri yang harus dipatuhi. Dengan pemahaman yang mendalam tentang bagaimana dan kapan setiap token digunakan, kita dapat membangun aplikasi bot yang tidak hanya fungsional tetapi juga aman dan andal. Pada sub-bab berikutnya, kita akan membahas cara menerapkan validasi token dalam kode Django kita, sehingga kita dapat memastikan bahwa bot yang dibuat oleh pengguna memenuhi semua persyaratan yang telah ditetapkan.

9.2 Memahami Format Token

Token adalah elemen penting dalam sistem otentikasi bot, khususnya pada platform seperti Telegram dan WhatsApp. Setiap token memiliki format yang berbeda-beda sesuai dengan kebutuhan dan standar platformnya. Memahami format token adalah langkah awal yang sangat penting dalam memastikan bahwa validasi token berjalan dengan benar. Pada subbab ini, kita akan mengulas format token yang digunakan oleh Telegram secara detail. Pemahaman yang mendalam tentang struktur token ini akan membantu kita dalam melakukan validasi di aplikasi Django, memastikan setiap token yang dimasukkan oleh pengguna sesuai dengan standar platform yang digunakan.

9.2.1 Format Token Telegram

Dalam pengembangan bot Telegram, format token yang digunakan sangat penting untuk memastikan bot dapat berfungsi dengan baik. Token Telegram adalah salah satu elemen utama yang menghubungkan bot dengan API Telegram. Token ini diberikan oleh BotFather saat developer membuat bot baru. Setiap token memiliki format yang unik dan terdiri dari dua bagian penting, yaitu `bot_id` dan `secret_key`, yang dipisahkan oleh tanda titik dua (:).

Sebagai contoh, token Telegram biasanya terlihat seperti ini:

```
5898177748:AAFBMw_gSELJwsFvvv0FueRGQ459cHSaNCk
```

Penjelasan Struktur Token

1. *Bagian pertama (bot_id)*

Bagian ini adalah serangkaian angka yang mewakili ID bot di sistem Telegram. ID ini merupakan identifikasi unik yang diberikan kepada bot yang dibuat oleh pengguna. Contoh dari `bot_id` di token di atas adalah 5898177748. Setiap bot yang dibuat melalui BotFather akan mendapatkan `bot_id` yang berbeda. `Bot_id` ini penting karena menjadi salah satu kunci dalam proses pengenalan bot oleh Telegram.

2. *Bagian kedua (secret_key)*

Bagian ini adalah kunci rahasia yang digunakan untuk mengamankan bot. `Secret_key` diberikan oleh Telegram dan harus dijaga kerahasiaannya karena kunci ini yang memungkinkan bot untuk melakukan operasi di API Te-

Validasi Token untuk Bot

Telegram. `Secret_key` biasanya terdiri dari huruf besar, huruf kecil, angka, dan beberapa karakter khusus seperti garis bawah (`_`). Pada token contoh di atas, `secret_key`-nya adalah `AAFBMw_gSELJWsFvvv0FueRGQ459cH-SaNck`.

Mengapa Format Ini Penting?

Telegram menggunakan kedua bagian ini untuk memastikan bahwa hanya bot yang diotorisasi dengan benar yang dapat mengakses API mereka. Ketika bot mengirimkan permintaan ke API Telegram, token ini akan digunakan untuk memverifikasi identitas bot. Telegram kemudian akan memvalidasi apakah `bot_id` dan `secret_key` tersebut sesuai dan valid. Jika token yang dimasukkan salah, API akan mengembalikan kesalahan dan bot tidak akan bisa beroperasi dengan benar.

Sangat penting untuk memastikan bahwa token ini disimpan dengan aman dan tidak dibagikan kepada orang lain. Jika token ini jatuh ke tangan yang salah, orang tersebut dapat mengendalikan bot dan melakukan tindakan yang tidak diinginkan. Oleh karena itu, kita perlu menerapkan langkah-langkah validasi untuk memastikan bahwa token yang diberikan oleh pengguna sesuai dengan format yang diharapkan.

Untuk memvalidasi format token Telegram, kita dapat menggunakan ekspresi reguler. Ekspresi reguler memungkinkan kita untuk memeriksa apakah token yang dimasukkan mengikuti pola yang sesuai. Sebagai contoh, kita dapat menulis fungsi validasi

Validasi Token untuk Bot

dalam aplikasi Django kita untuk memeriksa format token sebagai berikut:

```
import re

def validasi_token_telegram(token):
    # Memeriksa apakah token sesuai dengan
    # format Telegram
    pola_token = r'^\d{9}: [A-Za-z0-9_-]{35}$' #
    # Format ID dan token
    return re.match(pola_token, token) is not
    None # Mengembalikan True jika cocok
```

Dalam kode di atas, kita menggunakan modul `re` untuk memeriksa apakah token yang dimasukkan sesuai dengan pola yang telah ditentukan. Jika token tersebut valid, fungsi ini akan mengembalikan nilai `True`, yang menandakan bahwa token tersebut sesuai dengan format yang diharapkan. Jika tidak, kita akan mengembalikan nilai `False`, yang menunjukkan bahwa token tersebut tidak valid.

Sebagai contoh, di aplikasi Django, kita juga bisa menambahkan logika validasi di level form untuk memastikan token memiliki format yang benar. Ini dapat dilakukan dengan memeriksa apakah token terdiri dari dua bagian yang dipisahkan oleh tanda titik dua (:), dan apakah kedua bagian tersebut memenuhi kriteria panjang dan format yang diharapkan.

```
# File: forms.py

from django import forms
from django.core.exceptions import
ValidationError
```

Validasi Token untuk Bot

```
class TelegramBotForm(forms.Form):
    token = forms.CharField(max_length=100,
                             help_text="Masukkan token bot Telegram")

    def clean_token(self):
        token = self.cleaned_data.get('token')
        if ':' not in token:
            raise ValidationError("Token harus
berisi dua bagian yang dipisahkan oleh tanda
titik dua (:)")

        bot_id, secret_key = token.split(':', 1)
        if not bot_id.isdigit():
            raise ValidationError("Bagian
pertama dari token harus berupa angka (bot_id)")
        if len(secret_key) < 35: # Panjang
secret_key bisa bervariasi
            raise ValidationError("Kunci rahasia
(secret_key) harus cukup panjang dan valid")

        return token
```

Pada kode di atas, kita memastikan bahwa token memiliki format yang benar dengan memeriksa keberadaan tanda titik dua (:), memvalidasi bahwa bot_id adalah angka, dan memastikan panjang secret_key cukup untuk menjadi kunci yang valid. Jika token tidak memenuhi kriteria tersebut, sistem akan memberikan pesan kesalahan yang jelas kepada pengguna.

Penggunaan di Level Model

Selain melakukan validasi di form, kita juga dapat menerapkan validasi di level model untuk memastikan bahwa setiap token yang disimpan di database sudah valid. Berikut adalah contoh bagaimana kita dapat menambahkan validasi di model:

Validasi Token untuk Bot

```
# File: models.py

from django.db import models
from django.core.exceptions import
ValidationError

def validate_telegram_token(token):
    if ':' not in token:
        raise ValidationError("Token harus
berisi dua bagian yang dipisahkan oleh tanda
titik dua (:)")

    bot_id, secret_key = token.split(':', 1)
    if not bot_id.isdigit():
        raise ValidationError("Bagian pertama
dari token harus berupa angka (bot_id)")
    if len(secret_key) < 35:
        raise ValidationError("Kunci rahasia
(secret_key) harus cukup panjang dan valid")

class TelegramBot(models.Model):
    name = models.CharField(max_length=100)
    token = models.CharField(max_length=100,
validators=[validate_telegram_token])

    def __str__(self):
        return self.name
```

Dengan menggunakan validasi di model seperti di atas, kita dapat memastikan bahwa token Telegram selalu mengikuti format yang benar setiap kali disimpan di database. Jika token yang dimasukkan tidak valid, maka data tidak akan disimpan dan pengguna akan mendapatkan pesan kesalahan yang relevan.

Validasi Token untuk Bot

Pemahaman mengenai format token Telegram ini tidak hanya penting dalam validasi, tetapi juga dalam memahami bagaimana Telegram melakukan otentikasi bot di platform mereka. Dalam bagian selanjutnya, kita akan membahas format token pada platform lain, seperti WhatsApp, yang memiliki pendekatan otentikasi yang berbeda namun sama pentingnya.

Dengan memahami format token Telegram dan menerapkan langkah-langkah validasi yang tepat, kita dapat memastikan bahwa bot yang dibuat oleh pengguna memiliki akses yang sah ke API Telegram. Selanjutnya, kita akan membahas bagaimana cara melakukan validasi ini dalam aplikasi Django kita dan menampilkan pesan kesalahan yang informatif jika pengguna memasukkan token yang tidak valid.

9.2.2 Format Token WhatsApp

WhatsApp memiliki pendekatan yang berbeda dalam hal otentikasi bot, di mana integrasinya dilakukan melalui Twilio, sebuah layanan pihak ketiga yang memungkinkan pengiriman pesan melalui API. Dalam integrasi ini, otentikasi bot WhatsApp tidak hanya bergantung pada satu token seperti di Telegram, tetapi melibatkan beberapa parameter penting yang harus divalidasi secara bersamaan. Untuk setiap bot WhatsApp yang diintegrasikan dengan Twilio, tiga elemen utama yang perlu diperhatikan adalah `account_sid`, `auth_token`, dan `whatsapp_number`. Ketiga elemen ini harus memiliki format yang tepat dan valid agar komunikasi bot melalui Twilio berjalan dengan lancar.

Validasi Token untuk Bot

account_sid

`account_sid` adalah pengenalan unik yang diberikan oleh Twilio kepada setiap akun. Format `account_sid` selalu diawali dengan awalan AC, diikuti oleh kombinasi huruf dan angka yang unik, seperti contoh berikut:

```
ACa7914f90b10b1509ab0780f009d4a9fe
```

Bagian pertama dari SID ini (AC) merupakan kode yang menunjukkan bahwa SID tersebut berkaitan dengan akun Twilio. Sisa string adalah kombinasi karakter yang berfungsi sebagai identitas unik untuk akun tersebut. Dalam implementasi Django, kita harus memastikan bahwa format `account_sid` ini diawali dengan AC dan memiliki panjang tertentu yang sesuai dengan standar Twilio.

auth_token

Selain `account_sid`, Twilio juga memberikan `auth_token` sebagai kunci rahasia yang digunakan untuk otentikasi permintaan API. Token ini terdiri dari kombinasi acak huruf dan angka, dan harus dijaga kerahasiaannya dengan baik karena memberikan akses penuh ke layanan Twilio. Contoh `auth_token` yang valid:

```
07dca14a66ddc70305c5f47588ab84f2
```

Untuk memastikan keamanannya, `auth_token` harus divalidasi agar sesuai dengan panjang minimum tertentu dan berisi karakter yang diharapkan. Jika pengguna memasukkan `auth_token` yang tidak valid, aplikasi harus memberikan pesan kesalahan.

Validasi Token untuk Bot

an yang informatif, namun tetap menjaga privasi informasi sensitif ini.

whatsapp_number

Parameter terakhir yang digunakan adalah `whatsapp_number`, yaitu nomor telepon yang terdaftar di Twilio dan digunakan untuk mengirim serta menerima pesan melalui WhatsApp. Format nomor WhatsApp ini harus mengikuti standar internasional yang diawali dengan tanda plus (+) diikuti oleh kode negara dan nomor telepon. Contoh:

```
+14155238886
```

Pada contoh di atas, +1 adalah kode negara untuk Amerika Serikat, sedangkan 4155238886 adalah nomor telepon yang digunakan. Dalam validasi, nomor WhatsApp harus diperiksa apakah formatnya sesuai dengan standar nomor internasional. Kegagalan dalam memvalidasi nomor telepon ini dapat menyebabkan pesan tidak terkirim atau bot gagal berkomunikasi dengan pengguna WhatsApp.

9.2.3 Mengapa Validasi Ini Penting?

Validasi ketiga parameter ini sangat penting dalam memastikan bahwa bot dapat beroperasi dengan baik melalui Twilio. Kesalahan dalam salah satu parameter, seperti `account_sid` atau `auth_token`, dapat menyebabkan kegagalan otentikasi dengan API Twilio, sehingga bot tidak dapat mengirim atau menerima pesan. Selain itu, jika `whatsapp_number` tidak divalidasi dengan benar, pesan mungkin tidak akan sampai ke pengguna tujuan.

Validasi Token untuk Bot

Pada aplikasi Django kita, kita dapat memanfaatkan validasi di tingkat form atau model untuk memastikan bahwa ketiga parameter ini selalu dalam format yang benar sebelum data disimpan atau digunakan.

Berikut adalah contoh bagaimana validasi untuk ketiga elemen ini dapat diterapkan di form pada aplikasi Django:

```
# File: forms.py

from django import forms
from django.core.exceptions import
ValidationError
import re

class WhatsAppBotForm(forms.Form):
    account_sid = forms.CharField(max_length=34,
help_text="Masukkan Account SID dari Twilio")
    auth_token = forms.CharField(max_length=32,
help_text="Masukkan Auth Token dari Twilio")
    whatsapp_number =
forms.CharField(max_length=15,
help_text="Masukkan nomor WhatsApp terdaftar")

    def clean_account_sid(self):
        account_sid =
self.cleaned_data.get('account_sid')
        if not account_sid.startswith('AC'):
            raise ValidationError("Account SID
harus diawali dengan 'AC'")
        if len(account_sid) != 34:
            raise ValidationError("Account SID
harus terdiri dari 34 karakter")
        return account_sid
```

Validasi Token untuk Bot

```
def clean_auth_token(self):
    auth_token =
self.cleaned_data.get('auth_token')
    if len(auth_token) < 32:
        raise ValidationError("Auth Token
harus terdiri dari 32 karakter")
    return auth_token

def clean_whatsapp_number(self):
    whatsapp_number =
self.cleaned_data.get('whatsapp_number')
    if not re.match(r'^\+?\d{10,15}$',
whatsapp_number):
        raise ValidationError("Nomor
WhatsApp harus dalam format internasional yang
valid")
    return whatsapp_number
```

Dalam kode di atas, kita melakukan beberapa pemeriksaan:

1. **account_sid** harus diawali dengan AC dan memiliki panjang 34 karakter.
2. **auth_token** harus memiliki panjang minimum 32 karakter.
3. **whatsapp_number** harus dalam format nomor internasional yang diawali dengan tanda + dan memiliki panjang antara 10 hingga 15 digit.

Jika salah satu dari parameter ini tidak memenuhi persyaratan, sistem akan menampilkan pesan kesalahan kepada pengguna, memberikan petunjuk yang jelas tentang apa yang perlu diperbaiki.

Validasi Token untuk Bot

Selain di form, validasi ini juga dapat diterapkan di model untuk memastikan integritas data yang disimpan. Contoh implementasi di model bisa seperti ini:

```
# File: models.py

from django.db import models
from django.core.exceptions import
ValidationError
import re

def validate_whatsapp_number(number):
    if not re.match(r'^\+?\d{10,15}$', number):
        raise ValidationError("Nomor WhatsApp
        harus dalam format internasional")

class WhatsAppBot(models.Model):
    account_sid =
models.CharField(max_length=34)
    auth_token = models.CharField(max_length=32)
    whatsapp_number =
models.CharField(max_length=15,
validators=[validate_whatsapp_number])

    def __str__(self):
        return self.whatsapp_number
```

Validasi di level model ini memberikan lapisan perlindungan tambahan, memastikan bahwa setiap data yang disimpan di database sudah dalam format yang benar.

Dengan validasi yang tepat, kita dapat mencegah kesalahan sejak awal dan memastikan bot WhatsApp dapat berkomunikasi dengan benar melalui Twilio. Validasi ini adalah langkah krusial dalam memastikan bahwa sistem otentikasi bekerja dengan lancar,

dan pengguna mendapatkan pengalaman yang lebih baik ketika mengonfigurasi bot mereka.

9.2.4 Format Token untuk Platform Lain

Seiring dengan perkembangan platform bot di luar Telegram dan WhatsApp, sangat mungkin kita akan menghadapi jenis token yang berbeda-beda tergantung pada spesifikasi platform tersebut. Oleh karena itu, merancang sistem validasi token yang fleksibel adalah langkah penting agar aplikasi tetap dapat berkembang dan beradaptasi dengan mudah terhadap berbagai jenis token dari platform baru di masa depan. Ini memungkinkan aplikasi kita untuk tidak hanya mendukung platform yang sudah ada, tetapi juga siap menyambut platform yang belum kita integrasikan.

Selain Telegram, berbagai platform lain juga menyediakan API yang memungkinkan pengembang untuk membuat bot atau aplikasi yang terintegrasi. Setiap platform biasanya memiliki format token yang berbeda, dan memahami format ini sangat penting untuk memastikan bahwa aplikasi kita dapat berfungsi dengan baik.

Pendekatan Fleksibel untuk Validasi Token

Untuk mendukung berbagai platform di masa depan, kita perlu memastikan bahwa sistem validasi tidak bergantung pada satu format token saja. Jika kita membuat sistem validasi yang hanya cocok untuk Telegram atau WhatsApp, ketika platform baru ingin ditambahkan, kita harus menulis ulang atau menyesuaikan validasi secara signifikan. Sebagai gantinya, kita dapat meran-

Validasi Token untuk Bot

cang sistem yang dinamis di mana format token bisa diubah atau ditambahkan dengan mudah tanpa mengubah seluruh kode.

Penggunaan Regex yang Fleksibel

Salah satu cara untuk mencapai fleksibilitas ini adalah dengan menggunakan Regular Expressions (regex) yang dapat disesuaikan berdasarkan kebutuhan. Misalnya, setiap platform dapat memiliki pola token yang spesifik, dan regex yang digunakan untuk memvalidasi token tersebut bisa disimpan secara dinamis atau ditentukan berdasarkan platform yang dipilih oleh pengguna.

Pada Django, pendekatan ini bisa diterapkan di tingkat form atau model. Berikut adalah contoh bagaimana validasi dinamis untuk berbagai token dapat diimplementasikan di form Django:

```
# File: forms.py

from django import forms
from django.core.exceptions import
ValidationError
import re

class BotTokenForm(forms.Form):
    platform =
forms.ChoiceField(choices=[('telegram',
'Telegram'), ('whatsapp', 'WhatsApp'),
('platform_lain', 'Platform Lain')])
    token = forms.CharField(max_length=255,
help_text="Masukkan token yang sesuai dengan
platform yang dipilih")

    def clean_token(self):
        token = self.cleaned_data.get('token')
```

Validasi Token untuk Bot

```
platform =
self.cleaned_data.get('platform')

# Validasi berdasarkan platform
if platform == 'telegram':
    if not re.match(r'^\d{10}: [A-Za-z0-9_-]{35}$', token):
        raise ValidationError("Token
Telegram tidak valid")
    elif platform == 'whatsapp':
        if not re.match(r'^[A-Za-z0-9]{32}$', token): # Auth Token Twilio misalnya
            raise ValidationError("Token
WhatsApp tidak valid")
    else:
        # Validasi default atau token
platform lain
        if not re.match(r'^[A-Za-z0-9_-]{20,255}$', token): # Contoh umum token
platform lain
            raise ValidationError("Token
untuk platform ini tidak valid")

return token
```

Pada kode di atas, kita melihat bagaimana form Django diatur untuk menerima input token berdasarkan platform yang dipilih. Ketika pengguna memilih platform, sistem akan menyesuaikan validasi token sesuai dengan pola yang berlaku untuk platform tersebut. Misalnya, token untuk Telegram divalidasi agar sesuai dengan pola `^\d{10}: [A-Za-z0-9_-]{35}$`, sementara token WhatsApp mungkin memiliki panjang 32 karakter dengan karakter alfanumerik.

Validasi Token untuk Bot

Untuk platform lain yang belum diintegrasikan, kita bisa menggunakan regex yang lebih umum, seperti `^[A-Za-z0-9_-]{20,255}$`, yang menerima token dengan panjang antara 20 hingga 255 karakter. Ini adalah pola yang cukup fleksibel untuk menangani berbagai jenis token yang mungkin digunakan oleh platform bot lain di masa depan.

Mengakomodasi Platform Baru dengan Mudah

Dengan pendekatan di atas, ketika platform baru diperkenalkan, kita hanya perlu menambahkan logika validasi tambahan di bagian yang relevan, tanpa perlu mengubah seluruh struktur kode. Sebagai contoh, jika suatu hari kita menambahkan dukungan untuk platform seperti Slack atau Discord, kita bisa menambahkan pola validasi baru sesuai dengan format token yang mereka gunakan, tanpa mempengaruhi validasi token yang ada untuk Telegram atau WhatsApp.

Selain itu, sistem juga bisa dibuat lebih fleksibel lagi dengan menggunakan database atau konfigurasi eksternal untuk menyimpan pola validasi token per platform. Ini memungkinkan administrator sistem untuk menambah atau mengubah format token tanpa perlu mengubah kode di aplikasi utama.

Sebagai contoh, berikut adalah bagaimana kita bisa menggunakan model di Django untuk menyimpan pola token dinamis:

```
# Lokasi: models.py
from django.db import models
```

Validasi Token untuk Bot

```
from django.core.exceptions import
ValidationError
import re

class Platform(models.Model):
    name = models.CharField(max_length=50)
    token_pattern =
models.CharField(max_length=255) # Simpan pola
regex token

    def __str__(self):
        return self.name

    def validate_token(self, token):
        if not re.match(self.token_pattern,
token):
            raise ValidationError(f"Token untuk
platform {self.name} tidak valid")
```

Pada model di atas, setiap platform bisa memiliki pola regex token tersendiri yang disimpan di database. Ketika platform baru ditambahkan, administrator hanya perlu menambahkan entri baru di model `Platform` beserta pola tokennya. Kemudian, ketika form memvalidasi token, ia bisa mengambil pola dari database dan menggunakannya untuk validasi.

```
# File: forms.py

class BotTokenForm(forms.Form):
    platform =
forms.ModelChoiceField(queryset=Platform.objects
.all())
    token = forms.CharField(max_length=255,
help_text="Masukkan token yang sesuai dengan
platform yang dipilih")

    def clean_token(self):
```

Validasi Token untuk Bot

```
token = self.cleaned_data.get('token')
platform =
self.cleaned_data.get('platform')

# Validasi token berdasarkan platform
yang dipilih
platform.validate_token(token)

return token
```

Dengan pendekatan ini, kita tidak hanya membuat sistem yang fleksibel tetapi juga mudah diadaptasi. Setiap kali platform baru muncul, kita cukup memperbarui data di database, tanpa menyentuh kode utama aplikasi.

Setiap platform biasanya memiliki mekanisme otentikasi tersendiri, baik itu berupa token sederhana seperti yang digunakan oleh Telegram maupun kombinasi lebih kompleks seperti pada WhatsApp melalui Twilio. Platform-platform bot lain seperti Facebook Messenger, Slack, atau Discord juga memiliki token API mereka masing-masing. Oleh karena itu, fleksibilitas dalam memvalidasi token menjadi krusial dalam menjaga kompatibilitas sistem.

Sebagai contoh, mari kita lihat format token yang umum digunakan oleh platform seperti Discord dan Slack. Token untuk Discord biasanya memiliki struktur yang berbeda dari token Telegram. Format token Discord terdiri dari serangkaian karakter alfanumerik yang panjang, yang dihasilkan oleh sistem Discord dan dapat terlihat seperti ini:

```
ODI0MTg2ODI3NzQ2NDAwMjM5.YDtrEw.3f5rU8Eqz4M
```

Validasi Token untuk Bot

b8zQW_p-QZPQR7Bk. Token ini memberikan akses penuh ke bot yang telah dibuat di dalam server Discord.

Untuk memvalidasi token Discord, kita dapat menggunakan pendekatan serupa dengan yang kita gunakan untuk token Telegram, tetapi dengan pola yang sesuai dengan format Discord. Kita bisa menggunakan ekspresi reguler untuk memeriksa apakah token yang dimasukkan oleh pengguna valid.

Berikut adalah contoh fungsi validasi untuk token Discord:

```
import re

def validasi_token_discord(token):
    # Memeriksa apakah token sesuai dengan
    format Discord
    pola_token = r'^[A-Za-z0-9]{24}\.[A-Za-z0-9]{6}\.[A-Za-z0-9_]{27}$' # Format token Discord
    return re.match(pola_token, token) is not
    None # Mengembalikan True jika cocok
```

Dalam kode di atas, kita mendefinisikan pola untuk token Discord dan menggunakan fungsi `re.match` untuk memeriksa apakah token yang diberikan oleh pengguna cocok dengan format yang diharapkan. Jika token tersebut valid, fungsi ini akan mengembalikan nilai `True`, menunjukkan bahwa token tersebut dapat digunakan untuk mengakses API Discord.

Sama halnya dengan Slack, token yang digunakan biasanya dimulai dengan awalan `xoxb-`, yang diikuti oleh serangkaian angka dan huruf. Misalnya, token Slack bisa terlihat seperti ini:

Validasi Token untuk Bot

xoxb-123456789012-1234567890123-

ABCdefGhIJKlmnoPQRstuVwXyz. Validasi token Slack juga dapat dilakukan menggunakan ekspresi reguler yang sesuai.

Dengan memahami dan menerapkan validasi yang tepat untuk token dari berbagai platform, kita dapat memastikan bahwa bot atau aplikasi yang kita buat memiliki akses yang sah dan tidak rentan terhadap kesalahan penggunaan. Pada sub bab selanjutnya, kita akan membahas implementasi praktis dari validasi token ini dalam aplikasi Django kita, serta bagaimana menampilkan pesan kesalahan yang jelas dan informatif kepada pengguna ketika mereka memasukkan token yang tidak valid.

9.2.5 Mengapa Format Token Itu Penting?

Memahami format token sangat penting dalam pengembangan aplikasi, terutama saat bekerja dengan berbagai platform yang menyediakan API untuk integrasi. Format token bukan hanya sekedar serangkaian karakter acak; ia memiliki struktur yang spesifik yang menentukan bagaimana token tersebut harus digunakan dan divalidasi.

Pertama, format token membantu dalam mencegah kesalahan. Ketika pengguna diminta untuk memasukkan token, sangat mungkin mereka melakukan kesalahan ketik atau memasukkan karakter yang tidak sesuai. Dengan memvalidasi format token, kita dapat memberikan umpan balik langsung kepada pengguna jika token yang mereka masukkan tidak sesuai. Ini tidak hanya meningkatkan pengalaman pengguna, tetapi juga mencegah potensi kesalahan dalam komunikasi dengan API.

Validasi Token untuk Bot

Kedua, memahami format token membantu dalam menjaga keamanan aplikasi. Token yang tidak valid dapat menjadi pintu masuk bagi potensi penyalahgunaan. Jika kita tidak memvalidasi token dengan benar, ada risiko bahwa pengguna yang tidak berwenang dapat mencoba mengakses API dan melakukan tindakan yang tidak sah. Dengan menerapkan validasi yang ketat berdasarkan format yang ditentukan, kita dapat menambah lapisan keamanan yang penting bagi aplikasi kita.

Ketiga, dengan mengetahui format token, pengembang dapat lebih mudah melakukan debugging. Ketika terjadi masalah dalam komunikasi antara aplikasi dan API, memahami format token yang benar dapat membantu dalam menentukan apakah masalah tersebut berasal dari token yang tidak valid. Misalnya, jika API mengembalikan kesalahan autentikasi, pengembang dapat dengan cepat memeriksa token yang digunakan untuk memastikan bahwa formatnya benar dan tidak ada kesalahan dalam input pengguna.

Sebagai contoh, dalam proyek Django kita, jika kita membiarkan pengguna memasukkan token tanpa validasi yang ketat, kita bisa menghadapi masalah yang lebih besar saat berusaha untuk menghubungkan aplikasi dengan API. Oleh karena itu, implementasi validasi format token yang tepat tidak hanya merupakan praktik yang baik, tetapi juga merupakan langkah penting dalam pengembangan perangkat lunak yang aman dan efektif.

Dengan pemahaman ini, kita dapat melanjutkan ke langkah berikutnya dalam proses validasi token, yang akan melibatkan implementasi praktis di dalam aplikasi Django kita. Pada sub bab berikutnya, kita akan menjelajahi cara menerapkan validasi token ini secara efektif untuk memastikan bahwa semua token yang dimasukkan oleh pengguna memenuhi kriteria yang telah ditentukan.

9.3 Implementasi Validasi Token Di Proyek

Dalam pengembangan aplikasi Django yang berfungsi untuk membuat dan mengelola bot, salah satu langkah penting adalah memastikan bahwa token yang dimasukkan oleh pengguna valid dan unik. Di bagian ini, kita akan membahas cara mendefinisikan model untuk token dalam konteks bot yang telah kita buat sebelumnya.

9.3.1 Menerapkan Validasi di Model

Dalam aplikasi bot, penting untuk memastikan bahwa setiap data yang dimasukkan ke dalam model sudah divalidasi dengan benar sebelum disimpan ke dalam database. Dengan menambahkan validasi, kita dapat mencegah kesalahan data seperti token yang tidak valid atau nama bot yang terlalu pendek. Validasi ini juga dapat memperbaiki pengalaman pengguna dengan memberikan pesan yang lebih informatif jika terjadi kesalahan.

Di dalam model Bot yang telah didefinisikan sebelumnya, kita telah mencakup atribut untuk token. Atribut token dideklarasikan

Validasi Token untuk Bot

kan sebagai `CharField` dengan batas maksimal 255 karakter, dan kita juga menambahkan opsi `unique=True` untuk memastikan bahwa setiap token yang dimasukkan adalah unik. Ini penting karena kita tidak ingin dua bot memiliki token yang sama, yang bisa menyebabkan konflik saat berkomunikasi dengan API yang relevan.

Berikut adalah potongan kode model yang relevan:

```
# File: apps/bots/models.py
class Bot(models.Model):
    # ... kode lainnya ...
    token = models.CharField(max_length=255,
                             unique=True, help_text="Masukkan token bot yang unik.", blank=False, null=False)
    # ... kode lainnya ...
```

Dengan pengaturan ini, setiap kali pengguna mencoba untuk menyimpan bot baru, Django akan otomatis memeriksa apakah token tersebut sudah ada di database. Jika ada, Django akan memberikan kesalahan dan mencegah penyimpanan data yang tidak valid.

9.3.2 Mengaitkan Token dengan Bot

Mengaitkan token dengan bot juga berarti kita harus mempertimbangkan bagaimana token tersebut akan digunakan saat berinteraksi dengan API. Misalnya, jika kita menggunakan token Telegram atau WhatsApp, kita perlu memastikan bahwa token yang dimasukkan sesuai dengan format yang benar untuk masing-masing platform.

Validasi Token untuk Bot

Untuk menambahkan validasi token di model Bot, kita perlu menyesuaikan model tersebut agar bisa memvalidasi token Telegram dan WhatsApp sebelum disimpan ke dalam database. Validasi ini bisa dilakukan dengan menggunakan metode `clean()` di dalam model Django. Metode ini secara otomatis akan dipanggil ketika model disimpan, dan kita bisa menambahkan logika validasi di dalamnya.

Berikut adalah kode yang dimodifikasi untuk menambahkan validasi di model Bot:

```
# File: apps/bots/models.py

from django.db import models
from django.contrib.auth.models import User
from django.core.exceptions import
ValidationError # Import untuk validasi
import re # Import untuk memvalidasi pola token

class Bot(models.Model):
    BOT_CHOICES = [
        ('telegram', 'Telegram'),
        ('whatsapp', 'WhatsApp'),
    ]
    user = models.ForeignKey(User,
on_delete=models.CASCADE) # Bot akan terkait
dengan pengguna
    bot_type = models.CharField(max_length=20,
choices=BOT_CHOICES) # Jenis bot: Telegram atau
WhatsApp

    token_telegram =
models.CharField(max_length=255, unique=True,
help_text="Masukkan token bot yang unik.") #
Token Telegram
```

Validasi Token untuk Bot

```
webhook_url =
models.URLField(max_length=255, blank=True,
help_text="URL webhook bot akan diatur secara
otomatis.") # URL webhook

is_active =
models.BooleanField(default=True,
help_text="Status bot aktif atau tidak.") #
Status aktif atau tidaknya bot

# Pesan otomatis
start_message = models.TextField(blank=True,
default="Selamat datang di bot!",
help_text="Pesan yang dikirim saat pengguna
mengetik /start.")
help_message = models.TextField(blank=True,
default="Berikut cara menggunakan bot.",
help_text="Pesan yang dikirim saat pengguna
mengetik /help.")

messages = models.JSONField(default=list,
blank=True, help_text="Daftar pesan yang
diterima dan responsnya dalam format JSON.") #
Format JSON

# Timestamp
created_at =
models.DateTimeField(auto_now_add=True) #
Tanggal bot dibuat
updated_at =
models.DateTimeField(auto_now=True) # Tanggal
terakhir bot diperbarui

name = models.CharField(max_length=255,
blank=True, default="Bot Saya", help_text="Nama
bot.") # Nama bot
description = models.TextField(blank=True,
default="Ini bot pertama saya",
help_text="Deskripsi bot.") # Deskripsi bot
```

Validasi Token untuk Bot

```
def __str__(self):
    return f'{self.bot_type.capitalize()}
Bot - {self.user.username}'

# Tambahkan metode clean untuk validasi
def clean(self):
    # Validasi nama bot tidak boleh kosong
    if not self.name.strip():
        raise ValidationError("Nama bot
tidak boleh kosong.")

    # Validasi deskripsi bot tidak lebih
dari 500 karakter
    if len(self.description) > 500:
        raise ValidationError("Deskripsi bot
tidak boleh lebih dari 500 karakter.")

    # Validasi token Telegram hanya jika
bot_type adalah Telegram
    if self.bot_type == 'telegram':
        # Pola token Telegram yang benar:
angka-angka diikuti dengan titik dua, lalu
diikuti oleh string karakter acak
        if not re.match(r'^\d+:[\w-]+$' ,
self.token_telegram):
            raise ValidationError("Token
Telegram tidak valid. Pastikan token dalam
format yang benar seperti '123456:ABC-
DEF1234ghIkl-zyx57W2v1u123ew11'.")

class Meta:
    verbose_name = "Bot"
    verbose_name_plural = "Bots"
```

Penjelasan Kode Validasi

1. *Validasi Nama Bot*

Nama bot harus diisi dan tidak boleh kosong. Dengan

Validasi Token untuk Bot

menggunakan fungsi `strip()`, kita memastikan bahwa bahkan jika nama hanya berisi spasi kosong, tetap akan dianggap sebagai nama yang tidak valid. Jika nama kosong, akan muncul pesan kesalahan: "Nama bot tidak boleh kosong."

2. *Validasi Deskripsi Bot*

Deskripsi bot dibatasi hingga 500 karakter. Batasan ini penting untuk memastikan bahwa deskripsi tetap singkat dan padat. Jika deskripsi melebihi batas, pesan kesalahan akan muncul: "Deskripsi bot tidak boleh lebih dari 500 karakter."

3. *Validasi Token Telegram*

Token Telegram harus mengikuti pola tertentu: token dimulai dengan sejumlah angka, diikuti oleh titik dua, kemudian diikuti oleh serangkaian karakter acak. Contoh format token yang benar adalah `123456:ABC-DEF1234ghIkl-zyx57W2v1u123ew11`. Jika token tidak sesuai dengan pola ini, akan muncul pesan kesalahan: "Token Telegram tidak valid. Pastikan token dalam format yang benar."

Menambahkan Validasi di Formulir

Setelah kita menambahkan validasi pada model, formulir yang menggunakan model ini akan secara otomatis mewarisi validasi yang sudah ditentukan. Hal ini berarti, jika pengguna mencoba memasukkan nama bot kosong, deskripsi terlalu panjang, atau token Telegram yang tidak valid, formulir akan memberikan umpan balik kesalahan yang sesuai.

Validasi Token untuk Bot

Dengan validasi yang ditambahkan di model, kita memastikan bahwa setiap data yang masuk ke dalam database sudah memenuhi syarat yang diperlukan dan menghindari masalah yang mungkin muncul di kemudian hari.

Menambahkan validasi di level model memberikan keuntungan besar dalam menjaga integritas data. Pada model `Bot`, validasi yang telah kita tambahkan memastikan bahwa nama bot diisi, deskripsi bot tidak terlalu panjang, dan token Telegram valid sesuai dengan pola yang benar. Dengan begitu, data yang masuk ke dalam database lebih terjamin validitasnya, dan pengguna juga mendapatkan pengalaman yang lebih baik dengan adanya pesan kesalahan yang jelas dan informatif.

Pada sub-bab berikutnya, kita akan membahas bagaimana menghubungkan model dan template dengan formulir di dalam view serta bagaimana menampilkan pesan kesalahan ini di antarmuka pengguna secara efektif.

Jika ada satu saja atribut yang tidak valid atau kosong, maka Django akan mengembalikan error, sehingga data yang tidak valid tidak akan tersimpan di database.

Dengan pengaturan model ini, kita telah menciptakan fondasi yang kuat untuk validasi token dalam aplikasi bot kita. Langkah selanjutnya adalah menerapkan logika ini dalam tampilan (view) dan formulir (form) sehingga pengguna dapat memasukkan token yang valid saat membuat bot baru. Pada sub bab berikutnya, kita

akan menjelajahi cara membangun tampilan dan formulir yang sesuai untuk mengelola input token ini.

9.4 Menerapkan Validasi Di Form

Untuk memastikan bahwa validasi token dilakukan sebelum data dari form disimpan ke database, kita dapat menambahkan validasi di level form pada file `forms.py`. Langkah ini sangat penting untuk memastikan data yang diterima oleh sistem adalah valid, khususnya dalam hal format token Telegram dan informasi terkait WhatsApp. Form akan memberikan umpan balik yang tepat kepada pengguna apabila terdapat kesalahan dalam pengisian data.

Langkah pertama adalah menambahkan validasi pada metode `clean_<field_name>()` di dalam form. Kita akan menerapkan validasi untuk token Telegram menggunakan pola tertentu dan juga melakukan validasi untuk data lainnya seperti nama bot dan deskripsinya.

9.4.1 Validasi di Level Form

Django memberikan fleksibilitas dalam melakukan validasi data menggunakan `ModelForm`. Dalam kasus ini, kita akan menggunakan metode `clean_token_telegram()` untuk melakukan validasi spesifik pada token Telegram. Selain itu, kita akan menggunakan atribut `widgets` untuk mempercantik tampilan form, serta `help_texts` untuk memberikan petunjuk yang jelas kepada pengguna.

Validasi Token untuk Bot

Berikut adalah implementasi detail untuk memvalidasi format token Telegram sebelum data disimpan.

```
# File: apps/bots/forms.py

from django import forms
from django.core.exceptions import
ValidationError
import re # Untuk memvalidasi pola token
from .models import Bot # Mengimpor model Bot

class BotForm(forms.ModelForm):
    class Meta:
        model = Bot # Menghubungkan form dengan
model Bot
        fields = ['bot_type', 'token_telegram',
'name', 'description', 'is_active'] # Field
yang ditampilkan dalam form
        widgets = {
            'name':
forms.TextInput(attrs={'class': 'form-control',
'placeholder': 'Masukkan nama bot'}),
            'description':
forms.Textarea(attrs={'class': 'form-control',
'rows': 3, 'placeholder': 'Deskripsi bot'}),
        }
        help_texts = {
            'token_telegram': 'Masukkan token
unik yang diterima dari platform Telegram.',
            'name': 'Nama bot dapat diubah
sesuai keinginan.',
            'description': 'Deskripsi singkat
tentang bot dan fungsinya.',
        }

    # Validasi khusus untuk token Telegram
    def clean_token_telegram(self):
```

Validasi Token untuk Bot

```
        token_telegram =
self.cleaned_data.get('token_telegram')

        # Pola token Telegram: angka-angka
        diikuti dengan titik dua, lalu serangkaian
        karakter acak
        if token_telegram and not re.match(r'^\d+:[\w-]+$'
, token_telegram):
            raise ValidationError("Token
Telegram tidak valid. Pastikan format token
adalah seperti '123456:ABC-DEF1234ghIkl-
zyx57W2v1u123ew11'.")

        return token_telegram

# Validasi nama bot
def clean_name(self):
    name = self.cleaned_data.get('name')
    if not name.strip():
        raise ValidationError("Nama bot
tidak boleh kosong.")
    return name

# Validasi deskripsi bot
def clean_description(self):
    description =
self.cleaned_data.get('description')
    if len(description) > 500:
        raise ValidationError("Deskripsi bot
tidak boleh lebih dari 500 karakter.")
    return description
```

Penjelasan Validasi di Level Form:

1. **Validasi Token Telegram:** Pada metode `clean_token_telegram()`, kita memvalidasi token Telegram dengan menggunakan regex (pola reguler). Token Telegram harus mengikuti format `bot_id:secret_key`,

Validasi Token untuk Bot

di mana `bot_id` adalah angka-angka, diikuti oleh tanda : dan `secret_key` yang merupakan serangkaian karakter acak. Jika format token tidak sesuai, sistem akan menampilkan pesan error: *"Token Telegram tidak valid. Pastikan format token adalah seperti '123456'."*

2. **Validasi Nama Bot:** Nama bot tidak boleh kosong atau hanya terdiri dari spasi. Metode `clean_name()` memastikan bahwa nama bot berisi karakter yang valid. Jika tidak valid, pesan kesalahan: *"Nama bot tidak boleh kosong."* akan muncul.
3. **Validasi Deskripsi Bot:** Deskripsi bot dibatasi hingga 500 karakter. Dengan menggunakan metode `clean_description()`, kita memastikan bahwa deskripsi tidak melebihi batas ini. Jika deskripsi terlalu panjang, sistem akan memberikan pesan kesalahan: *"Deskripsi bot tidak boleh lebih dari 500 karakter."*

Dengan menambahkan validasi di level form, kita memastikan bahwa data yang dimasukkan pengguna sesuai dengan format yang diharapkan. Validasi token Telegram, nama bot, dan deskripsi bot diterapkan secara efektif melalui metode `clean_<field_name>()`. Selain itu, menampilkan pesan error di template membuat pengguna lebih mudah memahami kesalahan yang terjadi, sehingga pengalaman mereka lebih baik.

9.5 Menggunakan Validators Kustom

Selain menggunakan metode `clean_<field_name>()` di form, kita juga bisa membuat validator kustom di Django untuk memeriksa format token atau data lain yang lebih kompleks. Dengan validator kustom, validasi ini bisa digunakan tidak hanya di form, tetapi juga di level model atau field lain yang memerlukan validasi yang sama. Pendekatan ini membuat validasi lebih modular dan dapat digunakan kembali.

Membuat Validator Kustom

Validator kustom adalah fungsi atau kelas yang digunakan untuk memvalidasi data tertentu sebelum disimpan ke database. Pada contoh ini, kita akan membuat dua validator kustom:

- **Validator untuk Token Telegram:** Memastikan token mengikuti format `bot_id:secret_key`.
- **Validator untuk SID Akun Twilio:** Memvalidasi SID akun Twilio yang harus berisi karakter alfanumerik dengan panjang tertentu.

Implementasi Validator Kustom

Berikut adalah cara membuat validator kustom untuk token Telegram.

```
# File: apps/bots/validators.py
import re
```

Validasi Token untuk Bot

```
from django.core.exceptions import
ValidationError

# Validator untuk token Telegram
def validate_telegram_token(value):
    """Validasi format token Telegram yang harus
    berupa bot_id:secret_key."""
    if not re.match(r'^\d+:[\w-]+$', value):
        raise ValidationError(
            "Token Telegram tidak valid. Format
            yang benar adalah '123456:ABC-DEF1234ghIkl-
            zyx57W2v1u123ew11'."
        )
```

Menggunakan Validator Kustom di Model

Setelah membuat validator kustom, kita bisa langsung menggunakannya di model. Dengan cara ini, setiap kali model Bot digunakan, validasi akan berjalan otomatis saat objek dibuat atau diperbarui.

```
# File: apps/bots/models.py

from django.db import models
from django.contrib.auth.models import User
from .validators import validate_telegram_token

class Bot(models.Model):
    BOT_CHOICES = [
        ('telegram', 'Telegram'),
        ('whatsapp', 'WhatsApp'),
    ]
    user = models.ForeignKey(User,
on_delete=models.CASCADE) # Bot terkait dengan
pengguna
```

Validasi Token untuk Bot

```
bot_type = models.CharField(max_length=20,
choices=BOT_CHOICES) # Jenis bot: Telegram atau
WhatsApp

token_telegram = models.CharField(
    max_length=255,
    unique=True,
    validators=[validate_telegram_token], #
Menambahkan validator untuk token Telegram
    help_text="Masukkan token bot yang
unik."
)

webhook_url =
models.URLField(max_length=255, blank=True,
help_text="URL webhook bot akan diatur secara
otomatis.")
is_active =
models.BooleanField(default=True,
help_text="Status bot aktif atau tidak.")

# Pesan otomatis
start_message = models.TextField(blank=True,
default="Selamat datang di bot!",
help_text="Pesan yang dikirim saat pengguna
mengetik /start.")
help_message = models.TextField(blank=True,
default="Berikut cara menggunakan bot.",
help_text="Pesan yang dikirim saat pengguna
mengetik /help.")

messages = models.JSONField(default=list,
blank=True, help_text="Daftar pesan yang
diterima dan responsnya dalam format JSON.")

# Timestamp
```


Validasi Token untuk Bot

```
    created_at =
models.DateTimeField(auto_now_add=True)
    updated_at =
models.DateTimeField(auto_now=True)

    name = models.CharField(max_length=255,
blank=True, default="Bot Saya", help_text="Nama
bot.")
    description = models.TextField(blank=True,
default="Ini bot pertama saya",
help_text="Deskripsi bot.")

    def __str__(self):
        return f'{self.bot_type.capitalize()}
Bot - {self.user.username}'

    class Meta:
        verbose_name = "Bot"
        verbose_name_plural = "Bots"
```

Menggunakan Validator Kustom di Form

Jika kita ingin melakukan validasi di form alih-alih model, validator kustom ini juga dapat ditambahkan ke form field. Berikut contoh penggunaannya di form:

```
# File: apps/bots/forms.py

from django import forms
from .models import Bot
from .validators import validate_telegram_token

class BotForm(forms.ModelForm):
    class Meta:
        model = Bot # Menghubungkan form dengan
model Bot
```

Validasi Token untuk Bot

```
        fields = ['bot_type', 'token_telegram',
                  'name', 'description', 'is_active']
        widgets = {
            'name':
forms.TextInput(attrs={'class': 'form-control',
                        'placeholder': 'Masukkan nama bot'}),
            'description':
forms.Textarea(attrs={'class': 'form-control',
                      'rows': 3, 'placeholder': 'Deskripsi bot'}),
        }
        help_texts = {
            'token_telegram': 'Masukkan token
unik yang diterima dari platform Telegram.',

            'name': 'Nama bot dapat diubah
sesuai keinginan.',
            'description': 'Deskripsi singkat
tentang bot dan fungsinya.',
        }

# Menambahkan validasi di form untuk token
Telegram menggunakan validator kustom
token_telegram =
forms.CharField(validators=[validate_telegram_to
ken])
```

Penjelasan Validasi

1. *Validator Token Telegram:*

- Validator ini menggunakan regex untuk memastikan format token Telegram sesuai dengan pola `bot_id:secret_key`.
- Apabila format token tidak sesuai, akan muncul pesan kesalahan yang spesifik seperti: *"Token Telegram tidak valid. Format yang benar adalah '123456"*

."

Dengan menggunakan validator kustom, kita dapat memastikan bahwa data yang dimasukkan oleh pengguna valid sebelum disimpan ke dalam database. Validator kustom memberikan fleksibilitas tambahan karena bisa digunakan kembali di berbagai tempat, baik di model maupun di form. Pendekatan ini membuat logika validasi lebih modular dan memudahkan kita untuk menjaga konsistensi data di seluruh aplikasi.

9.6 Menangani Kesalahan Input Token

Dalam sistem yang menerima input dari pengguna, seperti token untuk menghubungkan bot Telegram atau WhatsApp, sangat penting untuk menangani kesalahan input dengan baik dan memberikan pesan yang jelas. Hal ini akan memudahkan pengguna dalam memperbaiki kesalahan mereka dan menghindari kebingungan selama proses pembuatan bot. Pada sub-bab ini, kita akan membahas cara menampilkan pesan kesalahan yang lebih informatif dan bagaimana memanfaatkan komponen Bootstrap untuk membuat tampilan pesan kesalahan yang lebih user-friendly.

9.6.1 Menampilkan Pesan Kesalahan yang Jelas

Menampilkan pesan kesalahan yang jelas adalah langkah pertama dalam menangani kesalahan input. Pesan kesalahan yang ambigu atau tidak spesifik dapat membingungkan pengguna. Sebagai contoh, jika token yang dimasukkan tidak sesuai dengan format yang diharapkan, penting untuk menampilkan pesan yang memberitahukan kesalahan secara spesifik, seperti "Token tidak valid. Pastikan Anda menggunakan format yang benar."

Pesan kesalahan yang jelas dan informatif sangat penting dalam proses validasi. Ketika pengguna memasukkan token yang tidak sesuai atau tidak valid, kita perlu memastikan bahwa mereka memahami alasan di balik kesalahan tersebut. Dengan menggunakan Django forms, kita sudah menyiapkan mekanisme untuk memberikan pesan kesalahan di dalam metode `clean_token()`. Namun, kita juga perlu menampilkan pesan ini di antarmuka pengguna agar mereka dapat melihat dan memperbaiki kesalahan dengan mudah.

Untuk ini, kita akan memanfaatkan mekanisme validasi bawaan Django yang telah dibahas sebelumnya, di mana kesalahan validasi pada form otomatis ditangkap dan disimpan dalam atribut `form.errors`. Kesalahan ini kemudian ditampilkan di template menggunakan loop untuk setiap field yang mengalami error.

Validasi Token untuk Bot

Ketika formulir dikirim dan validasi gagal, Django secara otomatis akan menambahkan pesan kesalahan ke dalam objek form. Di dalam template, kita dapat menampilkan pesan-pesan ini dengan menggunakan loop untuk iterasi setiap field.

Pada Django forms, pesan kesalahan ini dihasilkan oleh mekanisme validasi yang sudah ada. Ketika pengguna mengisi form dan validasi gagal, kesalahan tersebut akan disimpan dalam atribut `form.errors`. Untuk menampilkan pesan kesalahan ini di antarmuka pengguna, kita dapat memanfaatkan elemen-elemen Bootstrap, seperti `invalid-feedback`.

Berikut adalah contoh tampilan kesalahan di template menggunakan elemen Bootstrap untuk token:

```
<!-- File: apps/templates/bots/create_bot.html -->
-->
<div class="form-group" id="token-field">
  <label for="{ form.token.id_for_label }">Token</label>
  <input type="text" name="token"
id="{ form.token.id_for_label }"
value="{ form.token.value }" class="form-
control {% if form.token.errors %}is-invalid{%
endif %}">
    {% for error in form.token.errors %}
      <div class="invalid-feedback">{{ error
}}</div>
    {% endfor %}
</div>
```

Pada kode di atas, ketika token tidak valid, field tersebut akan mendapatkan class `is-invalid`, yang otomatis memberikan warna merah pada input, sesuai dengan gaya Bootstrap. Pesan

Validasi Token untuk Bot

kesalahan akan muncul di bawah input dengan class `invalid-feedback`, sehingga tampilannya terlihat profesional dan rapi.

9.6.2 Menggunakan Bootstrap untuk Notifikasi Kesalahan

Agar tampilan pesan kesalahan lebih user-friendly dan profesional, kita bisa memanfaatkan komponen Bootstrap seperti `invalid-feedback` untuk kesalahan pada setiap field, dan modal Bootstrap untuk menampilkan notifikasi kesalahan secara lebih menyeluruh jika ada kesalahan yang lebih besar.

Bootstrap menawarkan berbagai komponen yang dapat kita gunakan untuk meningkatkan tampilan pesan kesalahan. Dengan menggunakan kelas-kelas Bootstrap, kita dapat menampilkan pesan kesalahan dalam bentuk notifikasi yang menarik dan mudah dibaca. Misalnya, kita dapat membungkus pesan kesalahan dalam elemen `div` dengan kelas `alert` yang sesuai.

Berikut adalah contoh penggunaan modal Bootstrap untuk menampilkan pesan kesalahan ketika form tidak valid:

```
{% if form.errors %}
<div id="errorModal" class="modal fade"
tabindex="-1" aria-labelledby="errorModalLabel"
aria-hidden="true">
  <div class="modal-dialog">
    <div class="modal-content">
      <div class="modal-header">
        <h5 class="modal-title"
id="errorModalLabel">Terjadi Kesalahan</h5>
```

Validasi Token untuk Bot

```
        <button type="button" class="btn-close"
data-bs-dismiss="modal" aria-
label="Close"></button>
    </div>
    <div class="modal-body">
        {% for field in form %}
            {% for error in field.errors %}
                <p>{{ error }}</p>
            {% endfor %}
        {% endfor %}
        {% for error in form.non_field_errors %}
            <p>{{ error }}</p>
        {% endfor %}
    </div>
    <div class="modal-footer">
        <button type="button" class="btn btn-
secondary" data-bs-
dismiss="modal">Tutup</button>
    </div>
</div>
</div>
{% endif %}
```

Modal ini akan muncul jika form tidak valid, menampilkan semua pesan kesalahan dalam satu tampilan. Agar modal otomatis muncul setelah validasi form gagal, kita bisa menambahkan sedikit JavaScript:

```
document.addEventListener('DOMContentLoaded',
function () {
    if (document.querySelector('#errorModal')) {
        var errorModal = new
bootstrap.Modal(document.getElementById('errorMo
dal'));
        errorModal.show();
    }
})
```

```
});
```

Dengan pendekatan ini, setiap kesalahan yang tidak terkait dengan satu field saja, seperti kesalahan sistem atau kesalahan yang lebih umum, bisa ditampilkan dengan lebih jelas kepada pengguna. Modal juga memberikan tampilan yang lebih profesional dan user-friendly dibandingkan hanya menampilkan pesan di halaman biasa.

Dengan menampilkan pesan kesalahan yang jelas dan memanfaatkan komponen Bootstrap seperti `invalid-feedback` dan `modal`, kita dapat memberikan pengalaman yang lebih baik kepada pengguna. Penanganan kesalahan input yang efektif membantu mengurangi kebingungan dan memastikan bahwa proses pembuatan bot berjalan lancar.

Pemanfaatan komponen Bootstrap seperti ini juga membantu menjaga tampilan aplikasi tetap konsisten dan ramah pengguna. Kesalahan input yang ditangani dengan baik dan disajikan dengan desain yang menarik akan membantu mengurangi frustrasi pengguna, terutama saat mereka harus melakukan input yang sensitif seperti token.

Dengan demikian, sub-bab ini telah menjelaskan pentingnya memberikan pesan kesalahan yang jelas dan bagaimana kita dapat memanfaatkan Bootstrap untuk membuat notifikasi kesalahan yang lebih user-friendly.

9.7 Kode Lengkap Setelah Di Validasi

Kita bisa memilih untuk melakukan validasi di salah satu tempat saja, tergantung kebutuhan dan alur aplikasi. Namun, ada beberapa alasan mengapa kita mungkin ingin menggunakan salah satu, atau bahkan keduanya:

Validasi di Models:

- **Keuntungan:** Validasi di model memastikan bahwa setiap kali objek disimpan ke database, validasi dijalankan. Ini berguna jika model digunakan di berbagai tempat selain form, seperti API atau skrip khusus.
- **Kelemahan:** Validasi di model tidak memberikan pesan error yang rapi pada form HTML, karena validasi ini lebih bersifat internal.

Validasi di Forms:

- **Keuntungan:** Validasi di form memungkinkan kita memberikan umpan balik langsung kepada pengguna melalui form dengan pesan error yang ramah dan mudah dipahami, yang ditampilkan langsung di halaman.
- **Kelemahan:** Validasi di form hanya berlaku ketika data dikirim melalui form. Jika data dimasukkan ke model dari luar form (misalnya, melalui API), validasi ini tidak akan dijalankan.

Jadi, Bagaimana Sebaiknya?

Jika kita hanya ingin memastikan validasi saat data dimasukkan oleh pengguna melalui form, cukup letakkan validasi di **forms** saja. Namun, jika kita ingin validasi berjalan setiap kali objek

Validasi Token untuk Bot

model disimpan, terlepas dari cara data tersebut dimasukkan, sebaiknya letakkan validasi di **models**.

Opsi: Memilih Salah Satu

Jika kita memilih untuk melakukan validasi hanya di **forms**, maka validasi di **models** bisa dihapus. Sebaliknya, jika validasi hanya ada di **models**, form akan cukup memanggil `full_clean()` yang akan menjalankan validasi di model.

Contoh validasi jika hanya dilakukan di **forms**:

1. Hapus validasi di **models.py**:

```
def clean(self):
    # Validasi untuk memastikan token
    tetap unik
    if
    Bot.objects.exclude(id=self.id).filter(token=self.token).exists():
        raise ValidationError('Bot dengan token ini sudah ada.')
```

2. Tetap lakukan validasi di **forms.py**:

```
def clean_token(self):
    token = self.cleaned_data.get('token')
    bot_type =
    self.cleaned_data.get('bot_type')

    # Validasi token Telegram dan WhatsApp
    di form
    if bot_type == 'telegram':
        validate_telegram_token(token)
    elif bot_type == 'whatsapp':
```

Validasi Token untuk Bot

```
validate_whatsapp_token(token)

# Validasi token unik
if
Bot.objects.exclude(id=self.instance.id).f
ilter(token=token).exists():
    raise forms.ValidationError('Bot
dengan token ini sudah ada.')

return token
```

Dengan validasi di form, pesan error akan langsung ditampilkan kepada pengguna di halaman form yang di-submit.

- **Validasi di models:** Lebih aman untuk semua alur penyimpanan data, termasuk API dan skrip.
- **Validasi di forms:** Lebih ramah pengguna untuk validasi yang ditampilkan di halaman web.

Jika validasi hanya diperlukan saat pengguna mengisi form di halaman web, letakkan validasi di **forms**. Namun, jika validasi harus berlaku di seluruh aplikasi, sebaiknya letakkan di **models**.

9.7.1 Models.py

Berikut adalah kode model Bot yang telah diperbarui dengan penambahan validasi format token berdasarkan jenis bot jika ingin menggunakan validasi di model:

```
# File: apps/bots/models.py

from django.db import models
from django.contrib.auth.models import User
from django.core.exceptions import
ValidationError # Import untuk validasi
```

Validasi Token untuk Bot

```
import re # Import untuk memvalidasi pola token

class Bot(models.Model):
    BOT_CHOICES = [
        ('telegram', 'Telegram'),
        ('whatsapp', 'WhatsApp'),
    ]
    user = models.ForeignKey(User,
on_delete=models.CASCADE) # Bot akan terkait
dengan pengguna
    bot_type = models.CharField(max_length=20,
choices=BOT_CHOICES) # Jenis bot: Telegram atau
WhatsApp

    token_telegram =
models.CharField(max_length=255, unique=True,
help_text="Masukkan token bot yang unik.") #
Token Telegram
    webhook_url =
models.URLField(max_length=255, blank=True,
help_text="URL webhook bot akan diatur secara
otomatis.") # URL webhook

    is_active =
models.BooleanField(default=True,
help_text="Status bot aktif atau tidak.") #
Status aktif atau tidaknya bot

    # Pesan otomatis
    start_message = models.TextField(blank=True,
default="Selamat datang di bot!",
help_text="Pesan yang dikirim saat pengguna
mengetik /start.")
    help_message = models.TextField(blank=True,
default="Berikut cara menggunakan bot.",
help_text="Pesan yang dikirim saat pengguna
mengetik /help.")
```

Validasi Token untuk Bot

```
    messages = models.JSONField(default=list,
blank=True, help_text="Daftar pesan yang
diterima dan responsnya dalam format JSON.") #
Format JSON

    # Timestamp
    created_at =
models.DateTimeField(auto_now_add=True) #
Tanggal bot dibuat
    updated_at =
models.DateTimeField(auto_now=True) # Tanggal
terakhir bot diperbarui

    name = models.CharField(max_length=255,
blank=True, default="Bot Saya", help_text="Nama
bot.") # Nama bot
    description = models.TextField(blank=True,
default="Ini bot pertama saya",
help_text="Deskripsi bot.") # Deskripsi bot

    def __str__(self):
        return f'{self.bot_type.capitalize()}
Bot - {self.user.username}'

    # Tambahkan metode clean untuk validasi
    def clean(self):
        # Validasi nama bot tidak boleh kosong
        if not self.name.strip():
            raise ValidationError("Nama bot
tidak boleh kosong.")

        # Validasi deskripsi bot tidak lebih
dari 500 karakter
        if len(self.description) > 500:
            raise ValidationError("Deskripsi bot
tidak boleh lebih dari 500 karakter.")

        # Validasi token Telegram hanya jika
bot_type adalah Telegram
```

Validasi Token untuk Bot

```
if self.bot_type == 'telegram':
    # Pola token Telegram yang benar:
    # angka-angka diikuti dengan titik dua, lalu
    # diikuti oleh string karakter acak
    if not re.match(r'^\d+:[\w-]+$ ',
self.token_telegram):
        raise ValidationError("Token
Telegram tidak valid. Pastikan token dalam
format yang benar seperti '123456:ABC-
DEF1234ghIkl-zyx57W2v1u123ew11'.")

class Meta:
    verbose_name = "Bot"
    verbose_name_plural = "Bots"
```

Dengan penerapan ini, model **Bot** kini memiliki logika yang lebih robust untuk menangani validasi token sesuai dengan jenis bot yang dipilih pengguna.

9.7.2 Forms.py

Berikut adalah kode yang diperbarui untuk form **BotForm** dengan penerapan validasi token berdasarkan jenis bot, Jika ingin menambahkan validasi di forms:

```
# File: apps/bots/forms.py

from django import forms
from django.core.exceptions import
ValidationError
import re # Untuk memvalidasi pola token
from .models import Bot # Mengimpor model Bot

class BotForm(forms.ModelForm):
    class Meta:
```

Validasi Token untuk Bot

```
model = Bot # Menghubungkan form dengan
model Bot
    fields = ['bot_type', 'token_telegram',
'name', 'description', 'is_active',
'start_message', 'help_message'] # Field yang
ditampilkan dalam form
    widgets = {
        'name':
forms.TextInput(attrs={'class': 'form-control',
'placeholder': 'Masukkan nama bot'}),
        'description':
forms.Textarea(attrs={'class': 'form-control',
'rows': 3, 'placeholder': 'Deskripsi bot'}),
    }
    help_texts = {
        'token_telegram': 'Masukkan token
unik yang diterima dari platform Telegram.',
        'name': 'Nama bot dapat diubah
sesuai keinginan.',
        'description': 'Deskripsi singkat
tentang bot dan fungsinya.',
    }

    # Validasi khusus untuk token Telegram
    def clean_token_telegram(self):
        token_telegram =
self.cleaned_data.get('token_telegram')
        bot_type =
self.cleaned_data.get('bot_type')

        # Validasi hanya jika bot_type adalah
Telegram
        if bot_type == 'telegram':
            # Pola token Telegram: angka-angka
diikuti dengan titik dua, lalu serangkaian
karakter acak
            if token_telegram and not
re.match(r'^\d+:[\w-]+$ ', token_telegram):
```

Validasi Token untuk Bot

```
        raise ValidationError("Token
Telegram tidak valid. Pastikan format token
adalah seperti '123456:ABC-DEF1234ghIkl-
zyx57W2v1u123ew11'.")

    return token_telegram

# Validasi nama bot
def clean_name(self):
    name = self.cleaned_data.get('name')
    if not name.strip():
        raise ValidationError("Nama bot
tidak boleh kosong.")
    return name

# Validasi deskripsi bot
def clean_description(self):
    description =
self.cleaned_data.get('description')
    if len(description) > 500:
        raise ValidationError("Deskripsi bot
tidak boleh lebih dari 500 karakter.")
    return description
```

Dengan penerapan ini, form `BotForm` kini dapat memvalidasi token dengan benar berdasarkan jenis bot yang dipilih oleh pengguna.

9.7.3 Create_bot.html

Berikut adalah kode `create_bot.html` yang telah diperbarui dengan penanganan kesalahan dan penerapan form yang lebih rapi menggunakan Bootstrap:

```
<!-- File: apps/templates/bots/create_bot.html
-->
```


Validasi Token untuk Bot

```
{% extends 'dashboard/base.html' %}

{% block content %}
<div class="container mt-4">
  <!-- Card untuk Formulir -->
  <div class="card">
    <div class="card-header">
      <h5 class="card-title">Buat Bot
Baru</h5>
    </div>
    <div class="card-body">
      <!-- Form untuk Bot -->
      <form method="post">
        {% csrf_token %}

        <!-- Form Group untuk Platform
(muncul pertama) -->
        <div class="form-group">
          <label class="form-label">
            Platform
          </label>
          <div class="btn-group"
role="group" aria-label="Platform">
            <input type="radio"
name="bot_type" id="telegram" value="telegram"
{% if form.bot_type.value == 'telegram'
%}<checked{% endif %}> onclick="toggleFields()">
            <label
for="telegram">Telegram</label>

            <input type="radio"
name="bot_type" id="whatsapp" value="whatsapp"
{% if form.bot_type.value == 'whatsapp'
%}<checked{% endif %}> onclick="toggleFields()">
            <label
for="whatsapp">WhatsApp</label>
          </div>
        </div>
      </form>
    </div>
  </div>
</div>
```

Validasi Token untuk Bot

```
<!-- Form Group untuk Nama -->
<div class="form-group">
  <label
for="{{ form.name.id_for_label }}">
    Nama
  </label>
  <input type="text"
name="name" id="{{ form.name.id_for_label }}"
value="{{ form.name.value }}" class="form-
control {% if form.name.errors %}is-invalid{%
endif %}">
    {% for error in
form.name.errors %}
      <div class="invalid-
feedback">{{ error }}</div>
    {% endfor %}
  </div>

<!-- Form Group untuk Token
(hanya untuk Telegram) -->
<div class="form-group"
id="token-field">
  <label
for="{{ form.token_telegram.id_for_label }}">
    Token
  </label>
  <input type="text"
name="token_telegram"
id="{{ form.token_telegram.id_for_label }}"
value="{{ form.token_telegram.value }}"
class="form-control {% if
form.token_telegram.errors %}is-invalid{% endif
%}">
    {% for error in
form.token_telegram.errors %}
      <div class="invalid-
feedback">{{ error }}</div>
    {% endfor %}
```

Validasi Token untuk Bot

```

        </div>

        <!-- Button Submit -->
        <button type="submit" class="btn
btn-primary">
            Buat Bot
        </button>
    </form>
</div>
<!-- Modal untuk Pesan Error -->
{% if form.errors %}
    <div class="modal fade" id="errorModal"
tabindex="-1" aria-labelledby="errorModalLabel"
aria-hidden="true">
        <div class="modal-dialog">
            <div class="modal-content">
                <div class="modal-header bg-
danger text-white">
                    <h1 class="modal-title fs-5"
id="errorModalLabel">Terjadi Kesalahan</h1>
                    <button type="button"
class="btn-close" data-bs-dismiss="modal" aria-
label="Close"></button>
                </div>
                <div class="modal-body">
                    <div class="alert alert-
danger">
                        {% for field in form %}
                            {% for error in
field.errors %}
                                <p>{{ error
}}</p>
                            {% endfor %}
                        {% endfor %}
                        {% for error in
form.non_field_errors %}
                            <p>{{ error }}</p>
                        {% endfor %}
                    </div>
                </div>
            </div>
        </div>
    </div>
{% endif %}

```

Validasi Token untuk Bot

```
        </div>
    </div>
    <div class="modal-footer">
        <button type="button"
class="btn btn-secondary" data-bs-
dismiss="modal">Tutup</button>
    </div>
</div>
</div>
</div>

<script>
    // Pastikan bahwa Bootstrap JavaScript
    sudah di-load dan jQuery jika diperlukan

    document.addEventListener('DOMContentLoaded',
    function () {
        if
    (document.querySelector('#errorModal')) {
        var errorModal = new
    bootstrap.Modal(document.getElementById('errorMo
    dal'));
        errorModal.show();
    }
    });
</script>
    {% endif %}
</div>

<script>
$(document).ready(function() {
    // Tampilkan atau sembunyikan token field
    berdasarkan jenis bot yang dipilih
    $('#id_bot_type').change(function() {
        var botType = $(this).val();
        if (botType === 'whatsapp') {
            $
    ('#id_token_telegram').closest('.form-
```

Validasi Token untuk Bot

```
group').hide(); // Sembunyikan field token
Telegram
    } else {
        $
        ('#id_token_telegram').closest('.form-
group').show(); // Tampilkan field token
Telegram
    }
    }).trigger('change'); // Trigger event on
page load
});
</script>
<script>
    // Fungsi untuk
    menyembunyikan/menampilkan elemen sesuai
    platform
    function toggleFields() {
        var tokenField =
document.getElementById('token-field');
        var whatsappFields =
document.getElementById('whatsapp-fields');

        if
        (document.getElementById('telegram').checked) {
            tokenField.classList.remove('d-
none');

whatsappFields?.classList.add('d-none');
        } else if
        (document.getElementById('whatsapp').checked) {
            tokenField.classList.add('d-
none');

whatsappFields?.classList.remove('d-none');
        }
    }

    // Panggil toggleFields() saat halaman
    dimuat
```

Validasi Token untuk Bot

```
document.addEventListener('DOMContentLoaded',  
function () {  
    toggleFields();  
});  
  
</script>  
{% endblock %}
```

Dengan penerapan ini, tampilan form menjadi lebih baik, dan pengguna dapat melihat kesalahan input dengan jelas. Pastikan juga untuk memuat fungsi `add_class` di template filter untuk menambahkan kelas Bootstrap secara dinamis.

Kesimpulan Bab

Bab ini telah membahas pentingnya validasi token dalam pembuatan bot, dengan fokus pada bagaimana memastikan bahwa token yang dimasukkan pengguna adalah valid dan sesuai dengan format yang diharapkan. Validasi token tidak hanya meningkatkan keamanan aplikasi, tetapi juga memberikan pengalaman pengguna yang lebih baik, dengan menghindari kesalahan yang dapat terjadi akibat token yang tidak tepat.

Kita telah mempelajari berbagai jenis token dan penggunaannya, serta mengapa format token yang benar sangat penting. Selain itu, kita telah mengimplementasikan validasi token dalam konteks aplikasi Django, mulai dari pembuatan model yang sesuai hingga penanganan input melalui formulir. Melalui penggunaan metode validasi di level form, kita dapat menjamin bahwa hanya token yang valid yang akan disimpan dalam basis data.

Dalam menangani kesalahan input, kita menekankan pentingnya memberikan umpan balik yang jelas kepada pengguna. Dengan menampilkan pesan kesalahan yang informatif dan menggunakan komponen Bootstrap untuk notifikasi kesalahan, kita tidak hanya meningkatkan keterbacaan, tetapi juga estetika antarmuka pengguna.

Validasi Token untuk Bot

Dengan langkah-langkah yang telah kita bahas di bab ini, aplikasi kita sekarang lebih robust dan siap untuk menangani berbagai input dari pengguna dengan cara yang aman dan efisien. Selanjutnya, kita akan melanjutkan ke topik lain yang berhubungan dengan pengelolaan bot dan fungsionalitas tambahan yang dapat ditambahkan ke dalam aplikasi kita.

BAB 10 -Menyiapkan Webhook untuk Bot

Dalam dunia pengembangan bot, webhook memainkan peran krusial sebagai jembatan antara platform bot dan server aplikasi kita. Webhook memungkinkan aplikasi kita untuk menerima notifikasi secara real-time dari platform bot ketika terjadi berbagai peristiwa, seperti pesan baru atau pembaruan status.

Dengan menggunakan webhook, aplikasi kita dapat secara otomatis memproses data dan merespons interaksi pengguna tanpa perlu melakukan polling terus-menerus ke server. Di bab ini, kita akan membahas langkah-langkah untuk menyiapkan webhook dalam proyek Django yang bernama *platform_bot*, yang akan memungkinkan kita untuk menangani dan merespons data yang dikirim oleh platform bot.

10.1 Pengantar Webhook

Webhook merupakan konsep dasar dalam pengembangan aplikasi modern, terutama ketika berurusan dengan interaksi real-time antara sistem yang berbeda. Webhook adalah mekanisme yang memungkinkan satu aplikasi untuk memberikan data secara otomatis kepada aplikasi lain ketika peristiwa tertentu terjadi. Dalam konteks bot, webhook berfungsi sebagai jembatan komunikasi

Menyiapkan Webhook untuk Bot

antara platform bot (seperti Telegram atau WhatsApp) dan server Anda.

10.1.1 Apa Itu Webhook?

Webhook adalah URL endpoint yang ditetapkan untuk menerima data secara real-time dari platform eksternal. Ketika suatu peristiwa yang relevan terjadi di platform tersebut—misalnya, penerimaan pesan baru di bot—platform tersebut akan mengirimkan data terkait peristiwa tersebut ke URL webhook yang telah Anda tentukan. Ini memungkinkan server Anda untuk merespons secara langsung terhadap peristiwa tanpa harus secara aktif meminta data secara berkala.

Misalnya, jika Anda memiliki bot Telegram dan ingin agar bot tersebut dapat merespons pesan yang dikirimkan oleh pengguna, Anda akan mengonfigurasi webhook pada bot tersebut. Ketika pengguna mengirimkan pesan ke bot, Telegram akan mengirimkan data pesan ke endpoint webhook Anda. Server Anda kemudian akan memproses data tersebut dan menjalankan logika yang diperlukan, seperti mengirimkan balasan atau menyimpan informasi ke database.

Webhook adalah cara untuk memberikan aplikasi web informasi secara real-time. Berbeda dengan metode polling, di mana aplikasi secara berkala memeriksa server untuk pembaruan, webhook memungkinkan server untuk memberitahukan aplikasi secara otomatis ketika ada data baru. Ini tidak hanya mengurangi beban pada server tetapi juga membuat aplikasi kita lebih responsif terhadap interaksi pengguna.

Menyiapkan Webhook untuk Bot

Dengan menggunakan webhook, kita dapat menyederhanakan integrasi dengan platform bot. Misalnya, ketika seorang pengguna mengirimkan pesan ke bot, webhook akan mengirimkan data pesan tersebut ke server kita tanpa menunggu permintaan dari server. Hal ini memungkinkan bot kita untuk merespons pesan dengan cepat dan efisien.

10.1.2 Keuntungan Menggunakan Webhook

Menggunakan webhook memiliki beberapa keuntungan signifikan, terutama dalam hal efisiensi dan responsivitas aplikasi. Salah satu keuntungan utama adalah penghematan sumber daya. Tanpa webhook, aplikasi Anda harus terus-menerus memeriksa (polling) platform eksternal untuk mengetahui apakah ada pembaruan atau peristiwa baru. Ini dapat membebani server dan mengonsumsi bandwidth secara tidak efisien.

Dengan webhook, server Anda hanya menerima data ketika ada perubahan atau peristiwa yang relevan, sehingga mengurangi beban pada server dan meningkatkan efisiensi komunikasi. Selain itu, webhook juga memungkinkan aplikasi Anda untuk merespons peristiwa secara real-time, yang sangat penting untuk aplikasi seperti bot yang memerlukan interaksi cepat dengan pengguna.

10.1.3 Cara Kerja Webhook

Cara kerja webhook dapat dijelaskan melalui beberapa langkah kunci. Pertama, Anda harus mengatur URL webhook pada platform eksternal (misalnya, Telegram). URL ini akan menjadi end-

Menyiapkan Webhook untuk Bot

point di server Anda yang akan menerima data. Ketika peristiwa tertentu terjadi di platform eksternal, data akan dikirimkan secara otomatis ke URL webhook yang telah Anda daftarkan.

Di server Anda, endpoint webhook akan menerima request HTTP POST yang berisi data peristiwa. Misalnya, dalam kasus bot Telegram, data ini mungkin termasuk pesan yang dikirim oleh pengguna. Setelah data diterima, server Anda akan memproses data tersebut sesuai dengan logika aplikasi Anda. Ini bisa mencakup menyimpan data ke database, memproses perintah, atau mengirimkan balasan kepada pengguna.

10.1.4 Menyiapkan Webhook dalam Django

Untuk mengintegrasikan webhook dalam proyek Django Anda, Anda perlu menyiapkan beberapa komponen utama. Pertama, Anda harus membuat view Django yang dapat menangani request yang dikirimkan ke URL webhook. Selanjutnya, Anda akan menambahkan routing URL yang mengarahkan request ke view yang sesuai. Terakhir, Anda perlu menguji integrasi untuk memastikan bahwa webhook bekerja sebagaimana mestinya.

Proses ini akan memungkinkan server Anda untuk berinteraksi dengan berbagai platform bot melalui satu webhook, membuat pengelolaan bot menjadi lebih sederhana dan efisien.

Dengan pemahaman dasar tentang webhook ini, Anda akan siap untuk melanjutkan ke langkah berikutnya dalam mengimplementasikan webhook dalam proyek Django Anda.

Menyiapkan Webhook untuk Bot

Selanjutnya, kita akan membahas bagaimana membuat view untuk webhook, mengatur URL routing, dan menguji integrasi webhook untuk memastikan semuanya berjalan dengan baik.

10.2 Menyusun Dan Mengelola Webhook

Dalam implementasi webhook yang mendukung berbagai platform, seperti Telegram dan WhatsApp, penting untuk merancang sistem yang fleksibel dan dapat menangani data dari beberapa sumber. Satu webhook universal dapat digunakan untuk berbagai platform jika dirancang dengan baik, sehingga Anda tidak perlu membuat webhook terpisah untuk setiap platform.

Untuk mencapai ini, webhook Anda harus dapat memproses data dari berbagai platform dengan format yang berbeda. Ini melibatkan pengenalan dan pemrosesan data yang berbeda dari setiap platform dalam satu endpoint. Misalnya, data yang dikirim oleh Telegram berbeda dengan data yang dikirim oleh WhatsApp, tetapi Anda bisa mengatur sistem untuk memproses keduanya.

Sebagai contoh, Anda bisa menggunakan sebuah view di Django yang mengidentifikasi sumber data berdasarkan informasi yang diterima, seperti header atau parameter dalam request. Dengan pendekatan ini, satu endpoint webhook dapat digunakan untuk menerima dan memproses data dari berbagai platform.

10.2.1 Menangani Data dari Webhook

Setelah webhook terpasang, Anda perlu menangani data yang diterima dari berbagai platform. Data yang diterima dari webhook

Menyiapkan Webhook untuk Bot

biasanya dalam format JSON, dan setiap platform mungkin memiliki struktur data yang berbeda.

Misalnya, data dari Telegram biasanya berisi objek JSON dengan informasi tentang pesan, pengirim, dan lainnya. Data dari WhatsApp mungkin memiliki struktur yang berbeda, seperti informasi tentang pesan teks atau media. Untuk memproses data ini, Anda harus menulis kode yang dapat mengidentifikasi dan memproses format data yang berbeda.

Dengan cara ini, Anda dapat mengelola webhook yang universal dan menangani data dari berbagai platform dalam satu aplikasi Django. Selanjutnya, kita akan membahas bagaimana mengatur URL routing untuk webhook dan menguji integrasi untuk memastikan semuanya berfungsi sesuai harapan.

10.3 Menyiapkan Webhook Telegram

Setelah kita memahami dasar-dasar webhook pada sub bab sebelumnya, kita akan melanjutkan dengan mempersiapkan webhook untuk platform Telegram. Webhook ini memungkinkan bot Telegram menerima dan memproses pesan yang masuk secara otomatis. Untuk melakukan ini, kita harus memastikan bahwa aplikasi kita dapat menangani pembaruan yang dikirimkan oleh Telegram dengan benar.

Menyiapkan Webhook untuk Bot

10.3.1 Mengonfigurasi Webhook untuk Telegram

Langkah pertama dalam menyiapkan webhook Telegram adalah membuat endpoint yang akan menerima dan memproses pembaruan (update) yang dikirimkan oleh server Telegram. Telegram akan mengirimkan data JSON ke URL yang kita tetapkan sebagai webhook, dan tugas kita adalah memproses data tersebut dan meresponsnya dengan benar. Untuk menangani webhook ini, kita akan membuat sebuah view khusus di Django yang akan menangani request dari Telegram.

Berikut adalah kodenya:

```
# File: apps/webhook/views.py

import json
import requests
from django.http import HttpResponse,
HttpResponseBadRequest
from django.views.decorators.csrf import
csrf_exempt
import logging
from apps.bots.services.telegram import
handle_update as handle_telegram_update

# Logger untuk mencatat informasi dan error yang
terjadi
logger = logging.getLogger(__name__)

@csrf_exempt # Menonaktifkan CSRF protection
untuk endpoint ini
def telegram_webhook(request):
    if request.method == 'POST': # Memastikan
request adalah POST
```

Menyiapkan Webhook untuk Bot

```
try:
    # Memuat data JSON dari body request
    update =
json.loads(request.body.decode('utf-8'))
    logger.info(f"Telegram update
received: {update}") # Mencatat pembaruan yang
diterima

    # Memastikan bahwa ada 'message' di
dalam pembaruan yang diterima
    if 'message' in update:
        handle_telegram_update(update)
# Memproses pembaruan melalui service yang telah
dibuat

    return HttpResponse('ok') #
Mengembalikan respons HTTP 200 OK

    # Jika tidak ada 'message' dalam
pembaruan, kembalikan respons HTTP 400
    return HttpResponseBadRequest('No
message found in update')

except json.JSONDecodeError: #
Menangani kesalahan jika JSON tidak valid
    logger.error("Invalid JSON
received.")
    return
HttpResponseBadRequest('Invalid JSON format')

except Exception as e: # Menangani
semua error yang tidak terduga
    logger.error(f"Error processing
Telegram webhook: {e}")
    return HttpResponse('Error',
status=500)

# Jika request bukan POST, kembalikan
respons HTTP 400 Bad Request
```


Menyiapkan Webhook untuk Bot

```
return HttpResponseBadRequest('Bad Request',  
status=400)
```

Pada kode di atas, kita menyiapkan sebuah view yang akan menangani request POST dari Telegram. Saat Telegram mengirimkan pembaruan (update) ke webhook kita, server kita akan menerima data dalam format JSON. Dalam JSON tersebut, kita perlu memastikan bahwa data tersebut valid, khususnya bahwa ada pesan yang dikirimkan (message). Jika pesan ditemukan, maka pembaruan tersebut akan diproses oleh fungsi `handle_telegram_update`, yang berada di dalam service bot Telegram kita. Jika tidak ada pesan, kita akan mengembalikan respons HTTP 400, yang menandakan bahwa ada masalah dengan request yang diterima. Kita juga memastikan bahwa setiap kesalahan yang terjadi akan dicatat dalam log menggunakan Python's logging module.

10.3.2 Mengatur Webhook Telegram

Setelah kita membuat endpoint yang menangani pembaruan, langkah berikutnya adalah mengatur webhook itu sendiri di server Telegram. Ini melibatkan pemanggilan API Telegram untuk menghubungkan URL aplikasi kita dengan bot yang telah dibuat. Untuk melakukan ini, kita perlu menggunakan token bot yang telah didaftarkan pada platform Telegram, serta URL aplikasi kita yang akan digunakan sebagai webhook.

Berikut adalah kode untuk mengatur webhook Telegram:

```
# File: apps/webhook/views.py  
def set_telegram_webhook(token_telegram, url):
```

Menyiapkan Webhook untuk Bot

```
# Mengonfigurasi URL API Telegram dengan
token bot
TELEGRAM_API_URL =
f'https://api.telegram.org/bot{token_telegram}/'

# Mengirim request ke API Telegram untuk
mengatur webhook
response = requests.post(TELEGRAM_API_URL +
"setWebhook?url=" + url).json()

# Memeriksa apakah respons dari Telegram
menunjukkan kesuksesan
if not response.get('ok'):
    logger.error(f"Failed to set Telegram
webhook: {response}") # Log jika gagal
else:
    logger.info(f"Telegram webhook set
successfully: {response}") # Log jika berhasil

return response
```

Pada fungsi di atas, kita memanggil API Telegram untuk mengatur webhook. Kita mengirimkan token bot yang terdaftar dan URL aplikasi kita yang digunakan sebagai endpoint webhook. Jika webhook berhasil diatur, Telegram akan mengembalikan respons yang menunjukkan bahwa operasi ini berhasil, dan kita akan mencatat informasi tersebut di dalam log. Namun, jika terjadi kesalahan dalam pengaturan webhook, kita juga akan mencatat kesalahan tersebut di log.

10.4 Menyiapkan Webhook Whatsapp

Setelah sebelumnya kita menyiapkan webhook untuk Telegram, selanjutnya kita akan membahas cara menyiapkan webhook un-

Menyiapkan Webhook untuk Bot

tuk platform WhatsApp. Berbeda dengan Telegram, webhook WhatsApp menggunakan layanan pihak ketiga seperti Twilio untuk mengirim dan menerima pesan. Oleh karena itu, kita perlu memastikan integrasi antara Twilio dan aplikasi kita berjalan lancar.

10.4.1 Mengonfigurasi Webhook untuk WhatsApp

Sama seperti Telegram, kita juga perlu membuat endpoint di Django yang akan menangani pembaruan dari WhatsApp. Twilio akan mengirimkan data berupa pesan dan informasi terkait pengiriman dalam format yang sedikit berbeda, karena payload-nya biasanya dikirim sebagai form data, bukan sebagai JSON.

Berikut adalah kode untuk mengonfigurasi webhook WhatsApp:

```
# File: apps/webhook/views.py

import json
import requests
from django.http import HttpResponse,
HttpResponseBadRequest
from django.views.decorators.csrf import
csrf_exempt
import logging
from apps.bots.services.whatsapp import
handle_update as handle_whatsapp_update

# Logger untuk mencatat informasi dan error yang
terjadi
logger = logging.getLogger(__name__)

@csrf_exempt # Menonaktifkan CSRF protection
untuk endpoint ini
def whatsapp_webhook(request):
```

Menyiapkan Webhook untuk Bot

```
if request.method == 'POST': # Memastikan
    bahwa request adalah POST
    try:
        # Twilio mengirimkan data sebagai
        form-urlencoded, kita ambil dari request.POST
        if 'From' in request.POST and 'Body'
in request.POST:
            # Membuat struktur pembaruan
            sesuai dengan data yang dibutuhkan
            update = {
                'From':
request.POST['From'], # Nomor pengirim
                'Body': request.POST['Body']
# Isi pesan
            }

            # Memproses pembaruan
            menggunakan service untuk WhatsApp
            return
            handle_whatsapp_update(update)

            # Jika tidak ada data 'From' dan
            'Body', kembalikan respons HTTP 200 OK
            return HttpResponse('ok')

    except Exception as e: # Menangani
    semua error yang mungkin terjadi
        logger.error(f"Error processing
WhatsApp webhook: {e}")
        return
        HttpResponseBadRequest('Failed to process
update') # Mengembalikan respons HTTP 400 jika
        terjadi error

        # Jika request bukan POST, kembalikan
        respons HTTP 400 Bad Request
        return HttpResponseBadRequest('Bad Request')
```

Menyiapkan Webhook untuk Bot

Pada kode di atas, kita membuat view yang menerima request POST dari Twilio, di mana pesan dari WhatsApp dikirim dalam bentuk form-urlencoded. Data utama yang kita butuhkan adalah **From** (nomor pengirim) dan **Body** (isi pesan). Jika kedua data tersebut ada, kita membentuk pembaruan (**update**) yang akan diproses oleh fungsi `handle_whatsapp_update`, yang telah disiapkan di service bot WhatsApp. Jika tidak ada data yang sesuai, kita tetap mengembalikan respons 200 OK untuk menghindari error lebih lanjut dari sisi Twilio.

Fungsi ini juga mencatat kesalahan apa pun yang mungkin terjadi selama proses webhook dalam log, sehingga kita bisa dengan mudah melacak masalah jika ada.

10.4.2 Mengatur Webhook WhatsApp melalui Twilio

Tidak seperti Telegram, untuk WhatsApp, kita tidak bisa mengatur webhook langsung melalui API. Sebagai gantinya, kita harus mengonfigurasi webhook ini melalui dashboard Twilio. Twilio menyediakan interface untuk menghubungkan nomor WhatsApp yang telah terdaftar dengan URL aplikasi kita yang akan menangani webhook. Kode berikut hanya berfungsi sebagai pengingat bagi kita bahwa konfigurasi webhook harus dilakukan melalui Twilio:

```
# File: apps/webhook/views.py

def set_whatsapp_webhook(url):
    # Menampilkan informasi bahwa webhook harus
    diatur melalui dashboard Twilio
```

Menyiapkan Webhook untuk Bot

```
logger.info(f"Webhook WhatsApp harus diatur  
melalui dashboard Twilio ke URL: {url}")  
return {"status": "Webhook configuration  
needed via Twilio dashboard."}
```

Fungsi ini mencatat bahwa konfigurasi webhook WhatsApp tidak bisa diatur langsung melalui API seperti di Telegram, melainkan harus dilakukan secara manual melalui dashboard Twilio. Twilio memerlukan URL webhook yang akan digunakan untuk mengirimkan pesan yang masuk ke aplikasi kita.

10.4.3 Menghubungkan URL Webhook ke Twilio

Setelah kita memiliki URL yang akan menangani webhook WhatsApp, kita harus menghubungkan URL tersebut ke nomor WhatsApp yang telah kita daftarkan di Twilio.

Langkah-langkah ini bisa dilakukan melalui dashboard Twilio:

1. Masuk ke akun Twilio di dashboard mereka.
2. Pilih nomor WhatsApp yang telah terdaftar.
3. Di bagian "Messaging", tambahkan URL aplikasi kita pada kolom "Webhook".
4. Pastikan metode request yang dipilih adalah POST.
5. Simpan perubahan.

Dengan demikian, setiap pesan yang dikirim ke nomor WhatsApp yang terdaftar di Twilio akan dikirimkan ke URL webhook yang telah kita tetapkan di aplikasi Django kita.

Menyiapkan Webhook untuk Bot

10.4.4 Menjalankan dan Menguji Webhook

Setelah webhook diatur melalui dashboard Twilio, kita dapat menguji apakah webhook berfungsi dengan baik. Twilio akan mengirimkan pesan yang masuk ke URL webhook, dan aplikasi kita akan memproses pesan tersebut menggunakan fungsi `what-sapp_webhook` yang telah kita buat sebelumnya.

Untuk memeriksa apakah webhook berfungsi dengan benar, kita bisa melihat log aplikasi untuk memastikan bahwa pesan dari Twilio berhasil diproses. Jika ada kesalahan, Twilio juga akan memberikan detail error melalui dashboard mereka, sehingga kita bisa melakukan troubleshooting dengan mudah.

Pada tahap ini, webhook untuk WhatsApp sudah siap digunakan, dan aplikasi kita sekarang dapat menerima dan memproses pesan WhatsApp yang masuk. Ini memungkinkan bot kita merespons pengguna secara otomatis dan real-time, sama seperti pada platform Telegram.

10.5 Kode Lengkap Views.py

Berikut adalah kode lengkap dengan beberapa perbaikan kecil :

```
# File: apps/webhook/views.py

import json
import requests
from django.http import HttpResponse,
HttpResponseBadRequest
```

Menyiapkan Webhook untuk Bot

```
from django.views.decorators.csrf import
csrf_exempt
import logging
from apps.bots.services.telegram import
handle_update as handle_telegram_update
from apps.bots.services.whatsapp import
handle_update as handle_whatsapp_update

logger = logging.getLogger(__name__)

@csrf_exempt
def telegram_webhook(request):
    if request.method == 'POST':
        try:
            update =
json.loads(request.body.decode('utf-8'))
            logger.info(f"Telegram update
received: {update}")

            if 'message' in update:
                handle_telegram_update(update)
                return HttpResponse('ok') #
Pastikan mengembalikan respons yang valid

            return HttpResponseBadRequest('No
message found in update')
        except json.JSONDecodeError:
            logger.error("Invalid JSON
received.")
            return
HttpResponseBadRequest('Invalid JSON format')
        except Exception as e:
            logger.error(f"Error processing
Telegram webhook: {e}")
            return HttpResponse('Error',
status=500)
            return HttpResponseBadRequest('Bad Request',
status=400)
```


Menyiapkan Webhook untuk Bot

```
@csrf_exempt
def whatsapp_webhook(request):
    if request.method == 'POST':
        try:
            # Jika payload adalah form-
            urlencoded, ambil dari request.POST
            if 'From' in request.POST and 'Body'
in request.POST:
                update = {
                    'From':
request.POST['From'],
                    'Body': request.POST['Body']
                }
                return
            handle_whatsapp_update(update)

            return HttpResponse('ok')

        except Exception as e:
            logger.error(f"Error processing
webhook: {e}")
            return
            HttpResponseBadRequest('Failed to process
update')
        else:
            return HttpResponseBadRequest('Bad
Request')

def set_telegram_webhook(token_telegram, url):
    TELEGRAM_API_URL =
f'https://api.telegram.org/bot{token_telegram}/'
    response = requests.post(TELEGRAM_API_URL +
"setWebhook?url=" + url).json()

    if not response.get('ok'):
        logger.error(f"Failed to set Telegram
webhook: {response}")
```

Menyiapkan Webhook untuk Bot

```
    else:
        logger.info(f"Telegram webhook set successfully: {response}")

    return response

def set_whatsapp_webhook(url):
    logger.info(f"Webhook WhatsApp harus diatur melalui dashboard Twilio ke URL: {url}")
    return {"status": "Webhook configuration needed via Twilio dashboard."}
```

Dengan struktur ini, Anda bisa dengan mudah menambahkan dukungan untuk platform lain, seperti Facebook Messenger, Slack, dll.

10.6 Mengatur Routing URL

Untuk membuat view dapat diakses melalui URL, kita perlu mengonfigurasi routing URL di Django. Tambahkan URL endpoint yang akan digunakan oleh webhook untuk mengirimkan data.

Selanjutnya, kita perlu menambahkan URL routing untuk webhook agar Django tahu bagaimana menangani request yang masuk. Tambahkan konfigurasi URL berikut di file `urls.py` pada aplikasi `webhook`:

```
# File: apps/webhook/urls.py

from django.urls import path
from .views import telegram_webhook,
whatsapp_webhook
```

Menyiapkan Webhook untuk Bot

```
urlpatterns = [  
    # Endpoint webhook untuk Telegram  
    path('telegram/', telegram_webhook,  
name='telegram_webhook'),  
    # Endpoint webhook untuk WhatsApp  
    path('whatsapp/', whatsapp_webhook,  
name='whatsapp_webhook'),  
]
```

Selanjutnya, kita perlu memastikan bahwa URL aplikasi webhook termasuk dalam routing utama proyek. Buka file `platform_bot/urls.py` dan tambahkan konfigurasi berikut:

```
# File: platform_bot/urls.py  
  
from django.contrib import admin  
from django.urls import path, include  
  
urlpatterns = [  
    path('admin/', admin.site.urls),  
    path('', include('apps.webhook.urls')), #  
Menyertakan URL aplikasi webhook  
]
```

Dengan konfigurasi ini, halaman webhook akan ditampilkan ketika pengguna mengunjungi URL root dari aplikasi.

10.7 Konfigurasi `Settings.py`

Untuk menguji integrasi webhook dalam proyek Django, kita memerlukan URL publik yang dapat diakses oleh platform bot seperti Telegram atau WhatsApp. Salah satu cara untuk mendapatkan URL publik sementara adalah dengan menggunakan alat seperti ngrok. Ngrok memungkinkan kita untuk membuat

Menyiapkan Webhook untuk Bot

terowongan ke server lokal kita dan mendapatkan URL yang dapat diakses dari internet.

Menjalankan Ngrok

Pertama, jalankan perintah berikut di terminal untuk memulai ngrok dan membuat terowongan ke server lokal yang berjalan di port 8000:

```
$ ngrok http 8000
```

Perintah ini akan memberikan URL sementara yang dapat diakses dari internet. Ngrok akan menampilkan URL ini di terminal, seperti `https://ngrok-random_id.app`. Gunakan URL ini untuk konfigurasi webhook.

Mengonfigurasi `settings.py`

Selanjutnya, kita perlu memperbarui file `settings.py` untuk mencakup URL webhook dan mengatur `ALLOWED_HOSTS` yang diperlukan. Berikut adalah langkah-langkah yang harus diikuti:

Mengatur `ALLOWED_HOSTS`

Parameter `ALLOWED_HOSTS` di Django digunakan untuk menentukan daftar nama host/domain yang diperbolehkan untuk mengakses aplikasi Django. Ketika `ALLOWED_HOSTS` diatur ke `['*']`, ini berarti aplikasi Django akan menerima request dari semua host.

Menyiapkan Webhook untuk Bot

Pengaturan ini biasanya digunakan saat pengembangan untuk menghindari masalah CORS (Cross-Origin Resource Sharing) atau masalah lainnya yang mungkin muncul jika hanya menggunakan domain lokal. Namun, pada lingkungan produksi, `ALLOWED_HOSTS` sebaiknya diatur ke domain atau IP yang spesifik untuk mencegah akses yang tidak diinginkan.

Update `ALLOWED_HOSTS` seperti berikut:

```
# File: platform_bot/settings.py

ALLOWED_HOSTS = ['*'] # Mengizinkan semua host,
digunakan saat pengembangan
```

Mengirimkan POST Request dan Menguji Webhook

Setelah mengonfigurasi `ALLOWED_HOSTS`, kita dapat menguji integrasi webhook. Kirimkan POST request ke URL `/get-post/` menggunakan platform seperti Telegram atau WhatsApp untuk memastikan bahwa webhook menerima dan memproses update dengan benar.

Dengan langkah-langkah ini, kita telah berhasil menyiapkan webhook di proyek Django kita. Ngrok menyediakan URL sementara yang memungkinkan kita menguji aplikasi kita dari luar, sementara pengaturan `ALLOWED_HOSTS` memastikan aplikasi dapat menerima request dari semua sumber saat pengembangan.

Di bagian selanjutnya, kita akan membahas bagaimana mengintegrasikan bot dengan Django dan mengelola data yang diterima

Menyiapkan Webhook untuk Bot

dari webhook, melanjutkan proses untuk memastikan aplikasi berfungsi dengan baik dalam skenario nyata.

Kesimpulan Bab

Pada Bagian ini, kita telah membahas secara menyeluruh mengenai webhook, yang merupakan langkah penting dalam memastikan bot dapat merespons update atau pesan dari berbagai platform. Dimulai dengan ***Pengantar Webhook***, kita memahami pentingnya webhook dalam menghubungkan aplikasi eksternal dengan bot, memungkinkan komunikasi yang real-time dan otomatis.

Selanjutnya, pada bagian ***Menyiapkan view Webhook***, kita belajar menyusun dan mengelola webhook secara efektif untuk menangani berbagai jenis update dari platform seperti Telegram dan WhatsApp. Proses ini termasuk pengaturan webhook untuk masing-masing platform dengan penanganan payload yang berbeda.

Pada sub-bab ***Menangani Update dari Berbagai Platform***, kita telah membuat handler untuk bot agar bisa menangani berbagai jenis data yang masuk, seperti pesan dari Telegram dan notifikasi dari WhatsApp. Integrasi ini dikelola melalui fungsi webhook yang disusun di file `views.py`.

Mengatur Webhook untuk Platform Berbeda membahas cara menghubungkan bot dengan platform melalui API masing-masing, di mana kita mempelajari cara mengatur webhook Telegram dan WhatsApp melalui endpoint yang sesuai. Bagian ini juga melibatkan konfigurasi URL routing di Django melalui `urls.py`, yang memastikan request dari platform ditangani dengan benar oleh server.

Menyiapkan Webhook untuk Bot

Pada sub-bab ***Konfigurasi settings.py***, kita mempelajari pengaturan server Django yang diperlukan untuk mendukung webhook, termasuk pengaturan URL yang diperlukan agar bot dapat berfungsi dengan baik.

Dengan menyelesaikan bagian ini, Anda telah memahami prinsip-prinsip dan praktik terbaik dalam mengelola webhook untuk bot. Bab ini juga memberikan fondasi teknis yang solid untuk melanjutkan integrasi bot pada berbagai platform dan meningkatkan kemampuan bot dalam menangani komunikasi real-time.

BAB 11 - Integrasi Bot dengan Webhook

Dalam bab ini kita akan membahas integrasi webhook dengan bot, Mulai dari pengertian dan juga implementasi integrasi webhook dengan bot sehingga kita bisa membuat bot dan juga bot nya bisa berjalan dengan webhook yang sudah kita siapkan.

Integrasi antara bot dan webhook merupakan langkah penting dalam pengembangan aplikasi berbasis bot. Proses ini memungkinkan bot untuk menerima dan mengirimkan data secara real-time, berinteraksi dengan pengguna, dan mengelola perintah dengan lebih efisien. Dengan memahami cara mengintegrasikan bot dengan webhook, kita dapat memanfaatkan potensi penuh dari platform yang kita gunakan, seperti Telegram dan WhatsApp.

11.1 Pengantar Integrasi Webhook Dengan Bot

Saat membangun bot untuk berbagai platform seperti Telegram atau WhatsApp, salah satu komponen krusial yang harus diperhatikan adalah integrasi webhook. Webhook berfungsi sebagai mekanisme komunikasi antara server Anda dan platform tempat bot Anda beroperasi. Integrasi webhook memungkinkan bot Anda menerima pembaruan atau notifikasi secara real-time dari platform tersebut, seperti pesan baru atau perintah dari pengguna.

11.1.1 Apa Itu Integrasi Webhook?

Integrasi webhook adalah proses menghubungkan aplikasi atau sistem Anda dengan layanan eksternal melalui URL yang ditentukan. Dalam konteks bot, webhook berfungsi untuk mengirimkan informasi dari platform ke server Anda setiap kali ada peristiwa yang relevan. Misalnya, ketika pengguna mengirimkan pesan ke bot di Telegram, webhook akan mengirimkan data pesan tersebut ke URL webhook yang telah Anda konfigurasi di server Anda.

Integrasi webhook mengacu pada pengaturan komunikasi antara bot dan platform melalui URL tertentu yang berfungsi sebagai titik akhir. Ketika sebuah peristiwa terjadi di platform, seperti pesan yang diterima atau perintah yang diberikan, platform tersebut mengirimkan data ke URL webhook yang telah ditentukan. Dalam konteks bot, webhook berfungsi sebagai jembatan yang menghubungkan bot dengan platform, memungkinkan bot untuk menerima pembaruan dan merespons tindakan pengguna dengan cepat.

Webhook bekerja berdasarkan prinsip *push rather than pull*, di mana platform mengirimkan data ke bot secara otomatis tanpa perlu permintaan dari sisi bot. Hal ini sangat menguntungkan, karena mengurangi latensi dan memastikan bahwa bot selalu menerima informasi terbaru. Misalnya, ketika seorang pengguna mengirimkan pesan ke bot di Telegram, platform akan mengirimkan pembaruan tersebut ke URL webhook yang telah ditentukan. Bot kemudian dapat memproses informasi ini dan memberikan respon yang sesuai.

Integrasi Bot dengan Webhook

Pada dasarnya, webhook adalah HTTP callback yang memungkinkan server Anda mendapatkan data secara otomatis dari platform lain tanpa harus melakukan polling secara terus-menerus. Ketika platform mengirimkan pembaruan ke URL webhook Anda, server Anda dapat memproses informasi tersebut dan meresponsnya sesuai kebutuhan. Ini sangat berguna untuk bot yang memerlukan interaksi real-time dengan pengguna.

Untuk mengatur webhook, kita perlu menentukan URL yang akan digunakan oleh platform untuk mengirimkan data. Dalam proyek Django kita, konfigurasi ini dilakukan melalui pengaturan di `settings.py`. Setelah URL webhook ditentukan, kita harus mengimplementasikan logika dalam views untuk menangani pembaruan yang diterima dari berbagai platform. Berikut adalah contoh kode untuk mengatur webhook Telegram di `views.py`:

```
# File: apps/webhook/views.py

import requests

def set_telegram_webhook(token_telegram, url):
    TELEGRAM_API_URL =
f'https://api.telegram.org/bot{token_telegram}/'
    response = requests.post(TELEGRAM_API_URL +
"setWebhook?url=" + url).json()

    if not response.get('ok'):
        logger.error(f"Failed to set Telegram
webhook: {response}")
    else:
        logger.info(f"Telegram webhook set
successfully: {response}")
```

```
return response
```

Dalam contoh di atas, kita mengatur webhook dengan mengirimkan permintaan ke API Telegram, memberikan URL yang akan menerima pembaruan. Dengan langkah-langkah ini, kita akan memastikan bahwa bot kita terhubung dengan baik dan dapat berfungsi secara optimal.

Dengan pemahaman tentang konsep dasar integrasi webhook, kita dapat melanjutkan untuk membahas pentingnya pengujian dalam proses ini, sehingga bot yang kita kembangkan dapat bekerja dengan baik di berbagai kondisi.

11.1.2 Pentingnya Pengujian dalam Integrasi Webhook

Pengujian merupakan tahap penting dalam proses integrasi webhook. Tanpa pengujian yang memadai, Anda mungkin menghadapi masalah yang sulit dideteksi, seperti webhook yang tidak merespons atau kesalahan dalam pemrosesan data.

Pengujian membantu memastikan bahwa webhook Anda berfungsi dengan baik dan dapat menangani berbagai jenis pembaruan dari platform.

Pengujian dalam integrasi webhook merupakan langkah yang sangat krusial untuk memastikan bahwa bot berfungsi sesuai harapan dan dapat berinteraksi dengan pengguna secara efisien. Tanpa pengujian yang memadai, berbagai masalah dapat muncul, mulai dari kesalahan komunikasi hingga kegagalan dalam merespons

Integrasi Bot dengan Webhook

perintah pengguna. Oleh karena itu, penting bagi pengembang untuk melakukan serangkaian pengujian guna memastikan kehandalan dan performa bot yang optimal.

Ada beberapa alasan mengapa pengujian webhook sangat penting:

1. **Validasi Fungsi Webhook:** Pengujian memastikan bahwa webhook yang Anda konfigurasi benar-benar menerima dan memproses pembaruan dari platform. Ini penting untuk memastikan bahwa bot Anda dapat berfungsi dengan baik dalam situasi nyata.
2. **Deteksi Kesalahan:** Pengujian memungkinkan Anda untuk menemukan dan memperbaiki kesalahan dalam kode yang menangani webhook. Ini bisa berupa kesalahan dalam pemrosesan data, masalah koneksi, atau kesalahan dalam konfigurasi.
3. **Keandalan dan Stabilitas:** Dengan melakukan pengujian, Anda dapat memastikan bahwa webhook Anda stabil dan dapat diandalkan. Ini membantu dalam mencegah gangguan pada layanan bot Anda dan memastikan pengalaman pengguna yang mulus.

Untuk menguji webhook, Anda dapat menggunakan berbagai alat dan teknik. Salah satu metode yang umum adalah menggunakan alat seperti Postman untuk mengirimkan permintaan HTTP POST ke URL webhook Anda dan memeriksa respons yang diterima. Selain itu, platform tempat bot Anda beroperasi sering kali menyediakan alat atau fitur untuk memeriksa dan menguji webhook secara langsung.

Integrasi Bot dengan Webhook

Ada beberapa jenis pengujian yang perlu dilakukan dalam proses integrasi webhook. Pertama, **unit test** digunakan untuk menguji fungsi-fungsi individual dalam kode secara terpisah. Misalnya, kita bisa menguji fungsi yang menangani pembaruan dari platform, memastikan bahwa fungsi tersebut dapat memproses data dengan benar. Di `tests.py`, kita bisa menulis pengujian sebagai berikut:

```
# File: apps/webhook/tests.py

from django.test import TestCase
from .views import handle_update # Import
fungsi yang akan diuji

class WebhookTests(TestCase):
    def test_handle_update(self):
        # Simulasi pembaruan yang diterima
        update = {
            'message': {
                'chat': {'id': 123},
                'text': '/start'
            }
        }
        response = handle_update(update)
        self.assertEqual(response, "Selamat
datang!") # Pastikan output sesuai
```

Selain unit test, **integration test** juga penting untuk memastikan bahwa berbagai komponen sistem berfungsi dengan baik secara bersamaan. Dalam hal ini, kita bisa menguji keseluruhan alur komunikasi antara bot dan platform. Contoh situasi yang bisa diuji adalah ketika pengguna mengirimkan pesan ke bot, apakah bot

Integrasi Bot dengan Webhook

dapat menerima pesan tersebut dan mengirimkan respons yang tepat.

Situasi yang bisa terjadi jika pengujian diabaikan sangat beragam. Misalnya, bot mungkin tidak merespons dengan baik ketika pengguna mengirimkan perintah tertentu, atau bahkan gagal beroperasi sama sekali jika terjadi kesalahan pada integrasi webhook. Hal ini bisa menyebabkan pengalaman pengguna yang buruk, yang berpotensi mengurangi kepercayaan pengguna terhadap aplikasi yang kita kembangkan. Misalkan, ketika pengguna mengirimkan pesan ke bot, dan bot tidak dapat memproses perintah tersebut karena kesalahan dalam pengaturan webhook. Tanpa pengujian yang memadai, kesalahan seperti ini mungkin tidak terdeteksi hingga aplikasi berada di lingkungan produksi, menyebabkan frustrasi bagi pengguna.

Oleh karena itu, pengujian bukan hanya sekadar langkah tambahan, tetapi bagian integral dari proses pengembangan yang membantu memastikan bahwa bot dan webhook terintegrasi dengan baik. Dengan melakukan pengujian secara menyeluruh, kita dapat mencegah masalah sebelum mereka terjadi dan meningkatkan kualitas produk akhir.

Dengan memahami konsep integrasi webhook dan pentingnya pengujian, Anda dapat memastikan bahwa bot Anda berfungsi dengan baik dan dapat berinteraksi dengan pengguna secara real-time. Selanjutnya, kita akan membahas bagaimana menyiapkan integrasi webhook untuk bot di platform Anda.

11.2 Menyiapkan Integrasi Webhook Untuk Bot

Dalam langkah-langkah berikut, kita akan membahas proses mengintegrasikan bot dengan webhook secara mendetail. Integrasi ini memungkinkan bot untuk menerima dan memproses pesan dari pengguna secara real-time, menjadikannya alat komunikasi yang efektif dan responsif. Sebelum memulai proses integrasi, ada beberapa persiapan yang perlu dilakukan agar semuanya berjalan dengan lancar.

Pertama-tama, pastikan bahwa proyek Django Anda sudah terkonfigurasi dengan benar. Hal ini meliputi pengaturan file `settings.py`, di mana kita perlu menambahkan URL webhook yang akan digunakan oleh bot. Sebagai contoh, kita bisa menambahkan URL berikut:

```
# File: settings.py

WEBHOOK_URL = 'https://example.com/webhook/' #
Gantilah dengan URL webhook yang sesuai
```

Setelah memastikan bahwa pengaturan dasar sudah benar, langkah selanjutnya adalah menyiapkan view yang akan menangani pembaruan dari bot. View ini akan menerima data dari webhook dan memprosesnya sesuai dengan logika yang telah ditentukan. Disini kita bisa sudah membuat view berikut di `views.py`:

```
# File: apps/webhook/views.py

@csrf_exempt
def telegram_webhook(request):
```


Integrasi Bot dengan Webhook

```
# ...  
  
@csrf_exempt  
def whatsapp_webhook(request):  
  
# ...
```

Dengan view yang telah disiapkan, kita juga sudah menambahkan routing URL untuk mengarahkan permintaan ke view tersebut. Ini dapat dilakukan dengan menambahkan entri baru di `urls.py`:

```
# File: apps/webhook/urls.py  
  
from django.urls import path  
from .views import telegram_webhook,  
whatsapp_webhook  
  
urlpatterns = [  
    # Endpoint webhook untuk Telegram  
    path('telegram/', telegram_webhook,  
name='telegram_webhook'),  
    # Endpoint webhook untuk WhatsApp  
    path('whatsapp/', whatsapp_webhook,  
name='whatsapp_webhook'),  
]
```

Setelah langkah-langkah ini selesai, kita perlu mengonfigurasi bot di platform yang kita gunakan, misalnya Telegram atau WhatsApp, untuk menggunakan URL webhook yang telah kita buat. Pada tahap ini, pastikan token bot yang diperoleh dari platform terkonfigurasi dengan benar di aplikasi kita.

Integrasi Bot dengan Webhook

Dengan semua pengaturan di atas, integrasi antara bot dan webhook kini sudah siap untuk diuji. Penting untuk melakukan pengujian untuk memastikan bahwa semua komponen berfungsi dengan baik dan dapat berkomunikasi satu sama lain. Dalam proses ini, kita akan dapat mengidentifikasi potensi masalah yang mungkin muncul, sehingga kita dapat memperbaikinya sebelum aplikasi dihadapkan pada pengguna.

Proses mengintegrasikan bot dengan webhook ini bukan hanya sekadar langkah teknis, tetapi juga membangun fondasi yang kuat untuk pengalaman pengguna yang lebih baik dan responsif. Dengan mengikuti langkah-langkah ini, kita akan mampu menciptakan sistem yang handal dan efisien, yang mampu memenuhi kebutuhan pengguna secara real-time.

11.3 Implementasi Integrasi Webhook Dan Bot

Setelah menyiapkan integrasi webhook, langkah berikutnya adalah mengintegrasikan webhook dengan bot secara praktis.

Pada tahap ini, kita akan memperbaiki tampilan (views) di `apps/bots/views.py` dan `template create_bot.html` agar fungsionalitas pembuatan bot dapat terintegrasi dengan webhook. Kode yang diperbarui memungkinkan pengguna untuk membuat bot dan secara otomatis mengatur webhook yang sesuai berdasarkan platform yang dipilih.

Integrasi Bot dengan Webhook

11.3.1 Integrasi di View apps/bots

Sekarang, perbarui view di `apps/bots/views.py` untuk menggunakan handler yang telah dibuat. Kita perlu memastikan bahwa view webhook memanggil handler yang sesuai berdasarkan platform yang mengirimkan data.

Berikut adalah kode yang telah diperbaiki:

```
# File: apps/bots/views.py

from django.shortcuts import render, redirect,
get_object_or_404
from django.contrib.auth.decorators import
login_required
from django.contrib import messages # Tambahkan
import ini untuk menggunakan messages
from .forms import BotForm
from .models import Bot
from django.conf import settings
from apps.webhook.views import
set_telegram_webhook, set_whatsapp_webhook

WEBHOOK_URL = settings.WEBHOOK_URL

@login_required
def create_bot(request):
    if request.method == 'POST':
        form = BotForm(request.POST)
        if form.is_valid():
            bot = form.save(commit=False)
            bot.user = request.user
            bot.is_active = True # Set bot
aktif secara default
```

Integrasi Bot dengan Webhook

```
# Tentukan webhook URL sesuai dengan platform
if bot.bot_type == 'telegram':
    bot.webhook_url = f'{WEBHOOK_URL}/telegram/'
elif bot.bot_type == 'whatsapp':
    bot.webhook_url = f'{WEBHOOK_URL}/whatsapp/'

bot.save()

# Set webhook berdasarkan tipe bot
if bot.bot_type == 'telegram':
    set_telegram_webhook(bot.token_telegram,
bot.webhook_url)
    # Tambahkan pesan sukses untuk bot Telegram
    messages.success(request, 'Bot Telegram berhasil dibuat!')
elif bot.bot_type == 'whatsapp':
    set_whatsapp_webhook(bot.webhook_url)
    # Tambahkan pesan sukses untuk bot WhatsApp
    messages.success(request, f'Bot WhatsApp berhasil dibuat. Silakan atur URL webhook berikut di dashboard Twilio: {bot.webhook_url}')

    return redirect('manage_bots')
else:
    form = BotForm()

    return render(request, 'bots/create_bot.html', {'form': form})

@login_required # Pastikan hanya pengguna yang login yang bisa mengakses view ini
```

Integrasi Bot dengan Webhook

```
def manage_bots(request):
    bots = Bot.objects.filter(user=request.user)
    # Ambil daftar bot milik pengguna yang sedang login

    if request.method == 'POST':
        if 'delete' in request.POST: # Cek
            apakah ada permintaan untuk menghapus bot
            bot_id = request.POST.get('delete')
            # Ambil ID bot yang ingin dihapus dari form
            bot = get_object_or_404(Bot,
            id=bot_id) # Cari bot berdasarkan ID, atau
            tampilkan 404 jika tidak ditemukan
            bot.delete() # Hapus bot dari
            database
            messages.success(request, 'Bot
            berhasil dihapus.') # Tambahkan pesan setelah
            bot dihapus
            return redirect('manage_bots') #
            Setelah bot dihapus, kembali ke halaman
            manajemen bot

    return render(request,
    'bots/manage_bots.html', {'bots': bots}) #
    Render halaman dengan daftar bot yang dimiliki
    pengguna
```

Penjelasan Kode yang Diperbaiki

1. **Import yang Diperlukan:** Kode ini mengimpor `set_telegram_webhook` dan `set_whatsapp_webhook` dari modul `apps.webhook.views`. Ini adalah fungsi yang akan menangani pengaturan webhook sesuai dengan platform yang dipilih.

Integrasi Bot dengan Webhook

2. **Menetapkan URL Webhook:** Dalam kode yang diperbaiki, kita menetapkan `bot.webhook_url` menggunakan konfigurasi dari `settings.py`. Ini memastikan bahwa setiap bot yang dibuat memiliki URL webhook yang benar dan terintegrasi.
3. **Integrasi Webhook:** Setelah menyimpan bot, kita mengecek tipe bot yang dipilih. Jika bot adalah tipe Telegram, maka kita memanggil fungsi `set_telegram_webhook` dengan token dan URL webhook. Sebaliknya, jika bot adalah tipe WhatsApp, kita mengambil data dari formulir (Account SID, Auth Token, dan nomor WhatsApp) dan memanggil `set_whatsapp_webhook`.
4. **Alihkan ke Halaman Manajemen Bot:** Setelah proses penyimpanan dan integrasi webhook berhasil, pengguna akan diarahkan kembali ke halaman manajemen bot dengan menggunakan `redirect('manage_bots')`.

Menambahkan WEBHOOK_URL

Tambahkan URL webhook ke dalam `settings.py`. Ini akan digunakan dalam kode untuk mengonfigurasi endpoint webhook dengan platform bot:

```
# File: platform_bot/settings.py

WEBHOOK_URL = 'https://ngrok-random_id.app' #
Ganti dengan URL ngrok yang didapat
```

Dengan perbaikan ini, tampilan pembuatan bot kini telah terhubung secara efektif dengan sistem webhook, memungkinkan in-

Integrasi Bot dengan Webhook

teraksi yang lebih baik antara pengguna dan bot yang telah dibuat.

11.3.2 Memperbaiki Halaman Manage Bot

Berikut modifikasi yang sesuai untuk `manage_bots.html`, di mana kita akan menampilkan pesan sukses untuk bot yang sudah di buat.

```
<!-- File: apps/templates/bots/manage_bots.html -->

{% extends 'dashboard/base.html' %}

{% block content %}
    <div class="container">
        <h1>Manage Bots</h1>

        <!-- Tampilkan pesan sukses jika ada -->
        {% if messages %}
            <div class="alert alert-success alert-dismissible fade show" role="alert">
                {% for message in messages %}
                    {{ message }}
                {% endfor %}
                <button type="button" class="btn-close" data-bs-dismiss="alert" aria-label="Close"></button>
            </div>
        {% endif %}

        <table class="table table-striped">
            <thead>
                <tr>
                    <th>ID</th>
                    <th>Name</th>
                    <th>Platform</th>
                </tr>
            </thead>
        </table>
    </div>
{% endblock %}
```

Integrasi Bot dengan Webhook

```

        <th>Token</th>
        <th>Status</th>
        <th>Actions</th>
    </tr>
</thead>
<tbody>
    {% for bot in bots %}
        <tr>
            <td>{{ bot.id }}</td>
            <td>{{ bot.name }}</td>
            <td>{{ bot.bot_type }}</td>
            <td>{{ bot.token_telegram }}</td>
            <td>{{ bot.is_active|
yesno:"Active,Inactive" }}</td>
            <td>
                <a href="" class="btn btn-
primary">Edit</a>
                <form method="post"
style="display:inline;">
                    {% csrf_token %}
                    <button type="submit"
name="delete" value="{{ bot.id }}"
onclick="return confirm('Are you sure you want
to delete this bot?');" class="btn btn-
danger">Delete</button>
                </form>
            </td>
        </tr>
    {% empty %}
        <tr>
            <td colspan="6">No bots
available.</td>
        </tr>
    {% endfor %}
</tbody>
</table>

    <a href="{% url 'create_bot' %}" class="btn
btn-primary">Buat Bot Baru</a>

```


Integrasi Bot dengan Webhook

```
</div>
{% endblock %}
```

Penjelasan:

1. **Pesan Sukses:** Pesan sukses (`messages.success`) akan ditampilkan setelah bot berhasil dibuat (baik Telegram maupun WhatsApp). Pesan ini akan muncul di bagian atas halaman `manage_bots.html`.
2. **Menghapus Bot:** Pesan sukses juga ditampilkan ketika bot berhasil dihapus, menggunakan `messages.success`.
3. **Alert Bootstrap:** Pesan sukses menggunakan komponen `alert` dari Bootstrap dengan fitur `dismissible` agar pengguna dapat menutup pesan.

Dengan implementasi ini, pengguna akan diberi notifikasi setiap kali mereka berhasil membuat atau menghapus bot, baik itu untuk Telegram atau WhatsApp.

11.4 Pengujian Membuat Bot

Pengujian adalah bagian penting dari integrasi webhook dan bot untuk memastikan bahwa semua fungsi berjalan sesuai dengan harapan. Berikut ini adalah panduan langkah demi langkah untuk menguji integrasi bot dengan webhook, mulai dari menjalankan server Django hingga menggunakan bot di platform seperti Telegram.

Integrasi Bot dengan Webhook

11.4.1 Pengujian Membuat Bot Telegram

Menguji pembuatan bot Telegram melibatkan beberapa langkah penting, mulai dari mendapatkan token bot hingga mengirim pesan untuk memverifikasi bahwa integrasi telah berhasil. Berikut adalah langkah-langkah yang perlu diikuti untuk menguji bot Telegram yang baru dibuat.

Mendapatkan Token Bot di Telegram

Langkah pertama adalah mendapatkan token untuk bot Telegram. Buka aplikasi Telegram dan cari pengguna bernama **BotFather**. BotFather adalah bot resmi yang digunakan untuk membuat bot baru. Kirimkan perintah `/newbot` kepada BotFather. BotFather akan meminta kita untuk memberikan nama dan username untuk bot yang akan dibuat. Setelah berhasil, BotFather akan memberikan token API yang perlu disimpan, karena kita akan menggunakannya dalam aplikasi Django.

Menjalankan Ngrok dan Konfigurasi Setting

Untuk menguji webhook secara lokal, kita perlu menggunakan Ngrok. Ngrok memungkinkan kita untuk membuat tunnel dari localhost ke URL yang dapat diakses secara publik. Setelah menginstal Ngrok, jalankan perintah berikut di terminal:

```
$ ngrok http 8000
```

Perintah ini akan menampilkan URL publik yang diarahkan ke server lokal kita. Catat URL ini, karena kita akan menggunakannya untuk mengonfigurasi webhook bot Telegram.

Integrasi Bot dengan Webhook

Selanjutnya, kita perlu mengatur variabel lingkungan di `settings.py` untuk menyertakan URL Ngrok sebagai webhook. Pastikan `WEBHOOK_URL` diatur seperti berikut:

```
# File: settings.py
WEBHOOK_URL = 'https://<your-ngrok-url>' #
Ganti dengan URL Ngrok yang ditampilkan
```

Menjalankan Django Server

Setelah pengaturan dilakukan, kita perlu menjalankan server Django. Di terminal, navigasikan ke direktori proyek dan jalankan perintah berikut:

```
$ python manage.py runserver
```

Ini akan memulai server Django kita di `localhost:8000`, yang memungkinkan kita untuk mengakses aplikasi melalui browser.

Mengakses Halaman Create dan Memasukkan Token

Dengan server yang berjalan, buka browser dan akses halaman pembuatan bot di

`http://localhost:8000/create_bot/`. Isi form dengan nama bot dan token yang telah didapatkan dari BotFather. Pastikan juga memilih jenis bot sebagai Telegram sebelum mengirimkan formulir.

Mencoba Mengirim Pesan di Telegram

Integrasi Bot dengan Webhook

Setelah bot berhasil dibuat, kita dapat mencoba mengirim pesan ke bot di Telegram. Cari bot yang baru saja dibuat di aplikasi Telegram dengan nama yang telah ditentukan. Kirimkan pesan misalnya `/start` dan `/help`.

Respon di Ngrok Jika Berhasil

Setelah mengirim pesan, kembali ke terminal yang menjalankan Ngrok. Ngrok akan menampilkan log dari permintaan yang diterima. Jika integrasi berhasil, kita seharusnya dapat melihat log yang menunjukkan bahwa bot menerima pesan dan melakukan respon sesuai dengan pengaturan webhook.

```
Connections          ttl  opn  rt1  rt5  p50  p90
1      0.00  0.00  0.00  0.00

HTTP Requests
-----

23:25:18.785 WIB POST /getpost/          200 OK
```

Jika semua langkah diikuti dengan benar, kita akan mendapatkan konfirmasi bahwa bot Telegram kita telah berhasil terintegrasi dan berfungsi dengan baik.

Dengan demikian, pengujian pembuatan bot Telegram telah berhasil dilakukan.

11.4.2 Pengujian Membuat Bot Whatsapp

Pada bagian ini, kita akan fokus pada pengujian pembuatan bot WhatsApp menggunakan Twilio. Integrasi ini melibatkan beberapa langkah penting, dari konfigurasi webhook hingga menjalankan bot secara lokal menggunakan server Django. Langkah-langkah ini penting untuk memastikan bahwa bot dapat berfungsi dengan baik dalam menerima dan merespons pesan dari WhatsApp.

Mengonfigurasi Webhook di Twilio

Salah satu langkah penting dalam pembuatan bot WhatsApp adalah mengonfigurasi URL webhook di Twilio agar bot dapat menerima dan merespons pesan. Webhook adalah mekanisme yang memungkinkan aplikasi seperti Twilio untuk mengirim data ke server yang kita tentukan setiap kali ada aktivitas terkait, seperti penerimaan pesan.

1. **Akses Twilio Console:** Setelah Anda memiliki akun Twilio, masuk ke **Twilio Console**. Di sini, Anda akan menemukan opsi untuk mengelola proyek WhatsApp.
2. **Konfigurasi Webhook:** Di dalam Twilio Console, buka proyek yang telah Anda buat untuk bot WhatsApp. Cari menu **Messaging Services** atau **WhatsApp**, dan Anda akan menemukan opsi untuk menambahkan webhook. Webhook ini adalah URL yang akan menerima pesan dari pengguna WhatsApp dan kemudian memprosesnya di aplikasi Django.
 - Pada bagian **Webhook for incoming messages**, masukkan URL webhook yang akan digunakan. Misalnya, jika Anda menggunakan Ngrok untuk

Integrasi Bot dengan Webhook

menjalankan server lokal dan mengekspose URL publik, masukkan URL berikut:

```
https://ngrok.random.id.app/  
whatsapp/
```

- Pastikan untuk menyesuaikan URL ini dengan yang dihasilkan oleh Ngrok atau domain Anda jika aplikasi sudah dideploy.

3. *Jalankan Ngrok untuk Menghubungkan Webhook:*

Karena server Django biasanya berjalan di localhost, Anda perlu menggunakan **Ngrok** untuk membuka akses publik ke aplikasi lokal Anda. Jalankan Ngrok dengan perintah:

```
$ ngrok http 8000
```

Setelah dijalankan, Ngrok akan memberikan URL publik (misalnya, `https://random.id.ngrok.io`). Gunakan URL ini sebagai webhook di Twilio, ditambah dengan path `/whatsapp/`, seperti contoh di atas.

Menjalankan Django Server dan Menguji Bot

Setelah webhook dikonfigurasi, jalankan server Django untuk memastikan bot dapat menerima dan memproses pesan WhatsApp.

1. Jalankan server Django menggunakan perintah:

```
$ python manage.py runserver
```

Integrasi Bot dengan Webhook

2. Akses halaman pembuatan bot WhatsApp melalui browser di URL berikut:

```
http://localhost:8000/create_bot/
```

Isi formulir dengan informasi yang diperlukan untuk membuat bot WhatsApp. Namun, karena kredensial seperti Account SID dan Auth Token tidak diperlukan dalam sistem ini, Anda hanya perlu memastikan bahwa URL webhook telah dikonfigurasi dengan benar di Twilio.

Mengirim Pesan Uji ke Bot WhatsApp

Setelah bot berhasil dibuat dan server Django berjalan, uji bot dengan mengirimkan pesan ke nomor WhatsApp yang telah Anda daftarkan di Twilio.

1. Buka aplikasi WhatsApp di ponsel Anda.
2. Kirim pesan ke nomor WhatsApp yang terhubung dengan Twilio.
3. Periksa log di terminal yang menjalankan Ngrok untuk melihat apakah bot menerima dan memproses pesan tersebut.

Jika pengaturan webhook dan server sudah benar, Anda akan melihat log di terminal Ngrok yang menunjukkan bahwa pesan berhasil diterima dan diproses oleh bot.

Dengan langkah-langkah ini, Anda dapat menguji bot WhatsApp yang terintegrasi dengan Twilio dan memastikan bahwa webhook

Integrasi Bot dengan Webhook

berjalan dengan benar. Pastikan untuk selalu menggunakan URL Ngrok yang aktif selama pengembangan lokal, dan lakukan pengujian menyeluruh untuk memastikan bot WhatsApp berfungsi sebagaimana mestinya.

Dengan mengikuti langkah-langkah di atas, Anda akan dapat mengintegrasikan bot dengan webhook di platform seperti Telegram dan WhatsApp serta menguji bahwa semuanya berfungsi dengan baik. Pastikan untuk selalu melakukan pengujian menyeluruh di setiap platform dan memperbaiki setiap error yang muncul melalui log atau konsol Django.

Kesimpulan Bab

Pada Bagian ini, kita fokus pada proses ***Integrasi Bot dengan Webhook***, yang merupakan elemen krusial dalam menghubungkan bot dengan platform seperti Telegram dan WhatsApp. Dimulai dengan ***Pengantar Integrasi Webhook dengan Bot***, kita memahami bagaimana webhook bertindak sebagai jembatan komunikasi antara bot dan platform eksternal, serta pentingnya pengujian dalam memastikan bahwa webhook berfungsi dengan benar.

Dalam bagian ***Menyiapkan Integrasi Webhook untuk Bot***, kita membangun fondasi teknis untuk mengintegrasikan bot dengan webhook pada berbagai platform. Proses ini mencakup ***penyusunan struktur webhook*** agar dapat menangani update dari beberapa platform sekaligus. Pada bagian ini, kita mempelajari cara menghubungkan webhook dengan bot di Django, termasuk menyiapkan view dan handler untuk menangani update dari Telegram dan WhatsApp. Selain itu, kita juga menyusun konfigurasi yang diperlukan untuk setiap platform agar bot dapat bekerja secara optimal.

Pada bagian terakhir, ***Integrasi Webhook dan Bot***, kita menyempurnakan integrasi dengan membuat fungsi-fungsi untuk memproses update dari setiap platform. Kita memperbarui view di `apps/bots` untuk menangani berbagai jenis pesan dan update yang masuk dari platform eksternal. Bagian ini juga menekankan pentingnya pengujian, termasuk proses pembuatan, pengeditan, dan pengelolaan bot di platform, yang diujicobakan melalui server Django dan ngrok untuk memastikan bahwa webhook terhubung dan berjalan dengan baik.

Integrasi Bot dengan Webhook

Dengan menyelesaikan bab ini, kita tidak hanya mempelajari cara mengintegrasikan bot dengan webhook, tetapi juga memastikan bahwa pengujian dilakukan secara menyeluruh untuk setiap langkah proses. Bab ini memberikan pemahaman teknis yang mendalam dan keterampilan praktis dalam membangun, mengelola, dan menguji integrasi bot dengan berbagai platform melalui webhook, menjadikan bot Anda lebih responsif dan efisien dalam menangani interaksi pengguna.

BAB 12 -Membuat Fitur untuk Bot

Pada bab ini, kita akan membahas bagaimana membangun fitur-fitur penting untuk mengelola bot di platform Django, termasuk penambahan command, pengeditan bot, dan pengelolaan command yang ada. Fitur ini penting karena command pada bot berperan sebagai mekanisme utama interaksi antara pengguna dan bot. Dengan mengatur command yang tepat, bot bisa merespons perintah pengguna dengan cara yang efisien dan terstruktur.

Bab ini dimulai dengan pengenalan mengenai command pada bot serta pentingnya fitur ini dalam membangun interaksi yang kuat antara pengguna dan sistem bot. Setelah itu, kita akan membahas langkah-langkah teknis dalam menyiapkan model, form, dan view untuk menambahkan serta mengedit command. Selain itu, kita juga akan mengintegrasikan fitur-fitur tersebut dengan log aktivitas, sehingga setiap aksi penting seperti pembuatan, pengeditan, atau penghapusan command dan bot dapat dicatat dengan baik.

Melalui bab ini, kita akan mempelajari proses teknis mulai dari pembuatan model dan form hingga integrasi dengan model log aktivitas, serta bagaimana setiap komponen saling terkait untuk membangun sistem manajemen bot yang efektif dan efisien. Di akhir bab, kita juga akan melihat kode lengkap yang menyusun keseluruhan fitur-fitur ini agar bot yang dikembangkan bisa

Membuat Fitur untuk Bot

berjalan dengan baik di berbagai platform seperti Telegram dan WhatsApp.

Dengan demikian, bab ini menjadi fondasi penting dalam pengembangan bot yang fungsional dan fleksibel, siap digunakan dalam berbagai skenario interaksi dengan pengguna.

12.1 Pengantar Fitur Command Pada Bot

Saat membangun bot, salah satu fitur yang paling penting dan sering digunakan adalah **command**. Command pada bot adalah perintah yang dimasukkan oleh pengguna untuk memicu tindakan tertentu. Di platform bot seperti Telegram, command biasanya dimulai dengan tanda /, seperti `/start` atau `/help`. Fitur ini memungkinkan bot untuk merespons pengguna dengan cara yang terstruktur dan logis.

12.1.1 Apa Itu Command pada Bot?

Command adalah perintah yang diberikan pengguna kepada bot untuk melakukan fungsi tertentu. Dalam konteks aplikasi bot, seperti Telegram, command dapat digunakan untuk mengarahkan bot untuk menampilkan pesan, memulai layanan, atau menjalankan aksi lain yang telah diprogramkan. Misalnya, perintah `/start` biasanya digunakan untuk memulai interaksi awal dengan bot dan menyajikan informasi dasar kepada pengguna.

Membuat Fitur untuk Bot

Command pada bot dapat diartikan sebagai perintah yang diberikan oleh pengguna untuk mengarahkan bot melakukan tugas tertentu. Ini adalah cara utama bagi pengguna untuk berinteraksi dengan bot, dan bisa berupa teks sederhana yang diprogram untuk memicu berbagai aksi. Misalnya, dalam konteks bot Telegram, pengguna mungkin menggunakan command seperti `/start` untuk memulai interaksi atau `/help` untuk mendapatkan informasi tentang fungsi yang tersedia.

Command pada bot berfungsi seperti jalan pintas yang memungkinkan pengguna berinteraksi tanpa harus mengetikkan teks panjang. Setiap command dipetakan ke fungsi tertentu di backend bot, dan saat command tersebut diterima, bot akan mengeksekusi fungsi yang bersangkutan.

Pentingnya command dalam interaksi bot tidak bisa dianggap remeh. Dengan command, pengguna dapat mengakses fitur-fitur khusus yang disediakan oleh bot dengan mudah dan cepat. Command memberikan struktur pada komunikasi, memungkinkan pengguna untuk memahami cara berinteraksi dengan bot secara lebih intuitif. Selain itu, command juga memungkinkan pengembangan untuk menangani berbagai skenario penggunaan yang berbeda, mulai dari memberikan informasi hingga menjalankan tugas yang lebih kompleks.

Proses menambahkan dan mengedit command juga menjadi aspek yang penting dalam pengembangan bot. Pengembang harus merancang sistem yang memungkinkan command ditambahkan, diubah, atau dihapus sesuai kebutuhan. Dengan sistem ini, bot

Membuat Fitur untuk Bot

dapat terus berkembang dan menyesuaikan diri dengan umpan balik pengguna serta kebutuhan yang berubah seiring waktu.

Di bab ini, kita akan menjelajahi lebih dalam tentang bagaimana cara menambahkan fitur command pada bot kita, termasuk langkah-langkah teknis dan praktik terbaik yang perlu diperhatikan. Kami akan mulai dengan membahas model yang diperlukan untuk command, kemudian berlanjut ke pengembangan form, view, dan template yang mendukung interaksi tersebut. Dengan langkah-langkah ini, kita akan menciptakan bot yang tidak hanya responsif tetapi juga mudah untuk dikelola dan dikembangkan lebih lanjut.

12.1.2 Pentingnya Command dalam Interaksi Bot

Dalam dunia pengembangan bot, command memainkan peran yang sangat krusial dalam membentuk interaksi antara bot dan penggunanya. Command tidak hanya memberikan cara bagi pengguna untuk berkomunikasi dengan bot, tetapi juga menentukan bagaimana bot merespons dan menjalankan tugas-tugas tertentu. Dengan memahami pentingnya command, kita bisa merancang bot yang lebih efektif dan efisien.

Command berperan penting dalam menciptakan pengalaman pengguna yang interaktif dan mudah digunakan. Dengan command, pengguna bisa langsung memberi instruksi pada bot tanpa kebingungan. Sebagai contoh, sebuah bot e-commerce mungkin memiliki command `/order` untuk memulai proses pemesanan,

Membuat Fitur untuk Bot

atau `/status` untuk memeriksa status pemesanan yang sudah dilakukan.

Salah satu alasan utama mengapa command begitu penting adalah karena mereka menyediakan struktur yang jelas dalam interaksi. Pengguna tidak perlu menebak bagaimana cara menggunakan bot; mereka cukup memasukkan perintah yang telah ditentukan. Misalnya, ketika pengguna mengetik `/weather`, mereka mengharapkan bot memberikan informasi cuaca terkini. Dengan adanya command, bot bisa memberikan respons yang relevan secara cepat, meningkatkan pengalaman pengguna.

Selain itu, command juga memungkinkan pengembang untuk menangani berbagai skenario dengan lebih baik. Dengan mendefinisikan command yang berbeda, pengembang dapat mengarahkan bot untuk melakukan fungsi yang spesifik sesuai dengan kebutuhan pengguna. Ini menciptakan fleksibilitas dalam cara bot beroperasi, memungkinkan penambahan fitur baru dengan mudah tanpa perlu merombak seluruh sistem. Misalnya, pengembang dapat menambahkan command baru untuk mengakses informasi tertentu atau mengubah pengaturan bot hanya dengan menambahkan beberapa baris kode.

Command juga berfungsi sebagai alat untuk memfasilitasi komunikasi yang lebih efektif. Dalam situasi di mana informasi harus disampaikan dengan cepat, seperti dalam pengingat atau notifikasi, command dapat digunakan untuk memicu respons yang otomatis. Hal ini tidak hanya menghemat waktu, tetapi juga memas-

Membuat Fitur untuk Bot

tikan bahwa pengguna mendapatkan informasi yang mereka butuhkan dengan segera.

Dalam konteks keamanan, command dapat membantu dalam mengontrol akses ke fungsi tertentu dari bot. Dengan mendefinisikan siapa yang dapat menggunakan command tertentu, pengembang dapat melindungi data sensitif dan memastikan bahwa hanya pengguna yang berwenang yang dapat menjalankan perintah kritis.

Dengan adanya command, pengguna tidak perlu mengingat berbagai teks atau instruksi yang rumit; mereka cukup mengetik command yang sudah didefinisikan, dan bot akan langsung memprosesnya. Hal ini meningkatkan efisiensi interaksi antara pengguna dan bot, serta membantu memastikan bahwa interaksi berjalan sesuai skenario yang sudah dirancang.

Dengan memahami dan menerapkan command secara efektif, kita dapat meningkatkan interaksi pengguna dan memastikan bahwa bot berfungsi dengan baik, memenuhi ekspektasi pengguna, dan memberikan pengalaman yang memuaskan. Dalam bab selanjutnya, kita akan membahas lebih lanjut tentang bagaimana cara menambahkan dan mengedit command dalam bot kita, agar dapat berfungsi sesuai dengan tujuan yang diinginkan.

12.1.3 Proses Menambahkan dan Mengedit Command

Menambahkan dan mengedit command pada bot adalah proses penting yang memungkinkan pengembang untuk mengoptimal-

Membuat Fitur untuk Bot

kan interaksi pengguna. Proses ini terdiri dari beberapa langkah, mulai dari mendefinisikan command, mengimplementasikannya dalam kode, hingga memastikan bahwa command tersebut dapat diedit sesuai kebutuhan.

Langkah pertama dalam menambahkan command adalah mendefinisikannya. Ini meliputi menentukan nama command, fungsionalitas yang diinginkan, dan bagaimana bot akan merespons command tersebut. Misalnya, jika kita ingin menambahkan command untuk mendapatkan informasi cuaca, kita perlu memutuskan nama command, seperti `/weather`, dan menentukan parameter apa saja yang mungkin diperlukan, seperti lokasi atau jenis informasi cuaca yang diminta.

Setelah command didefinisikan, langkah berikutnya adalah mengimplementasikannya dalam kode. Di sini, kita akan membuat fungsi yang menangani command tersebut. Misalnya, kita bisa menggunakan Django untuk membuat view yang merespons ketika command diterima. Kode berikut adalah contoh sederhana untuk menangani command cuaca:

```
# File: apps/bots/views.py
from django.http import JsonResponse
import requests

def weather_command(request):
    location = request.GET.get('location',
    'default_location')
    # Mengambil informasi cuaca dari API
    response =
requests.get(f'https://api.weatherapi.com/v1/current.json?key=YOUR_API_KEY&q={location}')
```

Membuat Fitur untuk Bot

```
weather_data = response.json()

return JsonResponse(weather_data) #
Mengembalikan data cuaca dalam format JSON
```

Dalam kode ini, kita membuat fungsi `weather_command` yang menerima lokasi sebagai parameter, mengakses API cuaca, dan mengembalikan data cuaca dalam format JSON. Dengan cara ini, kita memastikan bahwa command yang ditambahkan dapat memberikan informasi yang relevan dan bermanfaat bagi pengguna.

Setelah command ditambahkan, penting untuk memastikan bahwa kita juga menyediakan cara untuk mengedit command tersebut di masa depan. Proses ini biasanya melibatkan pembuatan antarmuka pengguna (UI) yang memungkinkan pengguna untuk melihat, menambah, atau mengedit command yang sudah ada. Ini bisa dilakukan dengan menggunakan form di Django, di mana pengguna dapat menginput perubahan yang diinginkan dan menyimpannya ke dalam database.

Contoh sederhana dari form untuk mengedit command bisa terlihat seperti berikut:

```
# File: apps/bots/forms.py
from django import forms
from .models import Command

class CommandForm(forms.ModelForm):
    class Meta:
        model = Command
```

Membuat Fitur untuk Bot

```
fields = ['name', 'description'] #  
Misalnya, hanya nama dan deskripsi yang bisa  
diedit
```

Di sini, kita menggunakan Django ModelForm untuk membuat form yang memungkinkan pengguna untuk mengedit nama dan deskripsi command. Setelah form ini siap, kita perlu menyiapkan view yang dapat menangani input dari form dan menyimpan perubahan ke dalam database.

Dengan memahami proses menambahkan dan mengedit command ini, kita tidak hanya memperluas fungsionalitas bot, tetapi juga memberikan fleksibilitas bagi pengembang untuk menyesuaikan bot sesuai kebutuhan pengguna. Pada bagian selanjutnya, kita akan membahas lebih lanjut tentang cara menyiapkan fitur untuk mengedit bot dan command secara lebih mendetail.

Setiap langkah ini akan dibahas secara detail dalam bab selanjutnya, mulai dari persiapan model hingga pengujian fitur command pada bot. Hal ini memastikan bahwa setiap command yang dibuat berfungsi dengan benar dan sesuai dengan logika yang diinginkan.

12.2 Menyiapkan Model Dan Forms

Dalam pengembangan bot, model adalah fondasi yang sangat penting. Model mendefinisikan struktur data yang akan kita gunakan, termasuk command dan respons yang akan diproses oleh

Membuat Fitur untuk Bot

bot. Pada sub bab ini, kita akan membuat model dan forms untuk command yang akan digunakan oleh bot.

12.2.1 Membuat Model untuk Command

Kita akan menggunakan Django ORM (Object-Relational Mapping) untuk membuat model `BotCommand`. Model ini akan menyimpan informasi mengenai command yang terkait dengan bot tertentu, termasuk nama command dan respons yang harus diberikan ketika command tersebut diterima.

Berikut adalah implementasi model tersebut:

```
# File: apps/bots/models.py

from django.db import models

class BotCommand(models.Model):
    bot = models.ForeignKey(Bot,
        related_name='commands',
        on_delete=models.CASCADE)
    command = models.CharField(max_length=255,
        help_text="Masukkan perintah seperti '/start',
        'hello', dll.")
    response =
models.TextField(help_text="Masukkan respons
yang akan dikirim saat perintah diterima.")

    def __str__(self):
        return f"Command: {self.command} ->
Response: {self.response}"

    class Meta:
        unique_together = ['bot', 'command'] #
Pastikan setiap bot hanya memiliki satu perintah
dengan nama yang sama.
```

Membuat Fitur untuk Bot

```
verbose_name = "Bot Command"  
verbose_name_plural = "Bot Commands"
```

Mari kita bahas lebih dalam mengenai model ini. Pertama, kita memiliki relasi `ForeignKey` yang menghubungkan setiap `command` dengan bot tertentu. Ini berarti setiap `command` harus terasosiasi dengan satu bot, dan kita menggunakan `related_name='commands'` untuk memudahkan akses dari objek bot ke `command-command` yang terkait.

Field `command` menggunakan tipe data `CharField` dengan panjang maksimal 255 karakter. Ini akan digunakan untuk menyimpan nama `command`, misalnya `/start` atau `hello`. Kita juga menyediakan `help_text` yang memberikan petunjuk tentang penggunaan field tersebut, sehingga pengembang atau pengguna lain yang berinteraksi dengan form dapat lebih mudah memahami cara mengisi data.

Field `response` menggunakan tipe data `TextField` untuk menyimpan respons yang akan dikirimkan saat `command` diterima. Ini memungkinkan kita untuk menyimpan pesan yang lebih panjang dan lebih informatif yang dapat disampaikan kepada pengguna.

Metode `__str__` didefinisikan untuk memberikan representasi string yang informatif dari objek model. Ketika kita mencetak objek `BotCommand`, kita akan melihat nama `command` dan respons yang terkait, yang sangat berguna saat debugging atau saat menampilkan data di admin panel.

Membuat Fitur untuk Bot

Pada bagian *Meta*, kita mendefinisikan dua hal penting. Pertama, kita menetapkan `unique_together`, yang memastikan bahwa setiap bot hanya dapat memiliki satu command dengan nama yang sama. Ini mencegah duplikasi command yang tidak diinginkan dan menjaga konsistensi data. Kedua, kita menggunakan `verbose_name` dan `verbose_name_plural` untuk memberikan nama yang lebih baik dan lebih deskriptif di admin panel Django.

Model ini menyediakan dasar bagi kita untuk menyimpan data command di database. Setelah model selesai dibuat, jangan lupa untuk membuat dan menjalankan migrasi agar tabel `BotCommand` dapat dibuat di database:

```
$ python manage.py makemigrations
$ python manage.py migrate
```

Dengan demikian, kita telah menyelesaikan langkah pertama dalam menyiapkan fitur command pada bot, yakni dengan membuat model yang akan menyimpan command dan respons yang sesuai. Pada sub bab berikutnya, kita akan melanjutkan dengan membuat *view* dan antarmuka untuk menambahkan command baru melalui panel admin atau halaman khusus.

Dengan model ini, kita sudah siap untuk menyimpan command yang terkait dengan bot dalam database, dan langkah selanjutnya adalah membuat form yang memungkinkan pengguna untuk menambah atau mengedit command tersebut. Pada bagian berikutnya, kita akan membahas lebih lanjut tentang proses tersebut.

12.2.2 Membuat forms untuk Command

Setelah kita mendefinisikan model `BotCommand`, langkah selanjutnya adalah membuat form yang akan digunakan untuk menambah dan mengedit command. Form ini akan menyediakan antarmuka bagi pengguna untuk memasukkan data yang diperlukan, seperti nama command dan respons yang akan dikirim oleh bot. Kita akan menggunakan Django's `ModelForm` untuk membuat form ini, sehingga semua validasi dan pengelolaan data akan dilakukan secara otomatis. Form ini akan digunakan untuk mengambil input dari pengguna (admin atau user yang berhak) dan kemudian menyimpan data tersebut ke dalam database menggunakan model `BotCommand`.

Dalam aplikasi Django, pembuatan form dapat dilakukan dengan mudah menggunakan `ModelForm`, yang secara otomatis akan menghasilkan form berdasarkan model yang sudah kita buat sebelumnya. Di sini, kita akan membuat form untuk menambah command dan respons.

Form `BotCommandForm` memungkinkan pengguna untuk memasukkan perintah dan responsnya secara sederhana, sekaligus memberikan antarmuka yang mudah digunakan dengan placeholder dan teks bantuan (*help text*).

Berikut adalah implementasi form `BotCommandForm`:

```
# File: apps/bots/forms.py
from .models import BotCommand
```

Membuat Fitur untuk Bot

```
class BotCommandForm(forms.ModelForm):
    class Meta:
        model = BotCommand
        fields = ['command', 'response'] #
        # Field yang akan ditampilkan dalam form
        widgets = {
            'command':
forms.TextInput(attrs={'placeholder':
'Contoh: /start, /help'}),
            'response':
forms.Textarea(attrs={'rows': 3, 'placeholder':
'Masukkan respon yang akan dikirim.'}),
        }
        help_texts = {
            'command': 'Masukkan perintah
seperti "/start", "/help", atau lainnya.',
            'response': 'Masukkan respons yang
akan dikirim saat perintah diterima.',
        }
```

Mari kita uraikan bagian-bagian dari form ini. Pertama, kita mengimpor `forms` dari Django dan model `BotCommand` yang telah kita buat sebelumnya. Kemudian, kita mendefinisikan `BotCommandForm` sebagai subclass dari `forms.ModelForm`, yang memungkinkan kita untuk secara otomatis mengaitkan form ini dengan model yang telah ada.

Di dalam kelas `Meta`, kita menetapkan model yang akan digunakan, yaitu `BotCommand`, dan menentukan field yang akan ditampilkan dalam form. Dalam hal ini, kita memilih `command` dan `response`. Ini memastikan bahwa form hanya akan meminta informasi yang relevan kepada pengguna.

Membuat Fitur untuk Bot

Selanjutnya, kita menggunakan `widgets` untuk mengonfigurasi tampilan form. Untuk field `command`, kita menggunakan `TextInput` dengan atribut `placeholder` yang memberikan contoh perintah yang dapat dimasukkan. Hal ini membantu pengguna memahami format yang diharapkan. Untuk field `response`, kita menggunakan `Textarea` dengan jumlah baris yang ditentukan, serta `placeholder` untuk memberikan instruksi lebih lanjut mengenai isi respons.

Selain itu, kita juga menyediakan `help_texts`, yang berfungsi untuk memberikan petunjuk lebih jelas mengenai setiap field. Ini akan muncul di bawah setiap input dalam form, memberikan panduan langsung kepada pengguna tentang apa yang harus mereka masukkan.

Dengan form ini siap, pengguna dapat dengan mudah menambahkan `command` baru atau mengedit yang sudah ada melalui antarmuka yang lebih intuitif. Selanjutnya, kita akan membahas bagaimana cara membuat view yang akan menangani logika penyimpanan dan pengeditan `command` melalui form ini.

Setelah membuat form, langkah selanjutnya adalah mengintegrasikannya ke dalam view. Form ini nantinya akan digunakan untuk menampilkan halaman penambahan dan pengeditan `command`.

Pada bagian selanjutnya, kita akan membahas cara mengintegrasikan form ini ke dalam view dan memastikan bahwa input dari

Membuat Fitur untuk Bot

pengguna bisa diproses dan disimpan dengan benar ke dalam database.

12.3 Menyiapkan Fitur Untuk Edit Bot

Di bagian ini kita akan menyiapkan untuk fitur edit bot, mulai dari menyiapkan views, template edit bot, dan url routing untuk edit bot. Di halaman edit bot ini user bisa mengedit bot yang telah mereka buat.

12.3.1 Membuat View untuk Edit Bot

Setelah kita menyiapkan model dan form yang diperlukan, langkah selanjutnya adalah membuat view untuk mengelola proses pengeditan bot. View ini bertanggung jawab untuk mengambil data bot yang ingin diedit, memuat form dengan data yang sudah ada, serta memproses penyimpanan data setelah pengguna melakukan perubahan.

Mari kita mulai dengan mendefinisikan view untuk mengedit bot. Berikut adalah contoh implementasi dari view `edit_bot`:

```
# File: apps/bots/views.py

from .forms import BotForm # Pastikan BotForm
telah didefinisikan di forms.py

def edit_bot(request, bot_id):
    bot = get_object_or_404(Bot, id=bot_id) #
    Ambil bot berdasarkan ID
    if request.method == 'POST':
```

Membuat Fitur untuk Bot

```
        form = BotForm(request.POST,
instance=bot) # Muat data ke form
        if form.is_valid():
            form.save() # Simpan data yang
telah diedit
            return redirect('manage_bots') #
Alihkan ke halaman pengelolaan bot
        else:
            form = BotForm(instance=bot) # Jika
bukan POST, buat form dengan data bot

        context = {
            'form': form,
            'bot': bot,
        }
        return render(request, 'bots/edit_bot.html',
context) # Tampilkan template dengan context
```

Di sini, kita mulai dengan mengimpor fungsi-fungsi yang diperlukan dari Django, termasuk `render`, `get_object_or_404`, dan `redirect`. Kemudian, kita mengambil model `Bot` dan `BotForm` yang telah kita definisikan sebelumnya.

Fungsi `edit_bot` menerima `request` dan `bot_id` sebagai parameter. Pertama, kita mengambil objek `bot` yang sesuai dengan `bot_id` menggunakan `get_object_or_404`. Jika `bot` tidak ditemukan, pengguna akan diarahkan ke halaman kesalahan 404.

Selanjutnya, kita memeriksa apakah metode permintaan adalah `POST`. Jika ya, ini berarti pengguna telah mengirimkan form. Kita kemudian membuat instance `BotForm` dengan data yang diterima dan mengaitkannya dengan objek `bot` yang ada. Jika form

Membuat Fitur untuk Bot

valid, kita simpan perubahan dan mengalihkan pengguna ke halaman pengelolaan bot menggunakan `redirect`.

Jika metode permintaan bukan POST, kita membuat instance `BotForm` dengan data dari objek bot yang ada. Ini akan mengisi form dengan informasi yang ada, memungkinkan pengguna untuk melihat dan mengedit detail bot.

Terakhir, kita menyiapkan konteks yang berisi form dan objek bot untuk diteruskan ke template `edit_bot.html`. Template ini akan digunakan untuk menampilkan form kepada pengguna, memungkinkan mereka untuk mengedit informasi bot yang diinginkan.

Dengan view ini siap, pengguna sekarang dapat mengedit bot yang sudah ada dengan mudah. Pada tahap berikutnya, kita akan membahas bagaimana cara membuat template untuk menampilkan form edit ini.

12.3.2 Membuat Template untuk Edit Bot

Setelah kita menyiapkan view untuk mengedit bot, langkah berikutnya adalah membuat template yang akan menampilkan form edit kepada pengguna. Template ini akan menyediakan antarmuka yang intuitif untuk mengubah informasi bot, sehingga pengguna dapat dengan mudah melakukan modifikasi sesuai kebutuhan.

Mari kita buat template `edit_bot.html`. Berikut adalah contohnya:

```
<!-- File: apps/templates/bots/edit_bot.html -->
```

Membuat Fitur untuk Bot

```
{% extends 'dashboard/base.html' %}

{% block content %}
<div class="container mt-4">
  <div class="card">
    <div class="card-header">
      <h5 class="card-title">Edit Bot:
    {{ bot.name }}</h5>
    </div>
    <div class="card-body">
      <form method="post">
        {% csrf_token %}
        {{ form.as_p }} <!--
Menampilkan semua field form sebagai paragraf --
-->

        <button type="submit" class="btn
btn-primary">
          Simpan Perubahan
        </button>
        <a href="{% url 'manage_bots'
%}" class="btn btn-secondary">Kembali</a>
      </form>
    </div>
  </div>
</div>
{% endblock %}
```

Dalam template ini, kita mulai dengan mewarisi dari template dasar `dashboard/base.html`, yang memberikan struktur umum pada halaman. Kita kemudian membuka blok konten untuk mengisi bagian yang relevan.

Di dalam blok konten, kita membuat sebuah kontainer Bootstrap untuk menampung card yang berisi form edit bot. Judul card

Membuat Fitur untuk Bot

mencantumkan nama bot yang sedang diedit, memberikan konteks yang jelas kepada pengguna.

Form ini menggunakan metode POST untuk mengirim data kembali ke server. Kita menambahkan token CSRF untuk keamanan, yang diperlukan oleh Django untuk melindungi terhadap serangan CSRF.

Untuk menampilkan field form, kita menggunakan `{{ form.as_p }}` yang secara otomatis menghasilkan HTML untuk semua field dalam form, ditampilkan sebagai paragraf. Ini membuatnya lebih sederhana dan menjaga konsistensi gaya.

Setelah field form, kita menyediakan dua tombol: satu untuk menyimpan perubahan dan yang lainnya untuk kembali ke halaman pengelolaan bot. Tombol simpan menggunakan kelas Bootstrap `btn btn-primary` untuk memberikan tampilan yang menonjol, sedangkan tombol kembali menggunakan kelas `btn btn-secondary` untuk menandakan tindakan alternatif.

Dengan template ini, pengguna akan memiliki antarmuka yang bersih dan fungsional untuk mengedit informasi bot. Selanjutnya, kita akan membahas bagaimana mengatur URL routing untuk menghubungkan view edit bot ini.

12.3.3 Mengatur URL Routing untuk Fitur Edit Bot

Setelah kita menyiapkan view dan template untuk fitur edit bot, langkah selanjutnya adalah mengatur URL routing yang akan menghubungkan permintaan pengguna dengan view yang tepat. Pengaturan ini penting untuk memastikan bahwa pengguna dapat mengakses halaman edit bot dengan menggunakan URL yang sesuai.

Untuk mengatur URL routing, kita akan menambahkan path baru di dalam file `urls.py` yang terdapat pada aplikasi bot. Berikut adalah contoh kode yang bisa kita gunakan:

```
# File: apps/bots/urls.py
from django.urls import path
from . import views

urlpatterns = [
    # URL untuk halaman edit bot
    path('edit_bot/<int:bot_id>/',
        views.edit_bot, name='edit_bot'), # Menggunakan
        bot_id sebagai parameter
]
```

Dalam kode di atas, kita menggunakan fungsi `path()` dari Django untuk mendefinisikan URL baru. URL ini dirancang untuk menangani permintaan ke halaman edit bot. Kita menambahkan parameter `<int:bot_id>` dalam URL, yang berarti kita akan menerima ID bot sebagai integer. Parameter ini akan diteruskan ke view `edit_bot` yang telah kita buat sebelumnya, memungkinkan kita untuk mengidentifikasi bot yang ingin diedit.

Membuat Fitur untuk Bot

Dengan memberikan nama pada URL ini, yaitu `'edit_bot'`, kita juga mempermudah kita dalam merujuk ke URL ini di template dan kode lainnya. Ini sangat berguna ketika kita ingin membuat tautan ke halaman edit bot, seperti pada tombol kembali yang kita tambahkan sebelumnya.

Setelah menambahkan path ini, pastikan untuk melakukan pengujian untuk memastikan bahwa routing bekerja dengan baik. Dengan langkah ini, kita telah menghubungkan antarmuka pengguna untuk mengedit bot dengan fungsi yang relevan dalam aplikasi kita.

Dengan URL routing yang sudah diatur, kita siap melanjutkan ke langkah berikutnya, yaitu menyiapkan fitur untuk command bot.

12.4 Menyiapkan Fitur Untuk Tambah Command Bot

Di bagian ini kita akan menyiapkan untuk fitur tambah command, mulai dari menyiapkan views, template add command, dan url routing untuk edit bot. Di halaman add command ini user bisa membuat perintah atau command dan respon untuk bot mereka..

12.4.1 Membuat View untuk Menambah Command

Setelah berhasil membuat form untuk command, langkah selanjutnya adalah membuat view yang akan memproses form terse-

Membuat Fitur untuk Bot

but. View ini bertanggung jawab untuk menampilkan halaman penambahan command baru, menerima input dari pengguna, dan menyimpan data ke dalam database.

Pada tahap ini, kita akan membuat sebuah view berbasis fungsi (function-based view) yang menangani proses penambahan command untuk bot yang sudah ada. Pengguna yang ingin menambah command harus login terlebih dahulu, dan hanya pemilik bot yang bisa menambah command untuk bot mereka.

Berikut adalah kode lengkap untuk view `add_command`:

```
# File: apps/bots/views.py

from .models import BotCommand
from .forms import BotCommandForm

@login_required
def add_command(request, bot_id):
    # Ambil bot berdasarkan ID, pastikan bot
    # milik user yang sedang login
    bot = get_object_or_404(Bot, id=bot_id,
        user=request.user)

    # Jika request method adalah POST, proses
    # form
    if request.method == 'POST':
        form = BotCommandForm(request.POST)
        if form.is_valid():
            command = form.save(commit=False) #
            # Belum simpan ke database, tambahkan bot terlebih
            # dahulu
            command.bot = bot # Assign bot ke
            # command
```

Membuat Fitur untuk Bot

```
        command.save() # Simpan command ke
database
        return redirect('manage_bots',
bot_id=bot.id) # Redirect ke halaman edit bot
setelah sukses menambah command
    else:
        form = BotCommandForm() # Jika GET,
tampilkan form kosong

    # Render template add_command.html dengan
form dan data bot
    return render(request,
'bots/add_command.html', {'form': form, 'bot':
bot})
```

Fungsi `add_command` dimulai dengan memeriksa apakah pengguna sudah login menggunakan dekorator `@login_required`. Selanjutnya, kita mengambil objek bot berdasarkan `bot_id` yang diberikan. Dengan menggunakan `get_object_or_404`, kita memastikan bahwa bot yang diminta ada dan dimiliki oleh pengguna yang sedang login. Jika bot tidak ditemukan, pengguna akan diarahkan ke halaman 404.

Jika metode permintaan adalah `POST`, kita menginisialisasi form dengan data yang diterima. Setelah memvalidasi form, kita menyimpan data command yang baru, tetapi belum menyimpannya ke database dengan cara langsung. Sebagai langkah pertama, kita mengaitkan command dengan bot yang sesuai, dan kemudian kita menyimpan command tersebut ke database.

Setelah command berhasil ditambahkan, pengguna akan diarahkan kembali ke halaman edit bot menggunakan `redirect`. Jika

Membuat Fitur untuk Bot

metode permintaan adalah GET, kita menampilkan form kosong untuk pengguna mengisi data command yang baru.

Terakhir, kita merender template `add_command.html`, menyediakan form dan data bot untuk ditampilkan kepada pengguna. Dengan view ini, kita telah menyiapkan fondasi yang diperlukan untuk menambahkan command baru ke bot.

Setelah membuat view, langkah berikutnya adalah membuat template HTML untuk menampilkan form ini. Template ini akan berfungsi sebagai antarmuka bagi pengguna untuk menambahkan command baru. Pada sub-bab selanjutnya, kita akan membahas pembuatan template HTML tersebut.

12.4.2 Membuat Template untuk Menambah Command

Setelah menyiapkan view untuk menambah command, langkah berikutnya adalah membuat template HTML yang akan digunakan untuk menampilkan form penambahan command kepada pengguna. Template ini merupakan antarmuka yang akan digunakan oleh pengguna untuk memasukkan informasi tentang command baru yang ingin ditambahkan ke bot mereka.

Berikut adalah kode untuk template `add_command.html`:

```
<!-- File: apps/templates/bots/add_command.html -->

{% extends 'dashboard/base.html' %}
```

Membuat Fitur untuk Bot

```
{% block content %}
<div class="container mt-4">
  <h2>Tambah Command untuk Bot: {{ bot.name
}}</h2>

  <div class="card">
    <div class="card-header">
      <h5 class="card-title">Tambah
Command</h5>
    </div>
    <div class="card-body">
      <form method="post">
        {% csrf_token %}

        <!-- Form Group for Command
Field -->
        <div class="form-group">
          <label
for="{{ form.command.id_for_label }}">Comman
d</label>
          <input type="text"
name="command" id="{{ form.command.id_for_label
}}" value="{{ form.command.value }}"
class="form-control">
          </div>

        <!-- Form Group for Response
Field -->
        <div class="form-group">
          <label
for="{{ form.response.id_for_label }}">Response<
/label>
          <textarea name="response"
id="{{ form.response.id_for_label }}"
class="form-control">{{ form.response.value
}}</textarea>
          </div>
```

Membuat Fitur untuk Bot

```
        <button type="submit" class="btn btn-primary">Tambah Command</button>
    </form>
</div>
</div>
</div>
{% endblock %}
```

Dalam template ini, kita menggunakan sintaks templating Django untuk mengatur struktur halaman. Template ini memperluas template dasar yang kita gunakan dalam aplikasi, yang disebut `base.html`. Dengan menggunakan blok `content`, kita menempatkan konten khusus untuk halaman ini.

Di dalam konten, kita menampilkan judul halaman yang menunjukkan nama bot yang sedang ditambahkan command-nya. Selanjutnya, kita membuat sebuah kartu (card) yang berisi form untuk menambah command. Form ini menggunakan metode `POST` untuk mengirimkan data ke server.

Di dalam form, kita menggunakan token `CSRF` untuk melindungi dari serangan `CSRF`. Kemudian, kita membuat dua grup form: satu untuk field `command` dan satu lagi untuk field `response`. Field `command` diisi dengan input teks, sedangkan field `response` diisi dengan `textarea` agar pengguna bisa memasukkan respons yang lebih panjang.

Akhirnya, terdapat tombol untuk mengirim form yang bertuliskan "Tambah Command". Setelah pengguna mengisi form dan mengklik tombol ini, data akan diproses oleh view yang telah kita buat sebelumnya. Template ini dirancang untuk memberikan

Membuat Fitur untuk Bot

pengalaman pengguna yang intuitif dalam menambahkan command untuk bot mereka.

Dengan template ini, Anda sudah memiliki antarmuka pengguna yang siap untuk digunakan dalam menambahkan command ke bot. Langkah selanjutnya adalah menambahkan routing URL dan pengaturan untuk mengintegrasikan template ini dengan fitur lain dalam aplikasi, yang akan dibahas di sub-bab berikutnya.

12.4.3 Mengatur URL Routing untuk Fitur Tambah Command

Untuk memastikan fitur penambahan command dapat diakses oleh pengguna, kita perlu mengatur URL routing di Django. Routing URL akan menghubungkan view yang telah dibuat dengan URL yang dapat diakses melalui browser. Proses ini melibatkan penambahan rute baru ke dalam konfigurasi URL aplikasi dan memastikan bahwa view `add_command` dapat diakses melalui URL yang sesuai.

Konfigurasi URL untuk aplikasi Django biasanya ditemukan dalam file `urls.py`. Dalam proyek ini, kita perlu menambahkan rute baru di file `urls.py` yang berada di direktori aplikasi `bots`.

Berikut adalah contoh kode untuk mengatur URL routing:

```
# File: apps/bots/urls.py
from django.urls import path
```

Membuat Fitur untuk Bot

```
from .views import add_command

urlpatterns = [
    # Rute lain di sini...
    path('add_command/<int:bot_id>/',
    views.add_command, name='add_command'),
]
```

Dalam kode di atas, kita mengimpor fungsi `add_command` dari `views.py`. Kemudian, kita menambahkan entri baru dalam `urlpatterns`, yang mendefinisikan rute untuk menambah command. Rute ini menggunakan path `add_command/<int:bot_id>/`, di mana `<int:bot_id>` adalah parameter yang akan menangkap ID bot yang sedang ditambahkan command-nya.

Dengan cara ini, ketika pengguna mengakses URL seperti `yourdomain.com/add_command/1/`, aplikasi kita akan memanggil fungsi `add_command`, dan ID bot yang relevan akan diproses untuk menambahkan command yang baru. Nama rute ini diberikan sebagai `'add_command'`, sehingga kita dapat merujuknya dalam template dan view lain dengan mudah.

Pengaturan URL yang tepat sangat penting untuk menjaga struktur aplikasi kita tetap rapi dan mudah dinavigasi. Setelah routing diatur, pengguna dapat dengan mudah menambah command untuk bot mereka dengan mengikuti URL yang telah ditentukan.

12.4.4 Menampilkan Daftar Command

Setelah kita berhasil menambahkan command ke bot, langkah selanjutnya adalah menampilkan daftar command tersebut di halaman manajemen bot. Dengan cara ini, pengguna dapat melihat semua command yang telah mereka buat, serta melakukan pengeditan atau penghapusan jika diperlukan.

Untuk menampilkan daftar command, kita akan memodifikasi template `manage_bots.html` untuk menambahkan bagian yang menunjukkan command yang terkait dengan setiap bot. Berikut adalah cara kita dapat melakukannya:

```
<!-- File: apps/templates/bots/manage_bots.html -->

{% extends 'dashboard/base.html' %}

{% block content %}
    <div class="container">
        <h1>Manage Bots</h1>

        <!-- Tampilkan pesan sukses jika ada -->
        {% if messages %}
            <div class="alert alert-success alert-dismissible fade show" role="alert">
                {% for message in messages %}
                    {{ message }}
                {% endfor %}
                <button type="button" class="btn-close" data-bs-dismiss="alert" aria-label="Close"></button>
            </div>
        {% endif %}
    </div>
</block>
```


Membuat Fitur untuk Bot

```
<table class="table table-striped">
  <thead>
    <tr>
      <th>ID</th>
      <th>Name</th>
      <th>Platform</th>
      <th>Token</th>
      <th>Status</th>
      <th>Actions</th>
    </tr>
  </thead>
  <tbody>
    {% for bot in bots %}
      <tr>
        <td>{{ bot.id }}</td>
        <td>{{ bot.name }}</td>
        <td>{{ bot.bot_type }}</td>
        <td>{{ bot.token_telegram }}</td>
        <td>{{ bot.is_active |
yesno:"Active,Inactive" }}</td>
        <td>
          <a href="" class="btn btn-
primary">Edit</a>
          <form method="post"
style="display:inline;">
            {% csrf_token %}
            <button type="submit"
name="delete" value="{{ bot.id }}"
onclick="return confirm('Are you sure you want
to delete this bot?');" class="btn btn-
danger">Delete</button>
          </form>
        </td>
      </tr>
    </tbody>
  </table>
  <!-- Menampilkan command terkait
untuk setiap bot -->
  <ul>
```

Membuat Fitur untuk Bot

```
        {% for command in
bot.commands.all %}
        <li>{{ command.command }} - {{
command.response }}</li>
        {% empty %}
        <li>No commands
available.</li>
        {% endfor %}
    </ul>
    <a href="{% url 'add_command'
bot.id %}" class="btn btn-success">Tambah
Command</a>
    </td>
</tr>
{% empty %}
<tr>
    <td colspan="6">No bots
available.</td>
</tr>
{% endfor %}
</tbody>
</table>

    <a href="{% url 'create_bot' %}" class="btn
btn-primary">Buat Bot Baru</a>
</div>
{% endblock %}
```

Pada kode di atas, kami menambahkan kolom baru di tabel yang menampilkan command untuk setiap bot. Dengan menggunakan `{% for command in bot.commands.all %}`, kita bisa mengiterasi semua command yang terkait dengan bot tersebut. Jika tidak ada command yang tersedia, pesan "No commands available." akan ditampilkan.

Membuat Fitur untuk Bot

Di samping itu, kami juga menyediakan tombol untuk menambahkan command baru yang mengarah ke halaman penambahan command yang telah kita buat sebelumnya. Ini memberi pengguna akses cepat untuk memperkaya interaksi bot mereka dengan menambahkan lebih banyak command.

12.4.5 Memperbaiki Handle update

Agar bot bisa merespon sesuai dengan command yang ditambahkan, kita harus memperbaiki handle untuk bot agar bot bisa merespon dengan command yang ditambahkan.

Berikut perbaikan handle agar bot bisa merespon dengan benar:

Telegram.py

```
# File: apps/bots/services/telegram.py
from apps.bots.models import Bot, BotCommand
import requests

def handle_update(update):
    chat_id = update['message']['chat']['id']
    text = update['message']['text'].lower() #
    # Mengubah ke lowercase untuk konsistensi
    # perbandingan

    bot =
    Bot.objects.filter(bot_type='telegram').first()
    # Misalkan kita ambil bot pertama, bisa
    # disesuaikan

    if bot:
        # Cek apakah perintah cocok dengan yang
        # ada di model BotCommand
```

Membuat Fitur untuk Bot

```
        command =
BotCommand.objects.filter(bot=bot,
command__iexact=text).first()

        if command:
            response_text = command.response
        else:
            # Jika perintah tidak ditemukan,
gunakan perintah default /start dan /help
            if text == "/start":
                response_text =
bot.start_message
            elif text == "/help":
                response_text = bot.help_message
            else:
                response_text = 'Perintah tidak
dikenal. Ketik /help untuk melihat daftar
perintah.'

        send_message(bot.token, "sendMessage", {
            'chat_id': chat_id,
            'text': response_text
        })

def send_message(token, method, data):
    telegram_api_url =
f'https://api.telegram.org/bot{token}/{method}'
    response = requests.post(telegram_api_url,
data=data)
    if response.status_code != 200:
        logger.error(f"Failed to send message:
{response.text}") # Log error jika gagal
    return response
```

Dengan menautkan handle ke model BotCommand ,sekarang bot bisa merespon sesuai dengan comman yang di tambahkan.

Membuat Fitur untuk Bot

Whatsapp.py

```
# File: apps/bots/services/whatsapp.py

from django.http import HttpResponse # Untuk
mengembalikan respon HTTP
from django.views.decorators.csrf import
csrf_exempt # Agar view dapat diakses tanpa
validasi CSRF
from twilio.twiml.messaging_response import
MessagingResponse # Untuk membuat balasan ke
Twilio API
from apps.bots.models import Bot, BotCommand #
Mengimpor model Bot dan BotCommand dari aplikasi
bots
import logging # Untuk logging aktivitas dan
error

logger = logging.getLogger(__name__) #
Inisialisasi logger

@csrf_exempt # Dekorator untuk menonaktifkan
validasi CSRF pada view ini
def handle_update(update):
    # Mendapatkan pengirim pesan dan isi pesan
    user = update['From']
    message = update['Body'].strip().lower() #
    Mengubah pesan ke lowercase untuk perbandingan

    # Logging pesan yang diterima
    logger.info(f'{user} says {message}')

    # Mengambil bot pertama yang bertipe
    WhatsApp dari database
    bot =
    Bot.objects.filter(bot_type='whatsapp').first()
```

Membuat Fitur untuk Bot

```
response = MessagingResponse() # Membuat
respon menggunakan Twilio's MessagingResponse

if bot: # Jika bot ditemukan
    # Cek apakah perintah cocok dengan yang
    ada di model BotCommand
    command =
BotCommand.objects.filter(bot=bot,
command__iexact=message).first()

    if command:
        response_text = command.response #
Dapatkan respons dari perintah
    else:
        # Jika perintah tidak ditemukan,
        gunakan perintah default /start dan /help
        if message == "/start":
            response_text =
bot.start_message # Pesan sambutan
        elif message == "/help":
            response_text = bot.help_message
# Pesan bantuan
        else:
            response_text = 'Perintah tidak
dikenal. Ketik /help untuk melihat daftar
perintah.' # Pesan default

    response.message(response_text) #
Mengirim balasan pesan
    else:
        response.message('Bot tidak ditemukan.')
# Jika bot tidak ditemukan di database

# Mengembalikan respon sebagai HTTP response
return HttpResponse(str(response))
```

Penjelasan Perubahan:

Membuat Fitur untuk Bot

1. **Impor Model BotCommand:** Kita mengimpor model BotCommand untuk mengambil perintah yang ditentukan pengguna dari database.
2. **Pemeriksaan Perintah:** Mirip dengan kode Telegram, kita sekarang memeriksa apakah pesan yang diterima cocok dengan perintah yang ada di model BotCommand. Jika perintah ditemukan, respons diambil dari `command.response`.
3. **Pesan Default:** Jika perintah tidak cocok dengan perintah dalam database, kita menggunakan perintah default seperti `/start` dan `/help`.
4. **Respon:** Respon dari Twilio dihasilkan menggunakan `MessagingResponse`, dan jika bot tidak ditemukan, pesan yang sesuai dikirim kembali.

Dengan pendekatan ini, penanganan untuk WhatsApp menjadi lebih konsisten dengan penanganan untuk Telegram, yang memudahkan pemeliharaan dan pengembangan lebih lanjut.

Dengan fitur ini, manajemen bot menjadi lebih intuitif dan terorganisir, memberikan pengguna kemampuan untuk mengelola command mereka dengan lebih efektif.

12.5 Menyiapkan Fitur Untuk Mengedit Command

Setelah menyiapkan fitur untuk menambah command pada bot, langkah selanjutnya adalah membuat fitur untuk mengedit com-

Membuat Fitur untuk Bot

mand yang sudah ada. Mengedit command memungkinkan pengguna untuk memperbarui perintah dan respons yang dikirim oleh bot. Proses ini melibatkan pembuatan view yang memungkinkan pengguna untuk mengedit command melalui antarmuka web.

12.5.1 Membuat View untuk Mengedit Command

Setelah kita berhasil menambahkan fitur untuk menambah command, kini saatnya untuk memberikan kemampuan kepada pengguna untuk mengedit command yang telah ada. Fitur ini penting agar pengguna dapat memperbarui perintah dan respons sesuai dengan kebutuhan mereka. Dalam sub bab ini, kita akan membuat view yang menangani proses pengeditan command.

Pada langkah pertama, kita perlu mendefinisikan fungsi `edit_command` di dalam berkas `views.py`. Fungsi ini akan menerima `command_id` sebagai parameter untuk mengidentifikasi command yang ingin diedit. Jika command ditemukan, kita akan menampilkan form pengeditan dengan data yang sudah ada.

Berikut adalah implementasi dari fungsi tersebut:

```
# File: apps/bots/views.py

@login_required
def edit_command(request, command_id):
    # Mengambil command berdasarkan ID
    command = get_object_or_404(BotCommand,
                                id=command_id) # Mengambil command, jika tidak
                                                # ditemukan akan mengembalikan 404
```


Membuat Fitur untuk Bot

```
bot = command.bot # Mengambil bot terkait
dengan command

if request.method == 'POST':
    # Jika metode POST, proses data form
    form = BotCommandForm(request.POST,
instance=command) # Menggunakan instance
command untuk mengisi form
    if form.is_valid():
        form.save() # Menyimpan perubahan
command
        return redirect('manage_bots',
bot_id=bot.id) # Redirect ke halaman edit bot
setelah menyimpan
    else:
        # Jika metode GET, tampilkan form dengan
data command yang ada
        form = BotCommandForm(instance=command)
# Menampilkan form dengan data yang ada

        return render(request,
'bots/edit_command.html', {'form': form, 'bot':
bot}) # Render template dengan form dan data
bot
```

Dalam fungsi di atas, kami menggunakan decorator `@login_required` untuk memastikan bahwa hanya pengguna yang telah login yang dapat mengakses fitur ini. Pertama-tama, kita mencoba mengambil command berdasarkan `command_id` yang diterima. Jika command tidak ditemukan, pengguna akan mendapatkan halaman 404. Setelah itu, kita mengambil bot yang terkait dengan command tersebut agar kita dapat mengarahkan pengguna kembali ke halaman edit bot setelah pengeditan selesai.

Membuat Fitur untuk Bot

Jika permintaan yang diterima adalah metode POST, ini berarti pengguna telah mengisi form dan mengirimkan data. Kita akan membuat instance dari `BotCommandForm` dengan data yang dikirimkan dan mengaitkannya dengan command yang sedang diedit. Jika form valid, kita simpan perubahan dan redirect ke halaman edit bot yang sesuai. Sebaliknya, jika permintaan adalah metode GET, kita hanya menampilkan form pengeditan dengan data dari command yang ada.

Dengan fitur pengeditan ini, pengguna dapat dengan mudah menyesuaikan command sesuai dengan kebutuhan bot mereka, meningkatkan fleksibilitas dan kemampuan bot dalam memberikan respons yang lebih relevan.

12.5.2 Membuat Template untuk Mengedit Command

Setelah kita memiliki view yang berfungsi untuk mengedit command, langkah selanjutnya adalah membuat template yang akan menampilkan form pengeditan tersebut. Template ini akan memberikan antarmuka pengguna yang intuitif untuk memperbarui command yang telah ada.

Kita akan membuat file bernama `edit_command.html` di dalam folder `templates/bots`. Template ini akan mewarisi dari template dasar `base.html` yang sudah ada, sehingga tetap konsisten dengan tampilan aplikasi. Berikut adalah implementasi dari template tersebut:

Membuat Fitur untuk Bot

```
<!-- File: apps/templates/bots/edit_command.html
-->

{% extends 'dashboard/base.html' %}

{% block content %}
<div class="container mt-4">
  <h2>Edit Command untuk Bot: {{ bot.name }}</h2>

  <div class="card">
    <div class="card-header">
      <h5 class="card-title">Edit
Command</h5>
    </div>
    <div class="card-body">
      <form method="post">
        {% csrf_token %}

        <!-- Form Group for Command
Field -->
        <div class="form-group">
          <label
for="{{ form.command.id_for_label }}">Comman
d</label>
          <input type="text"
name="command" id="{{ form.command.id_for_label
}}" value="{{ form.command.value }}"
class="form-control">
          </div>

        <!-- Form Group for Response
Field -->
        <div class="form-group">
          <label
for="{{ form.response.id_for_label }}">Response<
/label>
          <textarea name="response"
id="{{ form.response.id_for_label }}"
```

Membuat Fitur untuk Bot

```
class="form-control">{{ form.response.value
}}</textarea>
        </div>
        <button type="submit" class="btn
btn-primary">Update Command</button>
    </form>
</div>
</div>
</div>
{% endblock %}
```

Dalam template di atas, kita memanfaatkan `block content` untuk menentukan isi halaman. Judul halaman mencerminkan bot yang sedang diedit, memberi konteks yang jelas kepada pengguna. Form pengeditan dibangun menggunakan metode POST untuk mengirimkan data ke server.

Setiap field dalam form diwakili oleh elemen HTML yang sesuai. Kita menampilkan field untuk command dan response, menggunakan nilai yang ada agar pengguna dapat melihat dan mengedit data yang sudah ada. Label untuk setiap field juga ditambahkan untuk meningkatkan aksesibilitas dan pemahaman pengguna.

Setelah pengguna mengisi form dan menekan tombol "Update Command", data yang diperbarui akan dikirim ke view `edit_command` untuk diproses. Jika semua berjalan dengan baik, pengguna akan diarahkan kembali ke halaman edit bot yang relevan, di mana perubahan dapat dilihat dan diperiksa.

Dengan template ini, pengguna mendapatkan pengalaman yang mudah dan efisien dalam mengedit command mereka, memungkinkan fleksibilitas dalam pengaturan bot.

Membuat Fitur untuk Bot

Dengan template HTML yang sudah siap, pengguna dapat mengedit command dengan mudah melalui antarmuka web. Langkah berikutnya adalah memastikan bahwa semua fitur terkait pengeditan command berfungsi dengan baik dan memverifikasi bahwa data yang diperbarui disimpan dengan benar di database.

12.5.3 Mengatur URL Routing untuk Fitur Edit Command

Setelah menyiapkan view dan template untuk mengedit command, langkah berikutnya adalah mengatur routing URL. Routing ini penting untuk menghubungkan URL yang dimasukkan oleh pengguna dengan fungsi view yang sesuai. Dengan pengaturan yang benar, pengguna dapat mengakses halaman edit command dengan mudah.

Kita akan menambahkan rute baru dalam file `urls.py` di dalam aplikasi bots kita. Berikut adalah potongan kode yang perlu ditambahkan:

```
# File: apps/bots/urls.py

from django.urls import path
from . import views

urlpatterns = [
    # URL lain
    path('edit_command/<int:command_id>/',
        views.edit_command, name='edit_command'),
]
```

Membuat Fitur untuk Bot

Pada kode di atas, kita menambahkan URL pattern baru menggunakan fungsi `path()`. Rute ini akan menerima `command_id` sebagai parameter yang menunjukkan command mana yang ingin diedit. Ketika pengguna mengakses URL yang sesuai, misalnya `/edit_command/1/`, Django akan memanggil fungsi `edit_command` yang sudah kita definisikan sebelumnya.

Menentukan nama untuk rute ini juga sangat penting. Dalam hal ini, kita menggunakan `name='edit_command'`. Penamaan ini memudahkan kita untuk merujuk ke rute ini di bagian lain dari aplikasi, seperti saat membuat link menuju halaman edit command.

Dengan mengatur routing URL ini, kita menjamin bahwa pengguna dapat dengan mudah mengedit command yang telah mereka buat sebelumnya. Proses ini membantu menciptakan pengalaman pengguna yang lebih baik dan membuat aplikasi kita lebih responsif terhadap tindakan pengguna.

12.6 Integrasi Dengan Model Log Aktivitas

Dalam pengembangan platform bot, melacak aktivitas pengguna adalah aspek yang sangat penting, terutama dalam konteks pengelolaan bot. Dengan integrasi fitur log aktivitas, kita dapat memonitor aksi-aksi penting yang dilakukan oleh pengguna, seperti membuat bot, mengedit bot, menambah command, atau menghapus bot. Hal ini tidak hanya berguna untuk keperluan audit, tetapi

Membuat Fitur untuk Bot

juga memberikan visibilitas yang lebih baik mengenai interaksi yang terjadi dalam aplikasi.

Di bagian ini, kita akan membahas bagaimana mengintegrasikan model log aktivitas ke dalam `fmembuat bot`, `mengedit bot`, `membuat command`, dan `mengedit command`. Setiap kali pengguna melakukan tindakan tersebut, kita akan menyimpan log aktivitasnya. Setelah itu, kita akan menampilkan log tersebut di *dashboard*.

Kita perlu memodifikasi setiap *view* di mana pengguna membuat atau mengedit bot atau command, untuk mencatat aktivitasnya ke dalam model `ActivityLog`.

12.6.1 Integrasi Untuk Create Bot

Pertama-tama, kita akan mengintegrasikan log aktivitas ketika pengguna membuat bot. Ketika pengguna men-submit form untuk membuat bot baru, sistem akan otomatis menyimpan aktivitas ini ke dalam model log aktivitas yang sudah kita definisikan sebelumnya. Dengan demikian, setiap kali bot dibuat, aktivitas tersebut akan tercatat.

Misalnya, setelah bot berhasil dibuat, kita bisa menambahkan baris kode untuk menyimpan aktivitas ini ke dalam log. Berikut adalah contoh implementasinya pada fungsi `create_bot` di dalam file `views`:

```
# File: apps/bots/views.py
```

Membuat Fitur untuk Bot

```
from apps.dashboard.models import ActivityLog

@login_required
def create_bot(request):
    if request.method == 'POST':
        form = BotForm(request.POST)
        if form.is_valid():
            bot = form.save(commit=False)
            bot.user = request.user
            bot.is_active = True # Set bot
aktif secara default

            # Tentukan webhook URL sesuai dengan
platform
            if bot.bot_type == 'telegram':
                bot.webhook_url =
f'{WEBHOOK_URL}/telegram/'
            elif bot.bot_type == 'whatsapp':
                bot.webhook_url =
f'{WEBHOOK_URL}/whatsapp/'

            bot.save()

            # Set webhook berdasarkan tipe bot
            if bot.bot_type == 'telegram':

set_telegram_webhook(bot.token_telegram,
bot.webhook_url)
                # Tambahkan pesan sukses untuk
bot Telegram
                messages.success(request, 'Bot
Telegram berhasil dibuat!')
            elif bot.bot_type == 'whatsapp':

set_whatsapp_webhook(bot.webhook_url)
                # Tambahkan pesan sukses untuk
bot WhatsApp
                messages.success(request, f'Bot
WhatsApp berhasil dibuat. Silakan atur URL
```


Membuat Fitur untuk Bot

```
webhook berikut di dashboard Twilio:
{bot.webhook_url}')

        # Tambahkan log aktivitas
        ActivityLog.objects.create(
            user=request.user,
            action=f"Membuat bot baru:
{bot.name}"
        )

        return redirect('manage_bots')
    else:
        form = BotForm()

    return render(request,
'bots/create_bot.html', {'form': form})
```

Pada kode di atas, setelah bot berhasil dibuat dan disimpan ke dalam database, kita menambahkan log aktivitas menggunakan model `ActivityLog`. Aksi ini mencatat informasi seperti user yang membuat bot, deskripsi aktivitas, dan referensi ke objek bot yang baru dibuat. Semua informasi ini akan disimpan di tabel log, sehingga dapat dengan mudah diakses kapan saja di masa mendatang.

12.6.2 Integrasi Untuk Edit Bot

Selain mencatat aktivitas pembuatan bot, kita juga perlu mencatat ketika pengguna mengedit bot. Sama seperti create bot, saat pengguna memperbarui informasi bot, aktivitas ini juga perlu dicatat ke dalam log.

Berikut adalah contoh implementasi untuk integrasi log aktivitas di dalam fungsi `edit_bot`:

Membuat Fitur untuk Bot

```
@login_required
def edit_bot(request, bot_id):
    bot = get_object_or_404(Bot, id=bot_id) #
    Ambil bot berdasarkan ID
    if request.method == 'POST':
        form = BotForm(request.POST,
instance=bot) # Muat data ke form
        if form.is_valid():
            form.save() # Simpan data yang
telah diedit

            # Tambahkan log aktivitas setelah
bot berhasil diedit
            ActivityLog.objects.create(
                user=request.user,
                action=f"Bot {bot.name} telah
diperbarui",
                bot=bot
            )

            return redirect('manage_bots') #
Alihkan ke halaman pengelolaan bot
    else:
        form = BotForm(instance=bot) # Jika
bukan POST, buat form dengan data bot

        context = {
            'form': form,
            'bot': bot,
        }
        return render(request, 'bots/edit_bot.html',
context) # Tampilkan template dengan context
```

12.6.3 Integrasi Untuk Hapus Bot

Menghapus bot dari platform bot creation juga merupakan aktivitas penting yang harus dicatat ke dalam log aktivitas. Seperti halnya mencatat pembuatan dan pengeditan bot, menghapus bot memungkinkan pengguna dan administrator memonitor perubahan yang dilakukan di dalam sistem. Dengan mencatat aktivitas ini, kita dapat meninjau riwayat penghapusan yang dilakukan oleh pengguna tertentu, yang sangat berguna untuk keperluan audit maupun troubleshooting.

Pada bagian ini, kita akan mengintegrasikan fitur log aktivitas ke dalam fungsi penghapusan bot. Setiap kali pengguna menghapus bot, kita akan mencatat informasi seperti siapa yang menghapus bot tersebut dan bot mana yang dihapus.

Fungsi `manage_bots` yang sudah ada bertanggung jawab untuk menampilkan daftar bot yang dimiliki oleh pengguna dan juga untuk menangani penghapusan bot melalui permintaan POST. Mari kita tambahkan log aktivitas ke dalam fungsi ini agar setiap penghapusan bot dapat dicatat dengan baik.

Berikut adalah implementasi yang sudah diintegrasikan dengan log aktivitas:

```
# File: apps/bots/views.py

@login_required # Pastikan hanya pengguna yang
login yang bisa mengakses view ini
def manage_bots(request):
```

Membuat Fitur untuk Bot

```
bots = Bot.objects.filter(user=request.user)
# Ambil daftar bot milik pengguna yang sedang
login

    if request.method == 'POST':
        if 'delete' in request.POST: # Cek
            apakah ada permintaan untuk menghapus bot
            bot_id = request.POST.get('delete')
            # Ambil ID bot yang ingin dihapus dari form
            bot = get_object_or_404(Bot,
            id=bot_id) # Cari bot berdasarkan ID, atau
            tampilkan 404 jika tidak ditemukan

            # Menyimpan aktivitas penghapusan ke
            dalam log sebelum bot dihapus
            ActivityLog.objects.create(
                user=request.user,
                action=f"Bot {bot.name} telah
dihapus"
            )
            bot.delete() # Hapus bot dari
            database

            messages.success(request, 'Bot
berhasil dihapus.') # Tambahkan pesan setelah
            bot dihapus

            return redirect('manage_bots') #
            Setelah bot dihapus, kembali ke halaman
            manajemen bot

    return render(request,
    'bots/manage_bots.html', {'bots': bots}) #
    Render halaman dengan daftar bot yang dimiliki
    pengguna
```

Pada kode di atas, terdapat beberapa langkah kunci yang dilakukan ketika bot dihapus:

Membuat Fitur untuk Bot

1. **Mengambil bot yang dihapus:** Ketika pengguna menekan tombol 'delete' pada salah satu bot yang mereka miliki, sistem akan menerima ID bot tersebut melalui POST. Kemudian, dengan fungsi `get_object_or_404`, kita mendapatkan objek Bot yang sesuai dengan ID tersebut.
2. **Mencatat aktivitas penghapusan:** Sebelum bot dihapus, kita mencatat aksi penghapusan ke dalam model `ActivityLog`. Informasi yang disimpan mencakup user yang melakukan penghapusan, aksi yang dilakukan (dalam hal ini penghapusan bot), dan referensi ke bot yang dihapus.
3. **Penghapusan bot:** Setelah aktivitas dicatat, bot kemudian dihapus dari database dengan metode `delete()`. Setelah itu, pengguna akan diarahkan kembali ke halaman `manage_bots`, yang menampilkan daftar bot yang mereka miliki.

Dengan cara ini, setiap kali bot dihapus oleh pengguna, aktivitas tersebut akan tercatat ke dalam sistem, memungkinkan administrator untuk memonitor aktivitas penghapusan yang terjadi. Log aktivitas memberikan visibilitas tambahan ke dalam perubahan yang terjadi pada sistem, yang sangat penting untuk menjaga integritas dan keamanan aplikasi.

Integrasi log aktivitas ini merupakan langkah penting dalam memastikan bahwa setiap aksi yang signifikan di dalam aplikasi dicatat dengan baik.

12.6.4 Integrasi Untuk Add Command

Pada tahap ini, kita akan mengintegrasikan log aktivitas ke dalam fitur penambahan command. Setiap kali pengguna menambahkan command baru ke bot yang telah mereka buat, tindakan tersebut akan dicatat dalam log aktivitas, memastikan bahwa setiap perubahan yang signifikan dalam sistem dapat ditelusuri. Dengan mencatat setiap penambahan command, administrator dan pengguna dapat memonitor dengan lebih baik bagaimana interaksi dilakukan terhadap bot-bot yang ada.

Dalam proses ini, kita akan menggunakan model Activity-Log untuk mencatat informasi terkait siapa yang menambahkan command, command apa yang ditambahkan, dan untuk bot mana command tersebut ditambahkan. Hal ini membantu dalam memberikan visibilitas penuh terhadap semua interaksi yang terjadi dalam aplikasi.

Kode di bawah ini menunjukkan bagaimana fitur penambahan command telah diintegrasikan dengan log aktivitas:

```
# File: apps/bots/views.py

@login_required
def add_command(request, bot_id):
    bot = get_object_or_404(Bot, id=bot_id,
user=request.user) # Mendapatkan bot
berdasarkan ID dan memastikan user adalah
pemilik bot

    if request.method == 'POST':
```

Membuat Fitur untuk Bot

```
        form = BotCommandForm(request.POST) #
Membuat form berdasarkan data POST
        if form.is_valid():
            command = form.save(commit=False) #
Simpan command baru tapi belum ke database
            command.bot = bot # Menetapkan bot
terkait dengan command
            command.save() # Menyimpan command
ke database

            # Mencatat aktivitas penambahan
command ke dalam log
            ActivityLog.objects.create(
                user=request.user,
                action=f"Menambahkan command
baru: {command.command} untuk bot {bot.name}"
            )

            return redirect('manage_bots',
bot_id=bot.id) # Setelah berhasil menyimpan,
redirect ke halaman edit bot
        else:
            form = BotCommandForm() # Jika GET,
tampilkan form kosong

            return render(request,
'bots/add_command.html', {'form': form, 'bot':
bot}) # Render template dengan form
```

Pada kode di atas, proses integrasi log aktivitas dilakukan melalui beberapa langkah utama:

1. **Mendapatkan objek bot:** Dengan menggunakan `get_object_or_404`, sistem memastikan bahwa bot yang ingin ditambahkan command-nya benar-benar ada dan bahwa pengguna yang melakukan tindakan adalah pemilik dari bot tersebut. Ini adalah langkah penting da-

Membuat Fitur untuk Bot

lam memastikan bahwa tidak ada user lain yang bisa menambahkan command ke bot yang bukan miliknya.

2. ***Proses pengisian form***: Jika metode request adalah POST, form yang diisi oleh pengguna akan diproses dan divalidasi. Apabila validasi form berhasil, objek `command` akan disimpan ke dalam database dengan menggunakan `form.save(commit=False)` yang memungkinkan kita menambahkan properti tambahan (seperti bot terkait) sebelum command benar-benar disimpan.
3. ***Mencatat log aktivitas***: Setelah command berhasil disimpan, kita mencatat aksi penambahan command tersebut ke dalam model `ActivityLog`. Log ini mencatat informasi seperti siapa pengguna yang menambahkan command dan command apa yang ditambahkan ke bot. Dengan mencatat informasi ini, kita memastikan setiap perubahan yang signifikan terhadap bot dapat dilacak dengan mudah.
4. ***Mengalihkan ke halaman edit bot***: Setelah command berhasil ditambahkan dan aktivitas dicatat, pengguna akan diarahkan kembali ke halaman `edit_bot` untuk bot tersebut, memungkinkan mereka untuk segera melihat command baru yang telah mereka tambahkan.

Fitur penambahan command yang diintegrasikan dengan log aktivitas ini memberikan kemampuan kepada pengguna untuk memonitor setiap tindakan yang terjadi di dalam aplikasi, baik itu untuk keperluan audit atau troubleshooting. Pengguna dan administrator kini dapat mengetahui dengan tepat kapan dan oleh siapa command-command baru ditambahkan ke bot.

12.6.5 Integrasi Untuk Edit Command

Proses pengeditan command dalam sebuah bot adalah fitur yang sangat penting, karena command yang sudah ditambahkan sebelumnya mungkin perlu diperbarui untuk menyesuaikan dengan perubahan kebutuhan pengguna atau bot. Untuk memastikan setiap perubahan pada command ini tercatat dengan baik, kita perlu mengintegrasikan log aktivitas pada fitur edit command. Hal ini memungkinkan kita untuk mengetahui kapan dan oleh siapa command telah diubah, memberikan transparansi yang lebih baik dalam sistem.

Dalam kasus ini, ketika seorang pengguna mengedit command, aplikasi akan mencatat perubahan tersebut dalam model `ActivityLog`. Dengan begitu, setiap tindakan pengeditan yang dilakukan dapat dilacak dan dipantau. Proses ini hampir serupa dengan fitur penambahan command, hanya saja di sini kita fokus pada pengeditan command yang sudah ada.

Kode berikut menjelaskan bagaimana kita mengimplementasikan fitur pengeditan command yang terintegrasi dengan log aktivitas:

```
# File: apps/bots/views.py

@login_required
def edit_command(request, command_id):
    command = get_object_or_404(BotCommand,
id=command_id) # Ambil command berdasarkan ID
    bot = command.bot # Mendapatkan bot terkait
    dengan command
```

Membuat Fitur untuk Bot

```
if request.method == 'POST':
    form = BotCommandForm(request.POST,
instance=command) # Muat data POST ke dalam
form
    if form.is_valid():
        form.save() # Simpan perubahan
command ke dalam database

        # Catat aktivitas pengeditan command
ke dalam log
        ActivityLog.objects.create(
            user=request.user,
            action=f"Mengedit command:
{command.command} di bot {bot.name}"
        )

        return redirect('manage_bots',
bot_id=bot.id) # Setelah berhasil menyimpan,
alihkan kembali ke halaman edit bot
    else:
        form = BotCommandForm(instance=command)
# Jika bukan POST, tampilkan form dengan data
command yang ada

        return render(request,
'bots/edit_command.html', {'form': form, 'bot':
bot}) # Render template dengan form dan bot
terkait
```

Pada kode di atas, ada beberapa langkah yang perlu diperhatikan dalam proses integrasi ini:

1. **Mengambil objek command dan bot:** Sistem mengambil command yang ingin diedit berdasarkan ID-nya menggunakan `get_object_or_404`. Dengan cara ini, kita memastikan bahwa command yang akan diedit benar-benar ada dan valid. Kemudian, bot yang terkait dengan

Membuat Fitur untuk Bot

command tersebut juga diambil agar kita bisa menampilkan informasi bot kepada pengguna dan mencatat aktivitas pengeditan untuk bot yang spesifik.

2. ***Pengolahan form***: Sama seperti pada fitur penambahan command, ketika pengguna mengirimkan form melalui metode **POST**, sistem akan memvalidasi data yang dimasukkan. Apabila form valid, perubahan yang dilakukan pada command akan disimpan ke dalam database.
3. ***Pencatatan log aktivitas***: Setelah perubahan command berhasil disimpan, kita langsung mencatat aktivitas tersebut ke dalam model **ActivityLog**. Informasi yang dicatat meliputi siapa pengguna yang melakukan pengeditan, command yang diedit, dan bot tempat command tersebut berada. Pencatatan ini penting untuk memberikan jejak audit yang jelas atas setiap perubahan yang dilakukan dalam aplikasi.
4. ***Pengalihan setelah penyimpanan***: Setelah command berhasil diubah, pengguna akan diarahkan kembali ke halaman edit bot (**edit_bot**), di mana mereka dapat melihat daftar command yang baru saja diubah.

Dengan adanya integrasi log aktivitas untuk fitur edit command ini, setiap perubahan yang dilakukan oleh pengguna menjadi terdata dengan baik. Pengguna lain dan administrator aplikasi dapat memantau siapa saja yang telah melakukan perubahan, serta command apa yang telah diubah, memastikan kontrol dan pengelolaan yang lebih baik terhadap bot dan command yang ada di dalam sistem.

12.6.6 Integrasi Untuk Hapus Command

```
@login_required
def manage_bots(request):
    bots = Bot.objects.filter(user=request.user)
    # Menampilkan hanya bot milik user yang sedang login

    if 'delete_command' in request.POST:
        command_id =
request.POST.get('delete_command') #
Mendapatkan ID command yang akan dihapus
        command =
get_object_or_404(BotCommand, id=command_id) #
Mendapatkan command berdasarkan ID

        # Menyimpan aktivitas penghapusan
command ke dalam log
        ActivityLog.objects.create(
            user=request.user,
            action=f"Menghapus command:
{command.command} dari bot {command.bot.name}"
        )

        command.delete() # Menghapus
command dari database
        return redirect('manage_bots') #
Mengarahkan kembali ke halaman manage bots

    return render(request,
'bots/manage_bots.html', {'bots': bots})
```

Dengan langkah-langkah di atas, kita telah berhasil mengintegrasikan *activity log* untuk setiap tindakan membuat, mengedit bot, membuat command, dan mengedit command, serta menampilkannya di *dashboard*.

12.7 Kode Lengkap Untuk Bots

Kuita akan review kembali kode lengkap di aplikasi bots kita untuk menghindari kesalahan. Berikut adalah kode lengkap untuk semuanya:

12.7.1 Models.py

```
# File: apps/bots/models.py

from django.db import models
from django.contrib.auth.models import User

class Bot(models.Model):
    BOT_CHOICES = [
        ('telegram', 'Telegram'),
        ('whatsapp', 'WhatsApp'),
    ]
    user = models.ForeignKey(User,
on_delete=models.CASCADE) # Bot akan terkait
dengan pengguna
    bot_type = models.CharField(max_length=20,
choices=BOT_CHOICES) # Jenis bot: Telegram atau
WhatsApp

    token_telegram =
models.CharField(max_length=255, unique=True,
help_text="Masukkan token bot yang unik.") #
Token Telegram
    webhook_url =
models.URLField(max_length=255, blank=True,
help_text="URL webhook bot akan diatur secara
otomatis.") # URL webhook

    is_active =
models.BooleanField(default=True,
help_text="Status bot aktif atau tidak.") #
Status aktif atau tidaknya bot
```

Membuat Fitur untuk Bot

```
# Pesan otomatis
start_message = models.TextField(blank=True,
default="Selamat datang di bot!",
help_text="Pesan yang dikirim saat pengguna
mengetik /start.")
help_message = models.TextField(blank=True,
default="Berikut cara menggunakan bot.",
help_text="Pesan yang dikirim saat pengguna
mengetik /help.")

messages = models.JSONField(default=list,
blank=True, help_text="Daftar pesan yang
diterima dan responsnya dalam format JSON.") #
Format JSON

# Timestamp
created_at =
models.DateTimeField(auto_now_add=True) #
Tanggal bot dibuat
updated_at =
models.DateTimeField(auto_now=True) # Tanggal
terakhir bot diperbarui

name = models.CharField(max_length=255,
blank=True, default="Bot Saya", help_text="Nama
bot.") # Nama bot
description = models.TextField(blank=True,
default="Ini bot pertama saya",
help_text="Deskripsi bot.") # Deskripsi bot

def __str__(self):
    return f'{self.bot_type.capitalize()}
Bot - {self.user.username}'

class Meta:
    verbose_name = "Bot"
    verbose_name_plural = "Bots"
```

Membuat Fitur untuk Bot

```
class BotCommand(models.Model):
    bot = models.ForeignKey(Bot,
related_name='commands',
on_delete=models.CASCADE)
    command = models.CharField(max_length=255,
help_text="Masukkan perintah seperti '/start',
'hello', dll.")
    response =
models.TextField(help_text="Masukkan respons
yang akan dikirim saat perintah diterima.")

    def __str__(self):
        return f"Command: {self.command} ->
Response: {self.response}"

    class Meta:
        unique_together = ['bot', 'command'] #
Pastikan setiap bot hanya memiliki satu perintah
dengan nama yang sama.
        verbose_name = "Bot Command"
        verbose_name_plural = "Bot Commands"
```

12.7.2 Forms.py

```
# File: apps/bots/forms.py

from django import forms
from django.core.exceptions import
ValidationError
import re # Untuk memvalidasi pola token
from .models import Bot, BotCommand # Mengimpor
model Bot dan BotCommand

class BotForm(forms.ModelForm):
    class Meta:
        model = Bot # Menghubungkan form dengan
model Bot
        fields = ['bot_type', 'token_telegram',
'name', 'description', 'is_active',
```

Membuat Fitur untuk Bot

```
'start_message', 'help_message'] # Tambahkan
start_message dan help_message
        widgets = {
            'name':
forms.TextInput(attrs={'class': 'form-control',
'placeholder': 'Masukkan nama bot'}),
            'description':
forms.Textarea(attrs={'class': 'form-control',
'rows': 3, 'placeholder': 'Deskripsi bot'}),
            'start_message':
forms.Textarea(attrs={'class': 'form-control',
'rows': 3, 'placeholder': 'Pesan yang dikirim
saat /start'}),
            'help_message':
forms.Textarea(attrs={'class': 'form-control',
'rows': 3, 'placeholder': 'Pesan yang dikirim
saat /help'}),
        }
        help_texts = {
            'token_telegram': 'Masukkan token
unik yang diterima dari platform Telegram.',
            'name': 'Nama bot dapat diubah
sesuai keinginan.',
            'description': 'Deskripsi singkat
tentang bot dan fungsinya.',
            'start_message': 'Pesan yang dikirim
ketika pengguna mengetik /start.',
            'help_message': 'Pesan yang dikirim
ketika pengguna mengetik /help.',
        }

        # Validasi khusus untuk token Telegram
        def clean_token_telegram(self):
            token_telegram =
self.cleaned_data.get('token_telegram')
            bot_type =
self.cleaned_data.get('bot_type')
```


Membuat Fitur untuk Bot

```
# Validasi hanya jika bot_type adalah
Telegram
    if bot_type == 'telegram':
        # Pola token Telegram: angka-angka
        diikuti dengan titik dua, lalu serangkaian
        karakter acak
        if token_telegram and not
re.match(r'^\d+:[\w-]+$' , token_telegram):
            raise ValidationError("Token
Telegram tidak valid. Pastikan format token
adalah seperti '123456:ABC-DEF1234ghIkl-
zyx57W2v1u123ew11'.")

        return token_telegram

# Validasi nama bot
def clean_name(self):
    name = self.cleaned_data.get('name')
    if not name.strip():
        raise ValidationError("Nama bot
tidak boleh kosong.")
    return name

# Validasi deskripsi bot
def clean_description(self):
    description =
self.cleaned_data.get('description')
    if len(description) > 500:
        raise ValidationError("Deskripsi bot
tidak boleh lebih dari 500 karakter.")
    return description

class BotCommandForm(forms.ModelForm):
    class Meta:
        model = BotCommand
        fields = ['command', 'response'] #
        Field yang akan ditampilkan dalam form
        widgets = {
```

Membuat Fitur untuk Bot

```
        'command':
forms.TextInput(attrs={'placeholder':
'Contoh: /start, /help'}),
        'response':
forms.Textarea(attrs={'rows': 3, 'placeholder':
'Masukkan respon yang akan dikirim.'}),
    }
    help_texts = {
        'command': 'Masukkan perintah
seperti "/start", "/help", atau lainnya.',
        'response': 'Masukkan respons yang
akan dikirim saat perintah diterima.',
    }
```

12.7.3 Views.py

```
# File: apps/bots/views.py

from django.shortcuts import render, redirect,
get_object_or_404
from django.contrib.auth.decorators import
login_required
from django.contrib import messages
from .forms import BotForm, BotCommandForm
from .models import Bot, BotCommand
from django.conf import settings
from apps.webhook.views import
set_telegram_webhook
from apps.webhook.views import
set_whatsapp_webhook
from apps.dashboard.models import ActivityLog

WEBHOOK_URL = settings.WEBHOOK_URL

@login_required
def create_bot(request):
    if request.method == 'POST':
        form = BotForm(request.POST)
        if form.is_valid():
```

Membuat Fitur untuk Bot

```
        bot = form.save(commit=False)
        bot.user = request.user
        bot.is_active = True # Set bot
aktif secara default

        # Tentukan webhook URL sesuai dengan
platform
        if bot.bot_type == 'telegram':
            bot.webhook_url =
f'{WEBHOOK_URL}/telegram/'
        elif bot.bot_type == 'whatsapp':
            bot.webhook_url =
f'{WEBHOOK_URL}/whatsapp/'

        bot.save()

        # Set webhook berdasarkan tipe bot
        if bot.bot_type == 'telegram':
set_telegram_webhook(bot.token_telegram,
bot.webhook_url)
            # Tambahkan pesan sukses untuk
bot Telegram
            messages.success(request, 'Bot
Telegram berhasil dibuat!')
        elif bot.bot_type == 'whatsapp':
set_whatsapp_webhook(bot.webhook_url)
            # Tambahkan pesan sukses untuk
bot WhatsApp
            messages.success(request, f'Bot
WhatsApp berhasil dibuat. Silakan atur URL
webhook berikut di dashboard Twilio:
{bot.webhook_url}')

        # Tambahkan log aktivitas
        ActivityLog.objects.create(
            user=request.user,
```

Membuat Fitur untuk Bot

```
        action=f"Membuat bot baru:
{bot.name}"
    )

    return redirect('manage_bots')
else:
    form = BotForm()

    return render(request,
'bots/create_bot.html', {'form': form})

@login_required # Pastikan hanya pengguna yang
login yang bisa mengakses view ini
def manage_bots(request):
    bots = Bot.objects.filter(user=request.user)
# Ambil daftar bot milik pengguna yang sedang
login

    if request.method == 'POST':
        if 'delete' in request.POST: # Cek
apakah ada permintaan untuk menghapus bot
            bot_id = request.POST.get('delete')
# Ambil ID bot yang ingin dihapus dari form
            bot = get_object_or_404(Bot,
id=bot_id, user=request.user) # Pastikan bot
milik pengguna yang sedang login

            # Menyimpan aktivitas penghapusan ke
dalam log sebelum bot dihapus
            ActivityLog.objects.create(
                user=request.user,
                action=f"Bot {bot.name} telah
dihapus"
            )
            bot.delete() # Hapus bot dari
database
```

Membuat Fitur untuk Bot

```
        messages.success(request, 'Bot
berhasil dihapus.') # Tambahkan pesan setelah
bot dihapus
        return redirect('manage_bots') #
Setelah bot dihapus, kembali ke halaman
manajemen bot

        elif 'delete_command' in request.POST:
            command_id =
request.POST.get('delete_command') #
Mendapatkan ID command yang akan dihapus
            command =
get_object_or_404(BotCommand, id=command_id) #
Mendapatkan command berdasarkan ID

            # Pastikan command milik bot milik
pengguna
            if command.bot.user == request.user:
                # Menyimpan aktivitas
penghapusan command ke dalam log
                ActivityLog.objects.create(
                    user=request.user,
                    action=f"Menghapus command:
{command.command} dari bot {command.bot.name}"
                )
                command.delete() # Menghapus
command dari database
                messages.success(request,
'Command berhasil dihapus.') # Tambahkan pesan
setelah command dihapus
            else:
                messages.error(request, 'Akses
ditolak. Command ini bukan milik Anda.')

            return redirect('manage_bots') #
Mengarahkan kembali ke halaman manage bots
```

Membuat Fitur untuk Bot

```
    return render(request,
'bots/manage_bots.html', {'bots': bots}) #
Render halaman dengan daftar bot yang dimiliki
pengguna

@login_required
def edit_bot(request, bot_id):
    bot = get_object_or_404(Bot, id=bot_id) #
Ambil bot berdasarkan ID
    if request.method == 'POST':
        form = BotForm(request.POST,
instance=bot) # Muat data ke form
        if form.is_valid():
            form.save() # Simpan data yang
telah diedit

            # Tambahkan log aktivitas setelah
bot berhasil diedit
            ActivityLog.objects.create(
                user=request.user,
                action=f"Bot {bot.name} telah
diperbarui",
                bot=bot
            )

            return redirect('manage_bots') #
Alihkan ke halaman pengelolaan bot
        else:
            form = BotForm(instance=bot) # Jika
bukan POST, buat form dengan data bot

            context = {
                'form': form,
                'bot': bot,
            }
            return render(request, 'bots/edit_bot.html',
context) # Tampilkan template dengan context
```

Membuat Fitur untuk Bot

```
@login_required
def add_command(request, bot_id):
    bot = get_object_or_404(Bot, id=bot_id,
user=request.user) # Mendapatkan bot
berdasarkan ID dan memastikan user adalah
pemilik bot

    if request.method == 'POST':
        form = BotCommandForm(request.POST) #
Membuat form berdasarkan data POST
        if form.is_valid():
            command = form.save(commit=False) #
Simpan command baru tapi belum ke database
            command.bot = bot # Menetapkan bot
terkait dengan command
            command.save() # Menyimpan command
ke database

            # Mencatat aktivitas penambahan
command ke dalam log
            ActivityLog.objects.create(
                user=request.user,
                action=f"Menambahkan command
baru: {command.command} untuk bot {bot.name}"
            )

            return redirect('manage_bots') #
Setelah berhasil menyimpan, redirect ke halaman
edit bot
        else:
            form = BotCommandForm() # Jika GET,
tampilkan form kosong

            return render(request,
'bots/add_command.html', {'form': form, 'bot':
bot}) # Render template dengan form

@login_required
```

Membuat Fitur untuk Bot

```
def edit_command(request, command_id):
    command = get_object_or_404(BotCommand,
                                id=command_id) # Ambil command berdasarkan ID
    bot = command.bot # Mendapatkan bot terkait
    dengan command

    if request.method == 'POST':
        form = BotCommandForm(request.POST,
                                instance=command) # Muat data POST ke dalam
        form
        if form.is_valid():
            form.save() # Simpan perubahan
            command ke dalam database

            # Catat aktivitas pengeditan command
            ke dalam log
            ActivityLog.objects.create(
                user=request.user,
                action=f"Mengedit command:
{command.command} di bot {bot.name}"
            )

            return redirect('manage_bots') #
Setelah berhasil menyimpan, alihkan kembali ke
halaman edit bot
        else:
            form = BotCommandForm(instance=command)
# Jika bukan POST, tampilkan form dengan data
command yang ada

            return render(request,
                            'bots/edit_command.html', {'form': form, 'bot':
bot}) # Render template dengan form dan bot
terkait
```

12.7.4 Services/telegram.py

```
# File: apps/bots/services/telegram.py
```


Membuat Fitur untuk Bot

```
from apps.bots.models import Bot, BotCommand
import requests

def handle_update(update):
    chat_id = update['message']['chat']['id']
    text = update['message']['text'].lower() #
    Mengubah ke lowercase untuk konsistensi
    perbandingan

    bot =
    Bot.objects.filter(bot_type='telegram').first()
    # Misalkan kita ambil bot pertama, bisa
    disesuaikan

    if bot:
        # Cek apakah perintah cocok dengan yang
        ada di model BotCommand
        command =
        BotCommand.objects.filter(bot=bot,
        command__iexact=text).first()

        if command:
            response_text = command.response
        else:
            # Jika perintah tidak ditemukan,
            gunakan perintah default /start dan /help
            if text == "/start":
                response_text =
            bot.start_message
            elif text == "/help":
                response_text = bot.help_message
            else:
                response_text = 'Perintah tidak
                dikenal. Ketik /help untuk melihat daftar
                perintah.'

        send_message(bot.token, "sendMessage", {
```

Membuat Fitur untuk Bot

```
        'chat_id': chat_id,
        'text': response_text
    })

def send_message(token, method, data):
    telegram_api_url =
f'https://api.telegram.org/bot{token}/{method}'
    response = requests.post(telegram_api_url,
data=data)
    if response.status_code != 200:
        logger.error(f"Failed to send message:
{response.text}") # Log error jika gagal
    return response
```

12.7.5 Services/whatsapp.py

```
# File: apps/bots/services/whatsapp.py

from django.http import HttpResponse # Untuk
mengembalikan respon HTTP
from django.views.decorators.csrf import
csrf_exempt # Agar view dapat diakses tanpa
validasi CSRF
from twilio.twiml.messaging_response import
MessagingResponse # Untuk membuat balasan ke
Twilio API
from apps.bots.models import Bot, BotCommand #
Mengimpor model Bot dan BotCommand dari aplikasi
bots
import logging # Untuk logging aktivitas dan
error

logger = logging.getLogger(__name__) #
Inisialisasi logger

@csrf_exempt # Dekorator untuk menonaktifkan
validasi CSRF pada view ini
def handle_update(update):
```

Membuat Fitur untuk Bot

```
# Mendapatkan pengirim pesan dan isi pesan
user = update['From']
message = update['Body'].strip().lower() #
Mengubah pesan ke lowercase untuk perbandingan

# Logging pesan yang diterima
logger.info(f'{user} says {message}')

# Mengambil bot pertama yang bertipe
WhatsApp dari database
bot =
Bot.objects.filter(bot_type='whatsapp').first()
response = MessagingResponse() # Membuat
respon menggunakan Twilio's MessagingResponse

if bot: # Jika bot ditemukan
    # Cek apakah perintah cocok dengan yang
    ada di model BotCommand
    command =
    BotCommand.objects.filter(bot=bot,
    command__iexact=message).first()

    if command:
        response_text = command.response #
    Dapatkan respons dari perintah
    else:
        # Jika perintah tidak ditemukan,
        gunakan perintah default /start dan /help
        if message == "/start":
            response_text =
            bot.start_message # Pesan sambutan
        elif message == "/help":
            response_text = bot.help_message
        # Pesan bantuan
        else:
            response_text = 'Perintah tidak
            dikenal. Ketik /help untuk melihat daftar
            perintah.' # Pesan default
```

Membuat Fitur untuk Bot

```
        response.message(response_text) #  
Mengirim balasan pesan  
    else:  
        response.message('Bot tidak ditemukan.')  
# Jika bot tidak ditemukan di database  
  
# Mengembalikan respon sebagai HTTP response  
return HttpResponse(str(response))
```

12.7.6 URLS.py

```
# File: apps/bots/urls.py  
  
from django.urls import path  
from . import views  
  
urlpatterns = [  
    path('create/', views.create_bot,  
name='create_bot'), # URL untuk halaman  
pembuatan bot  
    path('manage/', views.manage_bots,  
name='manage_bots'), # URL untuk halaman  
manajemen bot  
    path('edit_bot/<int:bot_id>/',  
views.edit_bot, name='edit_bot'), # Menggunakan  
bot_id sebagai parameter  
    path('add_command/<int:bot_id>/',  
views.add_command, name='add_command'), # URL  
untuk menambah command  
    path('edit_command/<int:command_id>/',  
views.edit_command, name='edit_command'), # URL  
untuk edit command  
]
```

Dengan kode lengkap ini di harapkan tidak ada kesalahan dalam kode.

Kesimpulan Bab

Pada Bab ini kita membahas secara mendalam tentang fitur command pada bot dalam proyek platform_bot. Command pada bot adalah perintah khusus yang diberikan oleh pengguna untuk memicu respons atau tindakan tertentu dari bot. Fitur ini merupakan elemen penting dalam interaksi bot, memungkinkan bot untuk merespons berbagai input dari pengguna dengan cara yang terstruktur dan efisien.

Dalam bagian ini, kita telah mengimplementasikan dua fitur utama:

1. **Menambah Command:** Fitur ini memungkinkan pengguna untuk menambahkan command baru ke bot mereka. Kami membuat model untuk menyimpan data command, formulir untuk menginput command baru, dan view serta template untuk menangani proses penambahan command. Dengan fitur ini, pengguna dapat memperluas fungsionalitas bot mereka dengan perintah baru yang sesuai dengan kebutuhan mereka.
2. **Mengedit Command:** Fitur ini memungkinkan pengguna untuk mengubah command yang sudah ada. Kami telah membuat view dan template untuk proses pengeditan command, serta mengatur URL routing untuk memastikan bahwa pengguna dapat mengakses halaman pengeditan dengan mudah. Pengeditan command memberikan fleksibilitas tambahan dalam mengelola command yang sudah ada, memungkinkan penyesuaian respons bot sesuai dengan perubahan kebutuhan.

Kunci Sukses dalam Menambah dan Mengedit Command

Keberhasilan dalam menambahkan dan mengedit command pada bot sangat bergantung pada beberapa faktor kunci:

1. ***Desain Model yang Efisien:*** Model BotCommand yang dirancang dengan baik memastikan bahwa data command disimpan dengan benar dan terstruktur. Dengan menggunakan atribut `unique_together`, kita mencegah duplikasi command untuk bot yang sama, menjaga integritas data.
2. ***Formulir yang User-Friendly:*** BotCommandForm dirancang untuk memudahkan pengguna dalam memasukkan dan mengedit command. Dengan menggunakan widget seperti `TextInput` dan `Textarea`, serta menambahkan teks bantuan yang jelas, formulir ini memberikan pengalaman pengguna yang intuitif.
3. ***Implementasi View yang Tepat:*** View untuk menambah dan mengedit command memastikan bahwa data dari formulir diproses dengan benar. Dengan menangani request POST untuk menyimpan data dan request GET untuk menampilkan formulir, view ini memfasilitasi interaksi pengguna dengan sistem.
4. ***Templat HTML yang Jelas:*** Templat `add_command.html` dan `edit_command.html` dirancang untuk menyediakan antarmuka pengguna yang bersih dan mudah dipahami. Dengan penggunaan elemen form yang jelas dan pengaturan layout yang baik, templat ini meningkatkan pengalaman pengguna saat menambah atau mengedit command.

Membuat Fitur untuk Bot

5. ***URL Routing yang Konsisten:*** Pengaturan URL routing yang tepat memastikan bahwa pengguna dapat mengakses fitur tambah dan edit command dengan mudah. Routing yang konsisten dan intuitif membantu menjaga navigasi aplikasi tetap sederhana dan mudah diakses.

Dengan mengimplementasikan fitur-fitur ini dengan benar, Anda memastikan bahwa bot dapat berfungsi dengan lebih baik dan responsif terhadap kebutuhan pengguna. Fitur command yang efektif dan mudah dikelola adalah kunci untuk meningkatkan fungsionalitas dan kegunaan bot dalam platform Anda.

BAB 13 -Penyempurnaan Halaman Home

Dalam bab ini, kita akan fokus pada penyempurnaan halaman home dari proyek platform_bot kita dengan menggunakan Bootstrap. Bootstrap adalah framework CSS yang sangat populer dan banyak digunakan dalam pengembangan web. Framework ini menawarkan berbagai komponen dan utilitas yang memudahkan pembuatan desain web yang responsif dan menarik. Tujuan dari bab ini adalah untuk memanfaatkan fitur-fitur Bootstrap untuk meningkatkan tampilan dan pengalaman pengguna dari platform_bot, menjadikannya lebih profesional dan mudah digunakan.

13.1 *Pendahuluan*

13.1.1 Tujuan dan Manfaat Penggunaan Bootstrap

Bootstrap adalah alat yang sangat berharga dalam pengembangan web karena beberapa alasan. Pertama, Bootstrap menyediakan berbagai komponen UI yang siap pakai, seperti tombol, formulir, tabel, dan navigasi, yang dapat membantu mempercepat proses pengembangan. Kedua, dengan sistem grid dan kelas utilitasnya, Bootstrap memungkinkan pembuatan desain responsif yang otomatis menyesuaikan dengan berbagai ukuran layar, dari desktop hingga perangkat mobile. Hal ini penting untuk memastikan bahwa platform_bot dapat diakses dan digunakan dengan nyaman di

Penyempurnaan Halaman Home

berbagai perangkat. Ketiga, Bootstrap menawarkan konsistensi visual yang dapat membantu menjaga tampilan antarmuka pengguna tetap seragam di seluruh aplikasi.

13.1.2 Pengenalan Dasar Bootstrap

Bootstrap dikembangkan oleh Twitter dan dirilis sebagai proyek open-source. Framework ini menawarkan beberapa komponen dasar yang dapat digunakan langsung dalam proyek kita.

Berikut adalah beberapa fitur utama dari Bootstrap:

1. **Grid System:** Sistem grid Bootstrap memungkinkan kita untuk membuat layout responsif dengan menggunakan kelas-kelas yang sudah ditentukan. Grid ini membagi halaman menjadi kolom-kolom yang dapat diatur untuk berbagai ukuran layar.
2. **Komponen UI:** Bootstrap menyediakan berbagai komponen seperti tombol, formulir, tabel, alert, modal, dan navigasi, yang dapat digunakan untuk mempercepat proses pembuatan antarmuka pengguna.
3. **Utilitas:** Bootstrap menawarkan kelas utilitas untuk melakukan berbagai penyesuaian gaya dengan cepat, seperti margin, padding, warna, dan visibilitas elemen.
4. **Kustomisasi:** Bootstrap juga mendukung kustomisasi tema dengan menyediakan file SASS yang memungkinkan kita untuk mengubah warna, ukuran, dan gaya sesuai dengan kebutuhan proyek.

13.1.3 Instalasi dan Konfigurasi Bootstrap di Proyek Django

Untuk mulai menggunakan Bootstrap di proyek Django kita, pertama-tama kita perlu menginstallnya dan mengonfigurasinya. Berikut adalah langkah-langkah untuk melakukannya:

Menginstal Bootstrap melalui CDN

Cara termudah untuk mengintegrasikan Bootstrap ke dalam proyek kita adalah dengan menggunakan Content Delivery Network (CDN). Kita hanya perlu menambahkan link ke file CSS dan JavaScript Bootstrap di template HTML kita.

Tambahkan kode berikut di bagian `<head>` dari template `base.html`:

```
<!-- Link ke CSS Bootstrap -->
<link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">

<!-- Link ke JavaScript Bootstrap -->
<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
<script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.3/dist/umd/popper.min.js"></script>
<script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
```

Penyempurnaan Halaman Home

Menginstal Bootstrap melalui Paket NPM

Jika lebih suka mengelola dependensi menggunakan NPM, kita bisa menginstal Bootstrap dengan perintah berikut. Pertama, pastikan Anda berada di direktori proyek Django Anda, lalu jalankan:

```
$ npm install bootstrap
```

Setelah itu, tambahkan Bootstrap ke file CSS dan JavaScript proyek Anda. Misalnya, tambahkan ke `static/css/styles.css` untuk CSS dan `static/js/scripts.js` untuk JavaScript:

```
/* Menambahkan Bootstrap ke file CSS */  
@import "~bootstrap/dist/css/bootstrap.min.css";
```

```
// Menambahkan Bootstrap ke file JavaScript  
import 'bootstrap';
```

Mengonfigurasi Template HTML

Setelah Bootstrap terinstal, kita perlu mengupdate template HTML kita untuk menggunakan komponen dan utilitas Bootstrap. Mari kita lihat beberapa contoh penggunaan Bootstrap pada halaman yang sudah ada:

Halaman Home (`home.html`)

Tambahkan kelas-kelas Bootstrap ke elemen HTML untuk membuat layout yang responsif dan menarik. Misalnya, gunakan kelas

Penyempurnaan Halaman Home

`container` untuk membungkus konten dan kelas `row` serta `col` untuk mengatur kolom:

```
<div class="container">
  <div class="row">
    <div class="col-md-6">
      <h1>Selamat Datang di Platform Bot</h1>
      <p>Platform Bot kami menawarkan berbagai
fitur untuk mengelola bot Telegram Anda.</p>
    </div>
    <div class="col-md-6">
      
    </div>
  </div>
</div>
```

Dengan menggunakan Bootstrap, kita dapat dengan mudah memperbaiki dan mempercantik antarmuka pengguna platform_bot kita. Selanjutnya, kita akan memanfaatkan Bootstrap untuk lebih lanjut menyempurnakan halaman-halaman lainnya dan memastikan bahwa setiap bagian dari aplikasi terlihat profesional dan berfungsi dengan baik di berbagai perangkat.

13.2 Penyempurnaan Halaman Home

Saat ini, halaman home dari platform_bot kita terdiri dari file `base.html` dan `home.html`. File `base.html` berfungsi sebagai template dasar yang mencakup elemen-elemen umum seperti navbar, footer, dan link ke file CSS dan JavaScript Bootstrap. File ini menetapkan struktur dasar halaman dan memastikan konsistensi desain di seluruh halaman aplikasi kita.

Penyempurnaan Halaman Home

File `home.html` mengextends `base.html` dan menyediakan konten spesifik untuk halaman utama, seperti judul sambutan dan deskripsi aplikasi. Namun, tampilan saat ini masih sederhana dan belum sepenuhnya memanfaatkan fitur Bootstrap untuk menciptakan desain yang responsif dan menarik. Pada bagian ini, kita akan memanfaatkan berbagai fitur Bootstrap untuk memperbaiki tampilan halaman home kita.

13.2.1 Perbaiki base.html

Kita perbaiki `base.html` dengan menambahkan lebih banyak fitur dari Bootstrap dan ikon dari Bootstrap Icons untuk meningkatkan tampilan dan fungsionalitas halaman. Kita akan memperbarui file `base.html` untuk memanfaatkan komponen Bootstrap lebih efektif dan menambahkan beberapa elemen visual tambahan.

Di sini, kita akan mengubah warna navbar menjadi warna primer, menambahkan latar belakang pada beberapa bagian, dan menyesuaikan footer.

Berikut adalah pembaruan untuk file `base.html`:

```
<!-- File: apps/templates/home/base.html -->

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-
width, initial-scale=1.0">
```

Penyempurnaan Halaman Home

```
<title>Platform Bot</title>
<!-- Link ke Bootstrap CSS -->
<link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3
.3/dist/css/bootstrap.min.css" rel="stylesheet"
integrity="sha384-
QWTKZyjpPEjISv5WaRU90FeRpok6YctnYmDr5pNlyT2bRjXh
0JMHjY6hw+ALEwIH" crossorigin="anonymous">
<!-- Link ke Bootstrap Icons -->
<link
href="https://cdn.jsdelivr.net/npm/bootstrap-
icons@1.10.5/font/bootstrap-icons.css"
rel="stylesheet">
</head>
<body class="d-flex flex-column min-vh-100">
  <!-- Header -->
  <nav class="navbar navbar-expand-lg navbar-
dark bg-primary shadow-sm">
    <div class="container">
      <a class="navbar-brand d-flex align-
items-center" href="{% url 'home' %}">
        <i class="bi bi-lightning-
charge-fill me-2"></i>Platform Bot
      </a>
      <button class="navbar-toggler"
type="button" data-bs-toggle="collapse" data-bs-
target="#navbarNav" aria-controls="navbarNav"
aria-expanded="false" aria-label="Toggle
navigation">
        <span class="navbar-toggler-
icon"></span>
      </button>
      <div class="collapse navbar-
collapse" id="navbarNav">
        <ul class="navbar-nav ms-auto
mb-2 mb-lg-0">
          <li class="nav-item">
```

Penyempurnaan Halaman Home

```


```

Penyempurnaan Halaman Home

```
</div>

<!-- Footer -->
<footer class="footer mt-auto py-4 bg-dark
text-white-50">
  <div class="container text-center">
    <span>© 2024 Platform Bot</span>
    <div class="mt-3">
      <a href="#" class="btn btn-
outline-light btn-sm me-2">
        <i class="bi bi-
facebook"></i>
      </a>
      <a href="#" class="btn btn-
outline-light btn-sm me-2">
        <i class="bi bi-
twitter"></i>
      </a>
      <a href="#" class="btn btn-
outline-light btn-sm">
        <i class="bi bi-
instagram"></i>
      </a>
    </div>
  </div>
</footer>

<!-- Bootstrap JS dan dependensi -->
<script
src="https://cdn.jsdelivr.net/npm/@popperjs/core
@2.11.8/dist/umd/popper.min.js"
integrity="sha384-I7E8VVD/ismYTF4hNIPjVp/Zjvgyol
6VFvRkX/vR+Vc4jQkC+hVqc2pM80Dewa9r"
crossorigin="anonymous"></script>
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.
3/dist/js/bootstrap.min.js" integrity="sha384-
0pUGZvbkm6XF6gxjEnlmuGrJXVbNuzT9qBBavbLwCs0GabYf
```


Penyempurnaan Halaman Home

```
Zo0T0to5eqruptLy"  
crossorigin="anonymous"></script>  
</body>  
</html>
```

Penjelasan Perubahan:

1. **Navbar Responsif:** Menambahkan tombol toggler pada navbar untuk tampilan responsif di perangkat mobile. Ini memungkinkan menu navbar beralih antara tampilan vertikal dan horizontal pada berbagai ukuran layar.
2. **Form Pencarian:** Menambahkan form pencarian di navbar dengan ikon pencarian dari Bootstrap Icons. Form ini memberikan pengguna opsi untuk mencari konten di situs.
3. **Ikon Media Sosial:** Menambahkan ikon media sosial di footer menggunakan tombol dengan ikon dari Bootstrap Icons. Ini meningkatkan keterhubungan sosial situs web.
4. **Penggunaan Container:** Menggunakan kelas `container` untuk menyelaraskan elemen-elemen dalam navbar dan footer sehingga tampil lebih rapi dan konsisten di berbagai perangkat.
5. **Responsif:** Memastikan bahwa semua elemen seperti navbar dan footer tetap responsif dan tampil dengan baik di berbagai ukuran layar.

Dengan pembaruan ini, `base.html` menjadi lebih fungsional dan estetik, memanfaatkan fitur Bootstrap untuk desain responsif dan ikon yang memperkaya tampilan. Selanjutnya, pastikan untuk menguji tampilan di berbagai perangkat dan menyesuaikan CSS tambahan jika diperlukan.

13.2.2 Menambahkan Komponen Bootstrap

Untuk lebih mempercantik tampilan halaman home, kita akan memanfaatkan berbagai komponen Bootstrap, seperti Navbar, Jumbotron, dan Card.

Navbar sudah ditambahkan dalam `base.html`, memberikan navigasi yang jelas kepada pengguna. Di halaman home, kita menggunakan Jumbotron untuk menampilkan pesan sambutan dengan desain yang mencolok. Selain itu, kita dapat menambahkan komponen Card untuk menampilkan informasi tambahan dengan gaya yang terstruktur.

Berikut adalah contoh penggunaan komponen Card:

```
<!-- apps/templates/home/home.html -->

{% extends 'home/base.html' %}

{% block content %}
<div class="jumbotron bg-primary text-white
rounded p-4 mb-4 text-center">
  <h1 class="display-4">Selamat Datang di
Platform Bot!</h1>
  <p class="lead">Temukan solusi otomatisasi
terbaik untuk kebutuhan bot Anda. Platform kami
menawarkan berbagai fitur canggih untuk
mempermudah pekerjaan Anda.</p>
  <hr class="my-4">
  <p>Mulai dengan mendaftar atau login untuk
akses penuh ke fitur-fitur hebat kami.</p>
  <a class="btn btn-light btn-lg"
href="{% url 'register' %}" role="button">Mulai
Sekarang</a>
</div>
{% endblock %}
```

Penyempurnaan Halaman Home

```
<p class="mt-4">Sudah punya akun? <a
class="text-muted" href="{% url 'login'
%}">Masuk di sini</a></p> <!-- Tautan ke halaman
Login -->
</div>

<!-- Card Section -->
<div class="container mb-5">
  <div class="row">
    <div class="col-md-4 mb-4">
      <div class="card shadow-sm">
        <div class="card-body text-
center">
          <i class="bi bi-robot fs-2
mb-3"></i>
          <h5 class="card-
title">Automatisasi Bot</h5>
          <p class="card-
text">Manfaatkan kemampuan otomatisasi kami
untuk mengoptimalkan tugas-tugas rutin dan
proses kerja Anda dengan bot yang mudah
disesuaikan.</p>
          <a href="#" class="btn btn-
primary">Pelajari Lebih Lanjut</a>
        </div>
      </div>
    </div>
    <div class="col-md-4 mb-4">
      <div class="card shadow-sm">
        <div class="card-body text-
center">
          <i class="bi bi-gear fs-2
mb-3"></i>
          <h5 class="card-
title">Pengaturan Fleksibel</h5>
          <p class="card-
text">Sesuaikan setiap aspek bot Anda dengan
pengaturan yang fleksibel dan antarmuka yang
intuitif. Kontrol penuh ada di tangan Anda.</p>
```

Penyempurnaan Halaman Home

```

        <a href="#" class="btn btn-
primary">Pelajari Lebih Lanjut</a>
    </div>
</div>
</div>
<div class="col-md-4 mb-4">
    <div class="card shadow-sm">
        <div class="card-body text-
center">
            <i class="bi bi-bar-chart-
line fs-2 mb-3"></i>
            <h5 class="card-
title">Analitik Mendalam</h5>
            <p class="card-
text">Dapatkan wawasan mendalam dengan fitur
analitik kami. Pantau performa bot Anda dan
ambil keputusan yang lebih baik berdasarkan
data.</p>
            <a href="#" class="btn btn-
primary">Pelajari Lebih Lanjut</a>
        </div>
    </div>
</div>
</div>
</div>
{% endblock %}
```

13.2.3 Menyempurnakan Tampilan Responsif

Dengan menggunakan kelas-kelas Bootstrap, halaman home kita sudah menjadi responsif, tetapi kita dapat melakukan beberapa penyesuaian tambahan untuk memastikan tampilan yang optimal di berbagai perangkat. Pastikan bahwa elemen-elemen seperti gambar, card, dan tombol berfungsi dengan baik di layar kecil dan besar.

Penyempurnaan Halaman Home

Misalnya, pada bagian `card` yang telah ditambahkan, kita memastikan bahwa ukuran gambar dan teks dapat menyesuaikan dengan ukuran layar. Untuk gambar, kita menggunakan kelas `img-fluid` untuk memastikan gambar responsif dan tidak meluber dari kontainer.

Penjelasan Perubahan:

1. **Navbar:** Mengubah warna navbar menjadi `bg-primary` dengan teks putih untuk kontras yang lebih baik. Menambahkan form pencarian dengan tombol berwarna terang agar lebih menonjol.
2. **Footer:** Mengubah warna latar belakang footer menjadi `bg-secondary` dengan teks putih dan menambahkan ikon media sosial dengan warna terang agar sesuai dengan skema warna.
3. **Halaman Home:**
 - Menggunakan latar belakang `bg-info` dengan teks putih pada jumbotron untuk menonjolkan sambutan dengan desain yang menarik.
 - Menambahkan bagian `card` untuk menampilkan fitur-fitur utama dengan gambar, judul, dan deskripsi singkat. Ini memberikan tampilan yang lebih dinamis dan informatif.

Dengan pembaruan ini, tampilan `base.html` dan `home.html` menjadi lebih berwarna, informatif, dan menarik secara visual, sambil tetap mempertahankan desain responsif dan fungsional.

Penyempurnaan Halaman Home

13.2.4 Menambahkan Konten di Halaman Home

Kita tambahkan lebih banyak konten ke halaman `home.html` dengan mengganti latar belakang jumbotron menjadi `bg-primary`. Kita akan menambahkan beberapa seksi tambahan seperti testimonial, berita terbaru, dan call-to-action. Ini akan membuat halaman utama lebih informatif dan menarik.

Berikut adalah pembaruan untuk `home.html` yang mencakup seksi tambahan untuk testimonial, berita terbaru, dan call-to-action. Kita juga akan mengubah latar belakang jumbotron menjadi `bg-primary`.

```
<!-- File: apps/templates/home/home.html -->

{% extends 'home/base.html' %}

{% block content %}
<!-- Hero Section -->
<div class="jumbotron bg-primary text-white
rounded p-4 mb-5 text-center">
    <h1 class="display-4"><i class="bi bi-
lightbulb-fill me-2"></i>Selamat Datang di
Platform Bot!</h1>
    <p class="lead">Temukan solusi otomatisasi
terbaik untuk kebutuhan bot Anda. Platform kami
menawarkan berbagai fitur canggih untuk
mempermudah pekerjaan Anda.</p>
    <hr class="my-4">
    <p>Mulai dengan mendaftar atau login untuk
akses penuh ke fitur-fitur hebat kami.</p>
</div>
```

Penyempurnaan Halaman Home

```
<a class="btn btn-light btn-lg mb-3"
href="{% url 'register' %}" role="button">Mulai
Sekarang</a>
<p class="mt-3">Sudah punya akun? <a
class="text-light" href="{% url 'login'
%}">Masuk di sini</a></p>
</div>

<!-- Feature Section -->
<div class="container mb-5">
  <h2 class="text-center mb-4">Fitur Unggulan
Kami</h2>
  <div class="row">
    <div class="col-md-4 mb-4">
      <div class="card border-0 shadow-sm
bg-light h-100 text-center">
        <div class="card-body">
          <i class="bi bi-robot fs-1
text-primary mb-3"></i>
          <h5 class="card-
title">Automatisasi Bot</h5>
          <p class="card-
text">Manfaatkan kemampuan otomatisasi kami
untuk mengoptimalkan tugas-tugas rutin dan
proses kerja Anda dengan bot yang mudah
disesuaikan.</p>
          <a href="#" class="btn btn-
primary">Pelajari Lebih Lanjut</a>
        </div>
      </div>
    <div class="col-md-4 mb-4">
      <div class="card border-0 shadow-sm
bg-light h-100 text-center">
        <div class="card-body">
          <i class="bi bi-gear fs-1
text-primary mb-3"></i>
          <h5 class="card-
title">Pengaturan Fleksibel</h5>
```

Penyempurnaan Halaman Home

```

                <p class="card-
text">Sesuaikan setiap aspek bot Anda dengan
pengaturan yang fleksibel dan antarmuka yang
intuitif. Kontrol penuh ada di tangan Anda.</p>
                <a href="#" class="btn btn-
primary">Pelajari Lebih Lanjut</a>
            </div>
        </div>
    </div>
    <div class="col-md-4 mb-4">
        <div class="card border-0 shadow-sm
bg-light h-100 text-center">
            <div class="card-body">
                <i class="bi bi-bar-chart-
line fs-1 text-primary mb-3"></i>
                <h5 class="card-
title">Analitik Mendalam</h5>
                <p class="card-
text">Dapatkan wawasan mendalam dengan fitur
analitik kami. Pantau performa bot Anda dan
ambil keputusan yang lebih baik berdasarkan
data.</p>
                <a href="#" class="btn btn-
primary">Pelajari Lebih Lanjut</a>
            </div>
        </div>
    </div>
</div>

<!-- Testimonial Section -->
<div class="container mb-5">
    <h2 class="text-center mb-4">Apa Kata
Pengguna Kami</h2>
    <div class="row">
        <div class="col-md-4 mb-4">
            <div class="card border-light
shadow-sm bg-white h-100">
                <div class="card-body">

```


Penyempurnaan Halaman Home

```

        <i class="bi bi-people-fill
fs-2 text-primary mb-3"></i>
        <p class="card-
text">"Platform Bot telah mengubah cara kami
bekerja. Fitur-fitur canggih dan antarmuka yang
ramah pengguna membuat pekerjaan kami jauh lebih
efisien."</p>
        <footer class="blockquote-
footer">Jane Doe, <cite title="Source Title">CEO
ExampleCorp</cite></footer>
    </div>
</div>
<div class="col-md-4 mb-4">
    <div class="card border-light
shadow-sm bg-white h-100">
        <div class="card-body">
            <i class="bi bi-chat-left-
quote fs-2 text-primary mb-3"></i>
            <p class="card-text">"Saya
sangat terkesan dengan dukungan pelanggan dan
fungsionalitas yang ditawarkan. Platform ini
sangat membantu dalam otomatisasi tugas sehari-
hari."</p>
            <footer class="blockquote-
footer">John Smith, <cite title="Source
Title">Freelancer</cite></footer>
        </div>
    </div>
</div>
<div class="col-md-4 mb-4">
    <div class="card border-light
shadow-sm bg-white h-100">
        <div class="card-body">
            <i class="bi bi-star-fill
fs-2 text-primary mb-3"></i>
            <p class="card-text">"Sangat
mudah untuk mengintegrasikan dengan sistem yang
```

Penyempurnaan Halaman Home

```
sudah ada. Platform ini benar-benar memenuhi
kebutuhan kami."</p>
    <footer class="blockquote-
footer">Alice Johnson, <cite title="Source
Title">Product Manager</cite></footer>
    </div>
</div>
</div>
</div>
</div>
</div>

<!-- Latest News Section -->
<div class="container mb-5">
    <h2 class="text-center mb-4">Berita
Terbaru</h2>
    <div class="row text-center">
        <div class="col-md-6 mb-4">
            <div class="card border-0 shadow-sm
bg-info text-white h-100">
                <div class="card-body">
                    <i class="bi bi-info-circle
fs-2 mb-3"></i>
                    <h5 class="card-
title">Update Terbaru: Fitur X Dirilis</h5>
                    <p class="card-text">Kami
baru saja meluncurkan fitur X yang akan
mempermudah proses Y. Baca lebih lanjut tentang
apa yang baru di versi ini.</p>
                    <a href="#" class="btn btn-
light">Baca Selengkapnya</a>
                </div>
            </div>
        </div>
        <div class="col-md-6 mb-4">
            <div class="card border-0 shadow-sm
bg-warning text-dark h-100">
                <div class="card-body">
                    <i class="bi bi-file-
earmark-text fs-2 mb-3"></i>
```

Penyempurnaan Halaman Home

```
                <h5 class="card-  
title">Panduan Lengkap: Cara Menggunakan  
Platform Bot</h5>  
                <p class="card-text">Ikuti  
panduan kami untuk memanfaatkan semua fitur  
Platform Bot dengan maksimal. Dapatkan tips dan  
trik dari para ahli kami.</p>  
                <a href="#" class="btn btn-  
dark">Pelajari Lebih Lanjut</a>  
            </div>  
        </div>  
    </div>  
</div>  
  
<!-- Call to Action Section -->  
<div class="bg-success text-white p-5 text-  
center">  
    <h2>Siap untuk Memulai?</h2>  
    <p class="lead">Daftar sekarang untuk  
mendapatkan akses penuh ke semua fitur hebat  
kami dan mulai meningkatkan produktivitas Anda  
hari ini!</p>  
    <a class="btn btn-light btn-lg" href="{% url  
'register' %}" role="button">Daftar Sekarang</a>  
</div>  
{% endblock %}
```

Penyesuaian yang dilakukan:

1. **Hero Section:** Menambahkan lebih banyak warna pada latar belakang dan teks. Teks tautan untuk login kini berwarna terang.
2. **Feature Section:** Menggunakan ikon yang lebih besar dan warna latar belakang yang lebih ringan agar fitur lebih menonjol.

Penyempurnaan Halaman Home

3. **Testimonial Section:** Menambahkan ikon pada setiap testimonial untuk memperkaya visualisasi dan menjaga tampilan tetap modern.
4. **News Section:** Menggunakan kartu berwarna untuk menampilkan berita terbaru, menambahkan kontras antara berita untuk menarik perhatian.
5. **Call to Action Section:** Menggunakan warna latar belakang yang berbeda (hijau) untuk menciptakan kesan mendesak dan menarik pengguna.

Dengan perubahan ini, halaman utama akan memiliki lebih banyak konten yang bermanfaat dan visual yang lebih menarik, membuat pengalaman pengguna lebih informatif dan menarik.

13.2.5 Menguji Tampilan Halaman Home

Setelah melakukan pembaruan, langkah terakhir adalah menguji tampilan halaman home di berbagai perangkat untuk memastikan bahwa semuanya berfungsi seperti yang diharapkan. Periksa tampilan di berbagai ukuran layar, baik menggunakan alat pengembang di browser atau dengan perangkat fisik.

Untuk menjalankan server Django dan melihat hasilnya, gunakan perintah berikut:

```
$ python manage.py runserver
```

Buka browser dan arahkan ke `http://127.0.0.1:8000/` untuk melihat halaman home. Pastikan semua elemen, termasuk navbar, jumbotron, dan card, tampil dengan baik dan responsif.

Penyempurnaan Halaman Home

Lakukan penyesuaian tambahan jika diperlukan untuk memastikan pengalaman pengguna yang optimal di seluruh perangkat.

Dengan langkah-langkah ini, halaman home dari platform_bot telah diperbarui dengan tampilan yang lebih profesional dan responsif, berkat penerapan fitur-fitur Bootstrap. Selanjutnya, kita akan menerapkan teknik serupa pada halaman-halaman lain untuk memastikan konsistensi desain di seluruh aplikasi.

13.3 Penyempurnaan Halaman Register

Saat ini, halaman register Anda menggunakan struktur HTML dasar dengan beberapa elemen formulir. Namun, tampilan dan responsifitas halaman ini dapat ditingkatkan dengan menerapkan sistem grid dari Bootstrap serta menambahkan komponen-komponen Bootstrap untuk meningkatkan penampilan dan fungsionalitas.

13.3.1 Mengintegrasikan Bootstrap Grid System

Bootstrap Grid System memudahkan pembuatan tata letak responsif yang fleksibel. Pada halaman register, kita akan menggunakan grid untuk memastikan formulir dapat menyesuaikan dengan berbagai ukuran layar.

Untuk mengintegrasikan sistem grid, perhatikan struktur yang ada di dalam `<div class="container">`. Kita akan

Penyempurnaan Halaman Home

mengoptimalkan penggunaan kolom dengan menambahkan kelas-kelas grid Bootstrap.

13.3.2 Menambahkan Komponen Bootstrap

Kita akan memperbarui halaman register untuk menggunakan komponen Bootstrap seperti `card`, `form-group`, dan `input-group` agar tampilan formulir menjadi lebih modern dan responsif.

Untuk meningkatkan tampilan halaman registrasi ini agar lebih profesional, kita akan menambahkan latar belakang pada form, lebih banyak ikon, serta menggunakan komponen dan warna yang ada di Bootstrap 5 agar halaman terlihat lebih modern. Menghilangkan styling khusus seperti `border-radius` yang menggunakan style inline, dan menggantinya dengan kelas Bootstrap.

Saya akan membuat beberapa penyesuaian berikut:

1. Menggunakan warna latar belakang `bg-light` pada card dengan tambahan border agar form terlihat lebih menonjol.
2. Menambahkan lebih banyak ikon Bootstrap untuk memberikan kesan interaktif pada label form.
3. Mengubah beberapa teks menjadi warna `text-secondary` agar lebih lembut dan terlihat profesional.
4. Memanfaatkan kelas utilitas seperti `rounded`, `shadow`, dan `p-4` untuk tata letak yang lebih rapi dan terstruktur.

Penyempurnaan Halaman Home

Berikut adalah hasil perbaikan kode:

```
<!-- File: apps/templates/account/register.html
-->
{% extends 'home/base.html' %}

{% block title %}Register - Platform Bot{%
endblock %}

{% block content %}
<div class="container mt-5">
    <!-- Header with Background and Text -->
    <h2 class="text-center bg-primary text-white
py-2 rounded">Registrasi Pengguna Baru</h2>

    <!-- Form Card with Background and Shadow --
>
    <div class="card mt-4 bg-light shadow-lg">
        <div class="card-body p-4">
            {% if form.errors %}
                <div class="alert alert-danger">
                    {% for field, errors in
form.errors.items %}
                        <strong>{{ field
}}</strong>
                        {% for error in errors
%}
                            <div>{{ error
}}</div>
                        {% endfor %}
                    {% endfor %}
                </div>
            {% endif %}

            <!-- Form -->
            <form method="POST"
enctype="multipart/form-data">
                {% csrf_token %}
```

Penyempurnaan Halaman Home

```
<!-- First Name and Last Name -->
<div class="row">
  <div class="col-md-6 mb-3">
    <label class="fw-bold"
for="id_first_name">
      <i class="bi bi-
person-fill"></i> {{ form.first_name.label }}
    </label>
    <div class="form-control
border border-primary">
      {{ form.first_name
}}
    </div>
    {% if
form.first_name.errors %}
      <div class="text-
danger">{{ form.first_name.errors }}</div>
    {% endif %}
  </div>
  <div class="col-md-6 mb-3">
    <label class="fw-bold"
for="id_last_name">
      <i class="bi bi-
person-fill"></i> {{ form.last_name.label }}
    </label>
    <div class="form-control
border border-primary">
      {{ form.last_name }}
    </div>
    {% if
form.last_name.errors %}
      <div class="text-
danger">{{ form.last_name.errors }}</div>
    {% endif %}
  </div>
</div>
```


Penyempurnaan Halaman Home

```
<!-- Username and Email -->
<div class="row">
  <div class="col-md-6 mb-3">
    <label class="fw-bold"
for="id_username">
      <i class="bi bi-
person-circle"></i> {{ form.username.label }}
    </label>
    <div class="form-control
border border-primary">
      {{ form.username }}
    </div>
    {% if
form.username.errors %}
      <div class="text-
danger">{{ form.username.errors }}</div>
    {% endif %}
  </div>
  <div class="col-md-6 mb-3">
    <label class="fw-bold"
for="id_email">
      <i class="bi bi-
envelope-fill"></i> {{ form.email.label }}
    </label>
    <div class="form-control
border border-primary">
      {{ form.email }}
    </div>
    {% if form.email.errors
%}
      <div class="text-
danger">{{ form.email.errors }}</div>
    {% endif %}
  </div>
</div>

<!-- Password -->
<div class="row">
  <div class="col-md-6 mb-3">
```

Penyempurnaan Halaman Home

```

                                <label class="fw-bold"
for="id_password1">
                                <i class="bi bi-
lock-fill"></i> {{ form.password1.label }}
                                </label>
                                <div class="form-control
border border-primary">
                                {{ form.password1 }}
                                </div>
                                {% if
form.password1.errors %}
                                <div class="text-
danger">{{ form.password1.errors }}</div>
                                {% endif %}
                                </div>
                                <div class="col-md-6 mb-3">
                                <label class="fw-bold"
for="id_password2">
                                <i class="bi bi-
lock-fill"></i> {{ form.password2.label }}
                                </label>
                                <div class="form-control
border border-primary">
                                {{ form.password2 }}
                                </div>
                                {% if
form.password2.errors %}
                                <div class="text-
danger">{{ form.password2.errors }}</div>
                                {% endif %}
                                </div>
                                </div>

                                <!-- Company and Address -->
                                <div class="row">
                                <div class="col-md-6 mb-3">
                                <label class="fw-bold"
for="id_company">
```

Penyempurnaan Halaman Home

```

                <i class="bi bi-
building"></i> {{ form.company.label }}
                </label>
                <div class="form-control
border border-primary">
                    {{ form.company }}
                </div>
                {% if
form.company.errors %}
                    <div class="text-
danger">{{ form.company.errors }}</div>
                    {% endif %}
                </div>
                <div class="col-md-6 mb-3">
                    <label class="fw-bold"
for="id_address">
                        <i class="bi bi-geo-
alt-fill"></i> {{ form.address.label }}
                        </label>
                        <div class="form-control
border border-primary">
                            {{ form.address }}
                        </div>
                        {% if
form.address.errors %}
                            <div class="text-
danger">{{ form.address.errors }}</div>
                            {% endif %}
                        </div>
                    </div>

                    <!-- City and Country -->
                    <div class="row">
                        <div class="col-md-6 mb-3">
                            <label class="fw-bold"
for="id_city">
                                <i class="bi bi-
building"></i> {{ form.city.label }}
                                </label>

```

Penyempurnaan Halaman Home

```

        <div class="form-control
border border-primary">
            {{ form.city }}
        </div>
        {% if form.city.errors
%}
            <div class="text-
danger">{{ form.city.errors }}</div>
            {% endif %}
        </div>
        <div class="col-md-6 mb-3">
            <label class="fw-bold"
for="id_country">
                <i class="bi bi-
globe"></i> {{ form.country.label }}
            </label>
            <div class="form-control
border border-primary">
                {{ form.country }}
            </div>
            {% if
form.country.errors %}
                <div class="text-
danger">{{ form.country.errors }}</div>
                {% endif %}
            </div>
        </div>

        <!-- Submit Button -->
        <div class="text-center">
            <button type="submit"
class="btn btn-primary my-3">
                <i class="bi bi-person-
plus-fill"></i> Daftar
            </button>

        <br /><br />

        <p>

```

Penyempurnaan Halaman Home

```
                Sudah punya akun? <a href="{%
url 'login' %}" class="text-primary">Masuk</a>
            </p>
        </div>
    </form>
</div>
</div>
</div>
{% endblock %}
```

Perubahan:

1. **Background Card:** Saya menambahkan `bg-light` dan `shadow-lg` pada card agar lebih menarik dan profesional.
2. **Form Structure:** Menggunakan lebih banyak ikon pada label input, dan menjaga konsistensi warna dengan `border-primary` untuk menambahkan highlight pada setiap form field.
3. **Error Handling:** Jika ada error, tampilan form akan tetap fokus dengan warna teks error berwarna merah (`text-danger`) di bawah field yang bersangkutan.
4. **Responsive Design:** Menggunakan `row` dan `col-md-6` untuk memastikan layout responsif dan sesuai di berbagai ukuran layar.

Dengan menggunakan sistem grid Bootstrap dan komponen-komponen seperti `card` dan `form-control`, halaman register menjadi responsif dan terlihat lebih profesional pada berbagai ukuran layar. Pastikan untuk menguji tampilan pada perangkat yang berbeda dan berbagai ukuran layar untuk memastikan bah-

Penyempurnaan Halaman Home

wa halaman register berfungsi dengan baik dan tampil menarik di semua perangkat.

13.3.3 Menguji Tampilan Halaman Register

Untuk menguji tampilan halaman register, jalankan server Django lokal dan buka halaman register di browser:

```
$ python manage.py runserver
```

Buka URL `http://localhost:8000/register/` di browser Anda dan periksa apakah halaman register tampil sesuai harapan. Periksa elemen-elemen formulir, pastikan semua komponen Bootstrap bekerja dengan baik dan tampilan halaman responsif pada berbagai ukuran layar.

Dengan pembaruan ini, halaman register Anda tidak hanya terlihat lebih profesional dengan integrasi Bootstrap tetapi juga memberikan pengalaman pengguna yang lebih baik dan responsif.

13.4 Penyempurnaan Halaman Login

Pada tahap ini, kita akan menyempurnakan halaman `login.html` agar tampak lebih menarik dan responsif, serta memberikan pengalaman yang lebih baik bagi pengguna saat melakukan login di platform bot.

13.4.1 Memahami Struktur Halaman Login Saat Ini

Sebelum melakukan modifikasi, kita perlu memahami struktur halaman `login.html` yang sudah ada. Saat ini, halaman ini memiliki form login yang ditampilkan dalam bentuk card, dengan elemen seperti `username`, dan, `password`.

File yang digunakan saat ini adalah `apps/templates/account/register.html`. Di dalam file ini, struktur form sudah cukup baik, namun kita akan melakukan beberapa penyesuaian untuk membuat tampilannya lebih menarik, dengan menggunakan warna latar belakang yang lebih segar dan tata letak form yang lebih rapi.

13.4.2 Mengintegrasikan Bootstrap Grid System

Salah satu langkah awal dalam menyempurnakan tampilan halaman registrasi adalah dengan memanfaatkan ***Bootstrap Grid System***. Grid system ini memungkinkan kita untuk mengatur tata letak yang responsif sesuai dengan ukuran layar pengguna.

Saat ini, form login sudah menggunakan `col-md-6` untuk mengatur lebar kolom, namun kita akan menambahkan lebih banyak kontrol atas tampilan di perangkat yang lebih kecil. Kita juga akan memanfaatkan kelas `offset-md-3` untuk memastikan form berada di tengah halaman pada layar besar.

Untuk membuat halaman login yang lebih menarik dengan menambahkan ***background color*** dan mempercantik form dengan

Penyempurnaan Halaman Home

icon, kita bisa memanfaatkan elemen-elemen dari **Bootstrap** untuk memperkaya tampilan, sambil menyesuaikan warna dan menambahkan ikon yang relevan. Di bawah ini adalah penjelasan dan modifikasi dari kode yang sudah ada.

Langkah-langkah Penyempurnaan:

1. **Menambahkan Background Color:** Kita akan menggunakan kelas Bootstrap untuk menambahkan latar belakang pada halaman. Pada bagian `<div class="content">`, tambahkan kelas background dengan warna yang lembut agar nyaman di mata pengguna.
2. **Menambahkan Ikon pada Form:** Kita akan menggunakan ikon dari **Bootstrap** untuk menambahkan visual ke kolom input form, sehingga pengguna lebih mudah mengenali fungsinya.

Kode Halaman Login yang Diperbarui:

```
<!-- File: apps/templates/account/login.html -->
{% extends 'home/base.html' %}

{% block title %}Login - Platform Bot{% endblock %}

{% block content %}
<div class="content" style="background-color: #f4f6f9; min-height: 100vh;">
  <div class="container">
    <!-- Display Django messages -->
    {% if messages %}
    <div class="row">
```


Penyempurnaan Halaman Home

```
<div class="col-md-6 offset-md-3">
    {% for message in messages %}
        <div class="alert alert-
{{ message.tags }}">
            <button type="button"
class="close" data-dismiss="alert" aria-
label="Close">
                <span aria-
hidden="true">&times;</span>
            </button>
            {{ message }}
        </div>
    {% endfor %}
</div>
{% endif %}

<div class="row pt-5">
    <div class="col-md-6 mt-5 offset-md-
3 pt-5 mt-5">
        <div class="card shadow-lg"
style="border-radius: 15px;">
            <div class="card-header
text-center py-4 bg-primary text-white"
style="border-radius: 15px 15px 0 0;">
                <h4 class="title"><i
class="bi bi-person-circle"></i> Login to
Platform Bot</h4>
            </div>
            <div class="card-body px-5
py-4">
                <form method="post"
id="login-form">
                    {% csrf_token %}
                    <div class="form-
group mb-3">
                        <label
class="fw-bold" for="id_username">
```

Penyempurnaan Halaman Home

```

                                <i class="bi
bi-person-fill"></i> Username
                                </label>
                                <div
class="input-group">
                                <div
class="input-group-prepend">
                                <span
class="input-group-text bg-primary text-white">
                                <i
class="bi bi-person-fill"></i>
                                </span>
                                </div>

{{ form.username|add_class:"form-control border
border-primary" }}
                                </div>
                                </div>

                                <div class="form-
group mb-3">
                                <label
class="fw-bold" for="id_password">
                                <i class="bi
bi-lock-fill"></i> Password
                                </label>
                                <div
class="input-group">
                                <div
class="input-group-prepend">
                                <span
class="input-group-text bg-primary text-white">
                                <i
class="bi bi-lock-fill"></i>
                                </span>
                                </div>

{{ form.password|add_class:"form-control border
border-primary" }}
```

[illegible]

1. ***Ikon Bootstrap***: Ditambahkan ikon Bootstrap pada bagian header, input field (username dan password), serta link pendaftaran untuk memperindah tampilan.
2. ***Warna dan Komponen Bootstrap***: Menggunakan bg-primary untuk header dan ikon, serta elemen input de-

Penyempurnaan Halaman Home

ngan `form-control border border-primary` untuk membuat form lebih berwarna dan rapi.

3. ***Penempatan Tombol:*** Tombol Login menggunakan ikon dan ukuran yang lebih besar untuk kesan lebih profesional.

Dengan ini, halaman login akan lebih menarik secara visual, tetap fungsional, dan responsif.

Kesimpulan Bab

Di Bab ini kita fokus pada *penyempurnaan halaman home* dalam proyek Django dengan memanfaatkan **Bootstrap** untuk meningkatkan tampilan dan responsivitas halaman.

Berikut adalah ringkasan dari setiap subbab:

1. *Pendahuluan*

- ***Tujuan dan Manfaat Penggunaan Bootstrap:*** Bootstrap merupakan framework CSS yang sangat membantu dalam mempercepat proses desain dan pengembangan antarmuka web dengan menyediakan komponen siap pakai dan sistem grid responsif.
- ***Pengenalan Dasar Bootstrap:*** Bootstrap menyediakan berbagai komponen dan utilitas CSS yang memungkinkan pembuatan desain web yang modern dan responsif dengan cepat.
- ***Instalasi dan Konfigurasi Bootstrap di Proyek Django:*** Instalasi dilakukan melalui CDN atau dengan mengunduh file dari situs Bootstrap. Konfigurasi di Django melibatkan pengaturan file statis untuk memastikan Bootstrap diintegrasikan dengan benar dalam proyek.

2. *Penyempurnaan Halaman Home*

- ***Perbaiki base.html:*** Perubahan dilakukan pada template `base.html` untuk menyertakan Bootstrap dan komponen tambahan, meningkatkan struktur halaman dengan menambahkan

Penyempurnaan Halaman Home

stylesheet dan script yang diperlukan untuk tampilan yang lebih baik.

- **Menambahkan Komponen Bootstrap:** Komponen seperti cards dan jumbotron ditambahkan untuk meningkatkan tampilan halaman home, memberikan informasi yang lebih terstruktur dan visual yang lebih menarik.
- **Menyempurnakan Tampilan Responsif:** Penyesuaian dilakukan untuk memastikan halaman home tampil dengan baik pada berbagai perangkat, termasuk penggunaan kelas responsif Bootstrap untuk menata elemen dengan efektif.
- **Menambahkan Konten di Halaman Home:** Konten tambahan seperti pesan status, log aktivitas, dan placeholder charts diintegrasikan untuk memberikan informasi yang lebih kaya kepada pengguna.
- **Menguji Tampilan Halaman Home:** Setelah perubahan diterapkan, tampilan halaman home diuji untuk memastikan bahwa semua elemen bekerja dengan baik dan tampil sesuai dengan desain yang diinginkan.

3. Penyempurnaan Halaman Register

- **Mengintegrasikan Bootstrap Grid System:** Sistem grid Bootstrap digunakan untuk menata elemen-elemen pada halaman register dengan rapi, membuat tata letak yang lebih terstruktur.
- **Menambahkan Komponen Bootstrap:** Komponen Bootstrap seperti form controls dan buttons

Penyempurnaan Halaman Home

ditambahkan untuk meningkatkan tampilan dan fungsionalitas halaman register.

- ***Menguji Tampilan Halaman Register:*** Tampilan halaman register diuji untuk memastikan responsivitas dan keakuratan tampilan sesuai dengan desain.

4. *Penyempurnaan Halaman Login*

- ***Memahami Struktur Halaman Login Saat Ini:*** Menilai struktur halaman login yang ada untuk mengidentifikasi area yang membutuhkan perbaikan.
- ***Mengintegrasikan Bootstrap Grid System:*** Implementasi sistem grid Bootstrap pada halaman login untuk memastikan tampilan yang rapi dan responsif.

Dengan menerapkan langkah-langkah yang dijelaskan di bab ini, halaman-halaman utama dalam aplikasi Django dapat ditingkatkan secara signifikan, menawarkan pengalaman pengguna yang lebih baik dan desain yang lebih modern. Penggunaan Bootstrap memungkinkan pembuatan antarmuka yang konsisten dan menarik dengan efisiensi yang tinggi.

BAB 14 -Penyempurnaan Halaman Dashboard

Dalam bab ini, kita akan fokus pada penyempurnaan halaman dashboard dari proyek platform_bot kita dengan menggunakan Bootstrap. Untuk penjelasan dan keunggulan bootstrap sudah kita bahas sebelumnya.

Selanjutnya kita akan mulai dengan menyempurnakan halaman dashboard. Sebelum kita memperbaiki nya, kita akan memisahkan terlebih dahulu untuk navbar, sidebar, dll ke file terpisah.

Untuk memisahkan komponen seperti navbar dan sidebar agar lebih terorganisir, kita dapat menggunakan konsep *template inheritance* di dalam Django atau *partial views* jika menggunakan framework lain. Langkah ini akan membantu menjaga kebersihan dan keteraturan struktur file serta memudahkan pemeliharaan.

Dalam membangun sebuah aplikasi web, penting untuk menjaga kebersihan dan modularitas kode, terutama saat bekerja dengan elemen-elemen yang sering digunakan berulang, seperti *navbar* dan *sidebar*. Dalam bab ini, kita akan membahas bagaimana memisahkan komponen-komponen ini dalam struktur direktori yang rapi menggunakan Django, sebuah framework web berbasis Python.

14.1 Memisahkan Struktur Direktori Template

Sebelum masuk ke pemisahan komponen, kita perlu memahami struktur direktori yang akan kita gunakan. Struktur yang baik memudahkan pengelolaan file dan memungkinkan penambahan komponen baru tanpa mengacaukan kode utama. Berikut ini adalah struktur direktori template yang akan kita buat untuk halaman dashboard:

```
apps/  
  templates/  
    dashboard/  
      base.html  
      navbar.html  
      sidebar.html  
      dashboard.html
```

Mari kita bahas satu per satu:

- ***apps/***: Folder ini adalah direktori proyek aplikasi Django Anda.
- ***templates/***: Direktori ini adalah tempat di mana semua file template HTML akan disimpan. Ini adalah folder default yang digunakan Django untuk mencari template jika Anda mengaturnya di pengaturan proyek.
- ***dashboard/***: Sub-direktori di dalam ***templates/*** yang dikhususkan untuk menyimpan template terkait halaman dashboard. Dengan memisahkannya dalam folder tersendiri, Anda memastikan bahwa template khusus dashboard tidak tercampur dengan template lain yang mungkin Anda buat di aplikasi lain.

Penyempurnaan Halaman Dashboard

- **base.html**: Template utama yang akan menjadi kerangka dasar dari seluruh halaman dashboard. Di dalamnya, Anda akan mengimpor komponen seperti *navbar* dan *sidebar* serta bagian konten utama.
- **navbar.html**: Template yang dikhususkan untuk bagian *navbar* (menu navigasi atas). Komponen ini biasanya seragam di seluruh halaman dashboard dan bisa dipanggil melalui `base.html`.
- **sidebar.html**: Template untuk *sidebar* (menu navigasi samping). Seperti *navbar*, *sidebar* biasanya digunakan secara konsisten di banyak halaman dashboard, jadi sangat cocok untuk dipisahkan.
- **dashboard.html**: Template untuk halaman utama dashboard yang akan mewarisi kerangka dari `base.html`.

14.1.1 Memisahkan base

File `base.html` adalah fondasi dari seluruh halaman dashboard. Di sini, kita akan menentukan kerangka dasar yang mencakup elemen-elemen inti seperti *navbar*, *sidebar*, dan tempat untuk konten utama. Sebagai langkah pertama, mari buat file `base.html` di direktori `dashboard/`:

```
<!-- File: apps/templates/dashboard/base.html -->
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>{% block title %}Dashboard{% endblock %}</title>
```

Penyempurnaan Halaman Dashboard

```
<!-- Menambahkan Bootstrap dan
Bootstrap Icons dari CDN -->
<link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3
.3/dist/css/bootstrap.min.css" rel="stylesheet">
<link
href="https://cdn.jsdelivr.net/npm/bootstrap-
icons@1.10.5/font/bootstrap-icons.css"
rel="stylesheet">
</head>
<body>
<!-- Include Navbar -->
{% include 'dashboard/navbar.html' %}

<!-- Include Sidebar -->
{% include 'dashboard/sidebar.html' %}

<!-- Main Content -->
<main class="mt-5 pt-3">
  {% block content %}
    <!-- Konten halaman akan masuk di sini -->
    {% endblock %}
  </main>

  <!-- Menyertakan JavaScript
  Bootstrap -->
  <script
src="https://cdn.jsdelivr.net/npm/@popperjs/core
@2.11.8/dist/umd/popper.min.js"></script>
  <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.
3/dist/js/bootstrap.min.js"></script>
</body>
</html>
```

Template ini berfungsi sebagai kerangka kerja dasar yang menyertakan *navbar* dan *sidebar* dari file terpisah serta menyediakan blok untuk konten halaman tertentu.

Penyempurnaan Halaman Dashboard

14.1.2 Memisahkan navbar

Template `navbar.html` adalah komponen yang menangani bagian navigasi atas. Komponen ini akan dipanggil ke dalam `base.html`. Dengan membuatnya terpisah, kita dapat menggunakannya kembali di banyak halaman tanpa perlu menulis ulang kode. Berikut adalah contoh sederhana untuk *navbar*:

```
<!-- File: apps/templates/dashboard/navbar.html -->
<nav class="navbar navbar-expand-lg navbar-light bg-light">
  <div class="container-fluid">
    <a class="navbar-brand" href="{% url 'home' %}">Platform Bot</a>
    <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#sidebarMenu" aria-controls="sidebarMenu" aria-expanded="false" aria-label="Toggle sidebar">
      <span class="navbar-toggler-icon"></span>
    </button>
  </div>
</nav>
```

14.1.3 Memisahkan sidebar

Sidebar juga merupakan komponen yang akan sering digunakan dalam halaman dashboard. Memisahnya menjadi file terpisah memungkinkan kita melakukan perubahan secara terpusat. Berikut adalah contoh untuk `sidebar.html`:

Penyempurnaan Halaman Dashboard

```
<!-- File: apps/templates/dashboard/sidebar.html
-->
<div class="col-md-3">
  <div class="collapse d-md-block"
id="sidebarMenu">
    <nav id="sidebar" class="bg-light
sidebar">
      <div class="position-sticky">
        <ul class="nav flex-column">
          <li class="nav-item">
            <a class="nav-link
active" href="{% url 'dashboard' %}">
              <i class="bi bi-
house-door"></i> Dashboard
            </a>
          </li>
          <li class="nav-item">
            <a class="nav-link"
href="{% url 'profile' %}">
              <i class="bi bi-
person"></i> Profile
            </a>
          </li>
          <li class="nav-item">
            <a class="nav-link"
href="#">
              <i class="bi bi-
plus"></i> Create Bot
            </a>
          </li>
          <li class="nav-item">
            <a class="nav-link"
href="#">
              <i class="bi bi-
gear"></i> Manage Bots
            </a>
          </li>
        </ul>
      </div>
    <!-- Tombol Logout di
Sidebar -->
```

Penyempurnaan Halaman Dashboard

```
        <li class="nav-item">
          <a class="nav-link text-
danger" href="{% url 'logout' %}">
            <i class="bi bi-box-
arrow-right"></i> Logout
          </a>
        </li>
      </ul>
    </div>
  </nav>
</div>
</div>
```

14.1.4 Memisahkan dashboard

Template `dashboard.html` adalah halaman yang spesifik untuk konten dashboard, dan halaman ini akan menggunakan `base.html` sebagai kerangka utamanya. Pada template ini, kita akan menulis konten di dalam `{% block content %}`.

Berikut adalah contohnya:

```
<!-- File:
apps/templates/dashboard/dashboard.html -->
{% extends 'dashboard/base.html' %}

{% block content %}
<div class="container">
  <div class="row">
    <div class="col-lg-12">
      <div class="jumbotron text-center
bg-primary text-white py-5">
        <h2>Selamat Datang di Dashboard,
{{ user_profile.user.username }}!</h2>
      </div>
    </div>
  </div>
</div>
{% endblock %}
```

Penyempurnaan Halaman Dashboard

```
<p>Di sini Anda dapat mengelola
bot, melihat statistik, dan memeriksa log
aktivitas terbaru Anda.</p>
</div>
</div>
</div>
<!-- Menampilkan pesan jika ada error atau
sukses -->
{% if messages %}
  <div class="messages">
    {% for message in messages %}
      <div class="alert
{{ message.tags }}">{{ message }}</div>
    {% endfor %}
  </div>
{% endif %}

<div class="row mt-5">
  <div class="col-lg-12">
    <h3 class="text-center">Log
Aktivitas</h3>
    <table class="table table-striped
table-hover">
      <thead class="thead-dark">
        <tr>
          <th
scope="col">Waktu</th>
          <th
scope="col">Aktivitas</th>
        </tr>
      </thead>
      <tbody>
        {% for log in activity_logs %}
          <tr>
            <td>{{ log.timestamp |
date:"d-m-Y H:i" }}</td>
```

Penyempurnaan Halaman Dashboard

```
                                <td>{{ log.action
}}</td>
                                </tr>
                                {% empty %}
                                <tr>
                                    <td colspan="2"
class="text-center">Tidak ada log aktivitas
ditemukan.</td>
                                </tr>
                                {% endfor %}
                            </tbody>
                        </table>
                    </div>
                </div>
            </div>
        {% endblock %}
```

Dengan memisahkan komponen seperti *navbar* dan *sidebar* ke dalam file terpisah, kita dapat menjaga kode lebih modular dan mudah dikelola. Selain itu, perubahan yang dilakukan pada satu komponen akan otomatis diterapkan ke seluruh halaman yang menggunakan komponen tersebut. Struktur template yang jelas dan bersih seperti ini tidak hanya mempermudah pemeliharaan tetapi juga meningkatkan efisiensi saat mengembangkan aplikasi berbasis web.

14.2 Memperbaiki Halaman

Selanjutnya kita akan mulai untuk memperbaiki semua halaman dashboard nya, mulai dari menyiapkan css dan js, dependensi dan lainnya , agar tampilan dashboard nya lebih interaktif dan responsif di berbagai platform.

Penyempurnaan Halaman Dashboard

14.2.1 Menyiapkan css dan js untuk Dashboard

Sekarang kita akan mulai untuk memperbaiki tampilan dashboard nya, pertama kita akan menyiapkan css dan js untuk dashboard nya, kita juga akan membahas bagaimana kode CSS dan JavaScript yang diberikan dapat diintegrasikan ke dalam proyek Django, serta menjelaskan fungsi setiap bagian dari kode tersebut. Mari kita mulai dengan CSS untuk tata letak sidebar dan kemudian melihat bagaimana JavaScript digunakan untuk mengaktifkan chart dan tabel.

Menyiapkan Kode CSS

File CSS yang kita gunakan berada di direktori berikut:

```
apps/static/assets/css/style.css
```

Berikut adalah penjelasan dari kode CSS yang diberikan.

```
:root {  
  --offcanvas-width: 270px;  
  --topNavbarHeight: 56px; /* Perbaiki dari --  
topNavbarHeight menjadi --topNavbarHeight */  
}
```

Pada bagian ini, kita mendefinisikan dua variabel CSS global yang digunakan untuk menentukan lebar sidebar (`--offcanvas-width`) dan tinggi navbar (`--topNavbarHeight`). Dengan menggunakan variabel ini, kita dapat dengan mudah mengubah ukuran sidebar atau navbar di seluruh aplikasi hanya dengan memperbarui nilai variabel tersebut.

```
.sidebar-nav {
```

Penyempurnaan Halaman Dashboard

```
width: var(--offcanvas-width);
}
```

Kelas `.sidebar-nav` digunakan untuk mengatur lebar dari sidebar sesuai dengan nilai variabel yang telah kita definisikan sebelumnya (`--offcanvas-width`). Di sini, sidebar akan memiliki lebar tetap sebesar 270px.

```
.sidebar-link {
  display: flex;
  align-items: center;
}
```

Pada `.sidebar-link`, kita memastikan bahwa setiap tautan di sidebar menggunakan flexbox untuk menata elemen-elemen di dalamnya. `align-items: center` memastikan teks dan ikon di dalam tautan akan sejajar secara vertikal di tengah.

```
.sidebar-link .right-icon {
  display: inline-flex;
  transition: all ease 0.25s;
}

.sidebar-link[aria-expanded="true"] .right-icon {
  transform: rotate(180deg);
}
```

Ini adalah pengaturan untuk ikon dropdown di sidebar. Ketika sebuah item sidebar diperluas (`aria-expanded="true"`), ikon akan berputar 180 derajat dengan transisi yang halus dalam waktu 0.25 detik.

Penyempurnaan Halaman Dashboard

CSS untuk Desktop (min-width: 992px)

```
@media (min-width: 992px) {  
  .offcanvas-start {  
    transform: none !important;  
    visibility: visible !important;  
    position: fixed;  
    top: var(--topNavbarHeight);  
    height: calc(100% - var(--  
topNavbarHeight));  
    z-index: 1030;  
  }  
  
  main {  
    margin-left: var(--offcanvas-width);  
    margin-top: var(--topNavbarHeight);  
  }  
  
  .navbar {  
    z-index: 1040;  
  }  
}
```

Untuk tampilan desktop, kita memastikan bahwa sidebar selalu terlihat dan tidak tergeser. Sidebar diposisikan secara tetap (`position: fixed`) tepat di bawah navbar dengan jarak sebesar `--topNavbarHeight`. `main` juga diberi margin kiri agar konten tidak tertutup oleh sidebar, dan navbar memiliki `z-index` yang lebih tinggi sehingga akan selalu berada di atas sidebar.

CSS untuk Mobile (max-width: 991.98px)

```
@media (max-width: 991.98px) {
```

Penyempurnaan Halaman Dashboard

```
.offcanvas-start {  
    transform: translateX(-100%);  
}  
  
.offcanvas-start.show {  
    transform: translateX(0);  
}
```

Untuk tampilan mobile, sidebar diatur agar tersembunyi secara default (`transform: translateX(-100%)`). Ketika tombol sidebar ditekan, kelas `.show` akan diaktifkan dan sidebar akan muncul dengan transisi mulus.

Kode Lengkap

```
/* File: apps/static/assets/css/style.css */  
  
:root {  
    --offcanvas-width: 270px;  
    --topNavbarHeight: 56px; /* Perbaiki dari --  
topNavbarMeight menjadi --topNavbarHeight */  
}  
  
.sidebar-nav {  
    width: var(--offcanvas-width);  
}  
  
.sidebar-link {  
    display: flex;  
    align-items: center;  
}  
  
.sidebar-link .right-icon {  
    display: inline-flex;  
    transition: all ease 0.25s;
```

Penyempurnaan Halaman Dashboard

```
}

.sidebar-link[aria-expanded="true"] .right-icon
{
    transform: rotate(180deg);
}

@media (min-width: 992px) {
    /* Pastikan sidebar selalu muncul di desktop */
    .offcanvas-start {
        transform: none !important;
        visibility: visible !important;
        position: fixed;
        top: var(--topNavbarHeight);
        height: calc(100% - var(--topNavbarHeight));
        z-index: 1030; /* Pastikan sidebar di atas konten lainnya */
    }

    /* Tambahkan margin untuk konten utama agar tidak tertutup sidebar */
    main {
        margin-left: var(--offcanvas-width);
        margin-top: var(--topNavbarHeight);
    }

    /* Pastikan navbar berada di atas sidebar */
    .navbar {
        z-index: 1040; /* Lebih tinggi dari sidebar */
    }
}

@media (max-width: 991.98px) {
    /* Sembunyikan sidebar di mobile */
    .offcanvas-start {
```

Penyempurnaan Halaman Dashboard

```
        transform: translateX(-100%);
    }

    .offcanvas-start.show {
        transform: translateX(0); /* Munculkan
sidebar saat tombol ditekan */
    }
}
```

Menyiapkan Kode JavaScript

File JavaScript yang digunakan berada di direktori berikut:

```
apps/static/assets/js/script.js
```

Pada bagian ini, JavaScript digunakan untuk dua fungsi utama, yaitu menampilkan chart menggunakan Chart.js dan menginisialisasi DataTables untuk tabel data.

Mengaktifkan Chart dengan Chart.js

```
const charts =
document.querySelectorAll(".chart");

charts.forEach(function (chart) {
    var ctx = chart.getContext("2d");
    var myChart = new Chart(ctx, {
        type: "bar",
        data: {
            labels: ["Red", "Blue", "Yellow", "Green",
"Purple", "Orange"],
            datasets: [
                {
                    label: "# of Votes",
```

Penyempurnaan Halaman Dashboard

```
data: [12, 19, 3, 5, 2, 3],
backgroundColor: [
  "rgba(255, 99, 132, 0.2)",
  "rgba(54, 162, 235, 0.2)",
  "rgba(255, 206, 86, 0.2)",
  "rgba(75, 192, 192, 0.2)",
  "rgba(153, 102, 255, 0.2)",
  "rgba(255, 159, 64, 0.2)",
],
borderColor: [
  "rgba(255, 99, 132, 1)",
  "rgba(54, 162, 235, 1)",
  "rgba(255, 206, 86, 1)",
  "rgba(75, 192, 192, 1)",
  "rgba(153, 102, 255, 1)",
  "rgba(255, 159, 64, 1)",
],
borderWidth: 1,
},
],
},
options: {
  scales: {
    y: {
      beginAtZero: true,
    },
  },
},
},
});
});
```

Pada kode ini, kita menggunakan `Chart.js` untuk membuat chart.

1. `querySelectorAll(".chart")` akan mencari semua elemen dengan kelas `chart` di halaman.
2. Setiap elemen `chart` akan di-loop menggunakan `forEach`, dan untuk setiap `chart`, kita inisialisasi `Chart`

Penyempurnaan Halaman Dashboard

menggunakan `getContext("2d")` untuk menggambar chart di dalam canvas 2D.

3. Data untuk chart bar disiapkan dengan labels (kategori warna) dan datasets (jumlah votes). Kita juga menambahkan warna latar belakang dan warna border untuk setiap bar di chart.

Inisialisasi DataTables

```
$(document).ready(function () {  
    $(".data-table").each(function (_, table) {  
        $(table).DataTable();  
    });  
});
```

Pada bagian ini, kita menggunakan plugin jQuery DataTables untuk membuat tabel lebih interaktif. Saat halaman sudah siap (`$(document).ready()`), setiap tabel dengan kelas `data-table` akan diinisialisasi menjadi tabel yang dapat disortir dan difilter secara otomatis oleh DataTables.

Kode Lengkap

```
// File: apps/static/assets/js/script.js  
  
const charts =  
document.querySelectorAll(".chart");  
  
charts.forEach(function (chart) {  
    var ctx = chart.getContext("2d");  
    var myChart = new Chart(ctx, {  
        type: "bar",  
        data: {
```


Penyempurnaan Halaman Dashboard

```
    labels: ["Red", "Blue", "Yellow", "Green",
"Purple", "Orange"],
    datasets: [
        {
            label: "# of Votes",
            data: [12, 19, 3, 5, 2, 3],
            backgroundColor: [
                "rgba(255, 99, 132, 0.2)",
                "rgba(54, 162, 235, 0.2)",
                "rgba(255, 206, 86, 0.2)",
                "rgba(75, 192, 192, 0.2)",
                "rgba(153, 102, 255, 0.2)",
                "rgba(255, 159, 64, 0.2)",
            ],
            borderColor: [
                "rgba(255, 99, 132, 1)",
                "rgba(54, 162, 235, 1)",
                "rgba(255, 206, 86, 1)",
                "rgba(75, 192, 192, 1)",
                "rgba(153, 102, 255, 1)",
                "rgba(255, 159, 64, 1)",
            ],
            borderWidth: 1,
        },
    ],
    options: {
        scales: {
            y: {
                beginAtZero: true,
            },
        },
    },
});

$(document).ready(function () {
    $(".data-table").each(function (_, table) {
```

Penyempurnaan Halaman Dashboard

```
$(table).DataTable();
});
});
```

Demikianlah penjelasan untuk kode CSS dan JavaScript yang kamu miliki. Pastikan untuk memasukkan kedua file ini ke dalam template Django agar bisa berfungsi dengan baik pada halaman dashboard.

14.2.2 Memperbaiki Template `base.html` untuk Dashboard

Selanjutnya kita akan membahas bagaimana memperbaiki template `base.html` dari struktur awal menjadi lebih fungsional, dengan tambahan dependensi yang dibutuhkan untuk mengelola berbagai elemen pada halaman dashboard. Kode ini merupakan bagian penting dari pengaturan template Django, yang memungkinkan kita untuk menyusun layout dengan lebih efisien dan menggunakan berbagai library, seperti Bootstrap, DataTables, dan Chart.js.

Perbaiki Template `base.html`

Setelah dilakukan perbaikan, berikut adalah kode yang telah ditingkatkan:

```
<!-- File: apps/templates/dashboard/base.html -->
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
```

Penyempurnaan Halaman Dashboard

```
<meta http-equiv="X-UA-Compatible"
content="IE=edge" />
<meta name="viewport" content="width=device-
width, initial-scale=1.0" />
<!-- Menambahkan Bootstrap dan Bootstrap
Icons dari CDN -->
<link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@5.0
.0-beta3/dist/css/bootstrap.min.css" />
<link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap-
icons@1.4.1/font/bootstrap-icons.css"/>
<link rel="stylesheet"
href="https://cdn.datatables.net/1.10.24/css/dat
aTables.bootstrap5.min.css" />
<link rel="stylesheet" href="{ ASSETS_ROOT
}}/css/style.css" />
<title>{% block title %}Dashboard{% endblock
%}</title>
</head>
<body>
<!-- Include Navbar -->
{% include 'dashboard/navbar.html' %}

<!-- Include Sidebar -->
{% include 'dashboard/sidebar.html' %}

<main class="mt-5 pt-3">
  <div class="container">
    {% block content %}
    <!-- Content will be injected here from
other templates -->
    {% endblock %}
  </div>
</main>

<!-- Menambahkan javascript dan JQuery CDN
-->
```

Penyempurnaan Halaman Dashboard

```
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.
0-beta3/dist/js/bootstrap.bundle.min.js"></
script>
<script
src="https://cdn.jsdelivr.net/npm/chart.js@3.0.2
/dist/chart.min.js"></script>
<script src="https://code.jquery.com/jquery-
3.5.1.js"></script>
<script
src="https://cdn.datatables.net/1.10.24/js/jquer
y.dataTables.min.js"></script>
<script
src="https://cdn.datatables.net/1.10.24/js/dataT
ables.bootstrap5.min.js"></script>
<script src="{ ASSETS_ROOT
}}/js/script.js"></script>
</body>
</html>
```

Mari kita jelaskan setiap bagian dari perubahan ini secara lebih mendalam.

Penambahan Meta Tag untuk Kompatibilitas

```
<meta http-equiv="X-UA-Compatible"
content="IE=edge" />
```

Tag ini memastikan kompatibilitas yang lebih baik di Internet Explorer dengan menginstruksikan browser untuk menggunakan Edge mode, yang mendukung fitur HTML5 dan CSS modern.

Menggunakan Versi Spesifik dari Bootstrap dan DataTables

Penyempurnaan Halaman Dashboard

```
<link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@5.0
.0-beta3/dist/css/bootstrap.min.css" />
<link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap-
icons@1.4.1/font/bootstrap-icons.css"/>
<link rel="stylesheet"
href="https://cdn.datatables.net/1.10.24/css/dat
aTables.bootstrap5.min.css" />
<link rel="stylesheet" href="{ ASSETS_ROOT
}}/css/style.css" />
```

- **Bootstrap v5.0.0-beta3:** Versi ini lebih stabil dan mendukung berbagai komponen UI modern. Kita juga menyertakan versi Bootstrap yang konsisten untuk menghindari ketidaksesuaian dengan versi lain.
- **DataTables dengan Bootstrap 5 Styling:** Ditambahkan untuk memastikan tabel interaktif memiliki tampilan yang serasi dengan Bootstrap 5.
- **Custom Styles:** File CSS khusus `style.css` dari `ASSETS_ROOT` disertakan untuk memungkinkan pengaturan gaya kustom yang lebih lanjut.

Menambahkan Container untuk Main Content

```
<main class="mt-5 pt-3">
  <div class="container">
    {% block content %}
    <!-- Content will be injected here from
other templates -->
    {% endblock %}
  </div>
</main>
```

Penyempurnaan Halaman Dashboard

Menambahkan kelas Bootstrap `container` di sekitar blok konten membantu menata konten dengan batasan yang lebih teratur di dalam layout grid Bootstrap.

Menambahkan Berbagai CDN untuk JavaScript

```
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.
0-beta3/dist/js/bootstrap.bundle.min.js"></
script>
<script
src="https://cdn.jsdelivr.net/npm/chart.js@3.0.2
/dist/chart.min.js"></script>
<script src="https://code.jquery.com/jquery-
3.5.1.js"></script>
<script
src="https://cdn.datatables.net/1.10.24/js/jquer
y.dataTables.min.js"></script>
<script
src="https://cdn.datatables.net/1.10.24/js/dataT
ables.bootstrap5.min.js"></script>
<script src="{{ ASSETS_ROOT
}}/js/script.js"></script>
```

Penambahan library berikut memungkinkan berbagai fitur tambahan di halaman dashboard:

- **Bootstrap Bundle JS:** Ini mencakup Popper.js yang digunakan untuk dropdown dan komponen lain yang interaktif di Bootstrap.
- **Chart.js:** Library ini digunakan untuk menggambar grafik (charts) di dashboard.
- **jQuery & DataTables:** jQuery adalah dependensi yang dibutuhkan oleh DataTables. DataTables digunakan un-

Penyempurnaan Halaman Dashboard

tuk mengelola tabel interaktif yang dapat disortir, difilter, dan dipaginasi.

- **Custom JavaScript:** File `script.js` dari `ASSETS_ROOT` akan mengandung kode kustom seperti inisialisasi chart atau pengaturan DataTables.

Dengan perbaikan ini, template `base.html` sekarang menjadi lebih lengkap dan siap untuk digunakan dalam proyek dashboard berbasis Django. Kode ini telah mencakup:

1. **Penambahan Bootstrap dan dependensi terkait** yang penting untuk tata letak dan gaya.
2. **Pengelolaan tabel interaktif** menggunakan DataTables.
3. **Penambahan Chart.js** untuk memungkinkan visualisasi data secara grafis.
4. **Struktur konten yang lebih baik** dengan penggunaan container dan modularisasi konten menggunakan blok.

Kita sekarang memiliki template yang fleksibel dan mendukung pengembangan dashboard dengan fitur modern.

14.2.3 Memperbaiki Sidebar untuk Dashboard

Pada bagian ini, kita akan membahas perbaikan yang dilakukan pada struktur sidebar untuk dashboard. Sidebar merupakan elemen penting dalam dashboard karena berfungsi sebagai navigasi utama yang membantu pengguna mengakses berbagai fitur dan halaman yang ada di dalam aplikasi.

Sebelumnya, sidebar menggunakan layout yang cukup sederhana dengan penggunaan kelas `collapse` dari Bootstrap. Namun,

Penyempurnaan Halaman Dashboard

untuk membuat tampilan sidebar lebih responsif dan modern, kita akan memperbaikinya dengan menggunakan komponen `offcanvas` dari Bootstrap yang lebih fleksibel.

Masalah dari Struktur Lama

1. ***Tidak Responsif Sepenuhnya:*** Sidebar menggunakan `collapse` yang mungkin tidak cukup fleksibel untuk perubahan ukuran layar yang lebih kecil.
2. ***Tata Letak Kaku:*** Menggunakan kelas `col-md-3`, yang memaksa sidebar berada di grid Bootstrap, membuatnya sulit untuk digunakan di berbagai perangkat dengan layar kecil.
3. ***Kurang Interaktif:*** Tidak ada visual cue yang cukup modern, seperti pemisah atau heading yang lebih menarik untuk membagi konten sidebar.

Untuk memperbaiki masalah tersebut, kita melakukan perubahan menggunakan komponen `offcanvas` dari Bootstrap. `offcanvas` memberikan kemampuan untuk menampilkan sidebar dengan animasi yang lebih halus dan juga mendukung pengalaman pengguna yang lebih baik pada layar kecil.

Berikut adalah kode setelah dilakukan perbaikan:

```
<!-- File: apps/templates/dashboard/sidebar.html -->
<div class="offcanvas offcanvas-start sidebar-nav bg-dark" tabindex="-1" id="sidebar">
  <div class="offcanvas-body p-0">
    <nav class="navbar-dark">
```


Penyempurnaan Halaman Dashboard

```
<ul class="navbar-nav">
  <li>
    <div class="text-muted small fw-bold
text-uppercase px-3">CORE</div>
  </li>
  <li>
    <a href="{% url 'dashboard' %}"
class="nav-link px-3 active">
      <span class="me-2"><i class="bi bi-
speedometer2"></i></span>
      <span>Dashboard</span>
    </a>
  </li>
  <li class="my-4"><hr class="dropdown-
divider bg-light"></li>
  <li>
    <div class="text-muted small fw-bold
text-uppercase px-3">BOTS</div>
  </li>
  <li>
    <a href="{% url 'create_bot' %}"
class="nav-link px-3">
      <span class="me-2"><i class="bi bi-
plus-circle-fill"></i></span>
      <span>Buat Bot</span>
    </a>
  </li>
  <li>
    <a href="{% url 'manage_bots' %}"
class="nav-link px-3">
      <span class="me-2"><i class="bi bi-
gear-wide-connected"></i></span>
      <span>Manage Bots</span>
    </a>
  </li>
  <li class="my-4"><hr class="dropdown-
divider bg-light"></li>
  <li>
```

Penyempurnaan Halaman Dashboard

```
        <div class="text-muted small fw-bold  
text-uppercase px-3 mb-3">Account</div>  
    </li>  
    <li>  
        <a href="{% url 'logout' %}"  
class="nav-link px-3 text-danger">  
            <span class="me-2"><i class="bi bi-  
box-arrow-right"></i></span>  
            <span>Logout</span>  
        </a>  
    </li>  
</ul>  
</nav>  
</div>  
</div>
```

Perubahan Utama dan Penjelasan

1. Menggunakan *offcanvas* dari *Bootstrap*:

- *offcanvas* memberikan pengalaman pengguna yang lebih interaktif, terutama untuk layar kecil. Ini memungkinkan sidebar ditampilkan dengan animasi yang lembut dari samping.
- `tabindex="-1"` digunakan agar sidebar tidak fokus secara otomatis ketika dimuat.
- Dengan menggunakan *offcanvas-body*, kita bisa memastikan konten sidebar akan responsif dan tidak memengaruhi tata letak halaman utama.

2. Tampilan yang Lebih Modern:

- **Pemisah:** Menambahkan `<hr class="dropdown-divider bg-light">` untuk memberikan pemisah visual an-

Penyempurnaan Halaman Dashboard

tar item navigasi, membantu pengguna untuk lebih mudah memahami struktur sidebar.

- **Heading Kecil:** Menambahkan teks heading kecil dengan kelas `text-muted small fw-bold text-uppercase px-3`, memberikan pembagian yang lebih jelas antara kategori di sidebar.

3. Ikon yang Lebih Deskriptif:

- Menggunakan ikon yang lebih mendeskripsikan fungsi dari setiap tautan, seperti `bi-speedometer` untuk dashboard dan `bi-plus-circle-fill` untuk tombol "Buat Bot", membuat sidebar lebih informatif.

4. Interaksi Logout:

- Tombol logout diberi warna teks merah menggunakan kelas `text-danger` agar lebih menonjol dan mudah ditemukan oleh pengguna.

Dengan perbaikan ini, sidebar tidak hanya menjadi lebih responsif dan mudah digunakan di layar kecil, tetapi juga memberikan tampilan yang lebih modern dan profesional. Struktur yang lebih teratur dan penggunaan komponen `offcanvas` dari Bootstrap membuat sidebar lebih fleksibel untuk berbagai ukuran layar tanpa mengorbankan fungsionalitasnya di desktop atau mobile.

14.2.4 Memperbaiki Navbar untuk Dashboard

Sekarang kita akan melanjutkan dengan memperbaiki navbar yang sudah kita pisahkan ke dalam file `navbar.html`. Kita

Penyempurnaan Halaman Dashboard

akan mengganti struktur navbar agar lebih modern dan fungsional. Navbar ini menggunakan Bootstrap 5 dan mencakup beberapa fitur seperti sidebar off-canvas, form pencarian, dan dropdown profil. Selain itu, kita akan menyimpan logo yang digunakan dalam navbar ke dalam folder `apps/static/assets/img/`.

Mari kita lihat struktur navbar yang baru:

```
<!-- File: templates/navbar.html -->
<!-- Navbar -->
<nav class="navbar navbar-expand-lg navbar-dark bg-dark fixed-top">
  <div class="container-fluid">
    <!-- Tombol Toggler untuk Sidebar Off-Canvas -->
    <button class="navbar-toggler" type="button" data-bs-toggle="offcanvas" data-bs-target="#sidebar" aria-controls="sidebar">
      <span class="navbar-toggler-icon"></span>
    </button>

    <!-- Logo dan Teks Navbar -->
    <a class="navbar-brand me-auto ms-lg-0 ms-3 text-uppercase fw-bold" href="{% url 'home' %}">
       Platform Bot
    </a>

    <!-- Tombol Toggler untuk Navbar -->
    <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#topNavBar" aria-controls="topNavBar" aria-expanded="false" aria-label="Toggle navigation">
      <span class="navbar-toggler-icon"></span>
```

Penyempurnaan Halaman Dashboard

```
</button>

<!-- Bagian Navbar yang Kolaps -->
<div class="collapse navbar-collapse"
id="topNavBar">
  <!-- Form Pencarian -->
  <form class="d-flex ms-auto my-3 my-lg-0">
    <div class="input-group">
      <input class="form-control"
type="search" placeholder="Search" aria-
label="Search" />
      <button class="btn btn-primary"
type="submit">
        <i class="bi bi-search"></i>
      </button>
    </div>
  </form>

  <!-- Dropdown Profil -->
  <ul class="navbar-nav">
    <li class="nav-item dropdown">
      <a class="nav-link dropdown-toggle ms-
2" href="#" role="button" data-bs-
toggle="dropdown" aria-expanded="false">
        <i class="bi bi-person-fill"></i>
      </a>
      <ul class="dropdown-menu dropdown-
menu-end">
        <li><a class="dropdown-item"
href="{% url 'profile' %}">Profile</a></li>
      </ul>
    </li>
  </ul>
</div>
</div>
</nav>
```

Navbar ini dibangun dengan beberapa elemen penting:

Penyempurnaan Halaman Dashboard

1. **Tombol Toggler untuk Sidebar Off-Canvas:** Mengontrol sidebar agar bisa muncul dari samping.
2. **Logo:** Menampilkan logo kita yang diambil dari folder `img`.
3. **Tombol Toggler untuk Navbar:** Digunakan untuk menampilkan atau menyembunyikan bagian navbar di perangkat kecil.
4. **Form Pencarian:** Terletak di sebelah kanan navbar untuk memudahkan pencarian.
5. **Dropdown Profil:** Berisi menu untuk mengakses profil dan fitur lain terkait akun pengguna.

Menyimpan Logo

Agar logo ditampilkan di navbar, kita harus menyimpan file gambar logo di dalam folder `img` yang berada di dalam struktur folder `static`. Kita bisa melakukan ini dengan menempatkan logo kita di:

```
apps/  
  static/  
    assets/  
      img/  
        logo.png
```

Dengan ini, ketika kita menggunakan `{{ ASSETS_ROOT }}/img/logo.png` di template, gambar logo kita akan diambil dari folder `static/assets/img/logo.png`.

Penjelasan Penempatan Logo

Penyempurnaan Halaman Dashboard

Dalam template `navbar.html`, logo didefinisikan dengan tag `` berikut:

```

```

Tag ini memastikan gambar logo ditampilkan dalam ukuran 30x24 piksel di bagian navbar. Atribut `src` merujuk pada lokasi gambar dalam folder `static`, dan `class="d-inline-block align-text-top"` memastikan bahwa logo sejajar secara vertikal dengan teks di sebelahnya.

Dengan memisahkan navbar ke dalam file `navbar.html` dan menyimpan logo di dalam folder `img`, kita telah meningkatkan keteraturan dan modularitas dari struktur kode kita. Ini akan mempermudah pemeliharaan proyek di masa depan dan memungkinkan kita mengembangkan fitur baru dengan lebih mudah.

14.2.5 Memperbaiki Halaman Dashboard

Pada bagian ini, kita akan membahas kekurangan dari struktur dashboard sebelumnya, perubahan yang telah dilakukan, serta penambahan elemen-elemen baru untuk memperkaya pengalaman pengguna. Kita juga akan membahas bagian *charts* yang masih memerlukan integrasi untuk berfungsi.

Kekurangan dari Kode Dashboard Sebelumnya

1. *Tampilan Kurang Dinamis dan Modern:*

Penyempurnaan Halaman Dashboard

- Sebelumnya, dashboard menampilkan elemen yang statis, seperti tabel log aktivitas dan pesan status. Namun, tidak ada elemen visual yang menarik seperti **cards** untuk memberikan gambaran umum dari data.
- **Jumbotron** di bagian atas menggunakan desain yang tidak memiliki elemen bayangan atau sudut yang membulat, sehingga terlihat sedikit kaku.

2. *Kurangnya Informasi Visual yang Komprehensif:*

- Dashboard sebelumnya tidak menyediakan cara yang baik untuk menampilkan informasi secara ringkas melalui visualisasi data seperti **charts** atau **cards**.
- Pengguna hanya disajikan tabel log aktivitas tanpa konteks yang lebih luas, sehingga pengalaman pengguna dalam memahami status atau perkembangan dari sistem terasa kurang.

3. *Pesan Error dan Sukses yang Tidak Terintegrasi dengan Baik:*

- Pesan error atau sukses hanya ditampilkan secara sederhana tanpa format yang menarik. Ini membuatnya sulit untuk menarik perhatian pengguna, terutama pada dashboard yang sibuk.

Berikut adalah kode yang lebih lengkap untuk memperbaiki dashboard agar lebih modern, lengkap dengan *cards*, *pesan status yang lebih serasi*, serta *placeholder untuk charts*. Ini juga termasuk integrasi dasar untuk pesan error atau sukses dan penyediaan template untuk *chart* yang siap dihubungkan dengan data di masa mendatang.

Penyempurnaan Halaman Dashboard

```
<!-- File:
apps/templates/dashboard/dashboard.html -->

{% extends 'dashboard/base.html' %}

{% block content %}
<div class="row mb-4">
  <div class="col-md-12 fw-bold fs-3">
    Dashboard
  </div>
</div>

<!-- Jumbotron Welcome Section -->
<div class="row mb-4">
  <div class="col-lg-12">
    <div class="jumbotron text-center bg-primary
text-white py-5 shadow-sm rounded">
      <h2>Selamat Datang di Dashboard,
{{ user_profile.user.username }}!</h2>
      <p>Di sini Anda dapat mengelola bot,
melihat statistik, dan memeriksa log aktivitas
terbaru Anda.</p>
    </div>
  </div>
</div>

<!-- Cards Section -->
<div class="row">
  <div class="col-md-3 mb-3">
    <div class="card bg-primary text-white h-100
shadow-sm">
      <div class="card-body d-flex align-items-
center justify-content-between">
        <span class="fs-5">Primary Card</span>
        <i class="bi bi-bar-chart-line fs-
3"></i>
      </div>
    </div>
  </div>
</div>
```

Penyempurnaan Halaman Dashboard

```
        <div class="card-footer d-flex align-items-center">
            <span>View Details</span>
            <span class="ms-auto">
                <i class="bi bi-chevron-right"></i>
            </span>
        </div>
    </div>
</div>

<div class="col-md-3 mb-3">
    <div class="card bg-warning text-dark h-100 shadow-sm">
        <div class="card-body d-flex align-items-center justify-content-between">
            <span class="fs-5">Warning Card</span>
            <i class="bi bi-exclamation-triangle-fill fs-3"></i>
        </div>
        <div class="card-footer d-flex align-items-center">
            <span>View Details</span>
            <span class="ms-auto">
                <i class="bi bi-chevron-right"></i>
            </span>
        </div>
    </div>
</div>

<div class="col-md-3 mb-3">
    <div class="card bg-success text-white h-100 shadow-sm">
        <div class="card-body d-flex align-items-center justify-content-between">
            <span class="fs-5">Success Card</span>
            <i class="bi bi-check-circle-fill fs-3"></i>
        </div>
    </div>
</div>
```

Penyempurnaan Halaman Dashboard

```
        <div class="card-footer d-flex align-items-center">
            <span>View Details</span>
            <span class="ms-auto">
                <i class="bi bi-chevron-right"></i>
            </span>
        </div>
    </div>
</div>

<div class="col-md-3 mb-3">
    <div class="card bg-danger text-white h-100 shadow-sm">
        <div class="card-body d-flex align-items-center justify-content-between">
            <span class="fs-5">Danger Card</span>
            <i class="bi bi-x-circle-fill fs-3"></i>
        </div>
        <div class="card-footer d-flex align-items-center">
            <span>View Details</span>
            <span class="ms-auto">
                <i class="bi bi-chevron-right"></i>
            </span>
        </div>
    </div>
</div>
</div>

<!-- Log Aktivitas Section -->
<div class="row">
    <div class="col-md-12 mb-3">
        <div class="card shadow-sm">
            <div class="card-header d-flex align-items-center">
                <span><i class="bi bi-clock-history me-2 text-primary"></i></span>
                <span class="fw-bold">Log Aktivitas</span>
            </div>
        </div>
    </div>
</div>
```

Penyempurnaan Halaman Dashboard

```

        </div>
        <div class="card-body">
            <div class="table-responsive">
                <table id="activity-log-table"
class="table table-striped table-hover data-
table" style="width: 100%">
                    <thead>
                        <tr>
                            <th>Waktu</th>
                            <th>Aktivitas</th>
                        </tr>
                    </thead>
                    <tbody>
                        {% for log in activity_logs %}
                        <tr>
                            <td>{{ log.timestamp|date:"d-m-Y
H:i" }}</td>
                            <td>{{ log.action }}</td>
                        </tr>
                        {% empty %}
                        <tr>
                            <td colspan="2" class="text-
center">Tidak ada log aktivitas ditemukan.</td>
                        </tr>
                        {% endfor %}
                    </tbody>
                </table>
            </div>
        </div>
    </div>
</div>
</div>
</div>

<!-- Placeholder for Charts -->
<div class="row">
    <div class="col-md-6 mb-3">
        <div class="card h-100 shadow-sm">
            <div class="card-header d-flex align-
items-center">

```

Penyempurnaan Halaman Dashboard

```
        <span class="me-2"><i class="bi bi-bar-chart-fill text-primary"></i></span>
        <span class="fw-bold">Chart
Aktifitas</span>
    </div>
    <div class="card-body">
        <canvas class="chart" width="400"
height="200" id="activity-chart" width="400"
height="200"></canvas>
    </div>
</div>

<div class="col-md-6 mb-3">
    <div class="card h-100 shadow-sm">
        <div class="card-header d-flex align-items-center">
            <span class="me-2"><i class="bi bi-bar-chart-fill text-primary"></i></span>
            <span class="fw-bold">Log
Aktivitas</span>
        </div>
        <div class="card-body">
            <canvas class="chart" width="400"
height="200" id="log-chart" width="400"
height="200"></canvas>
        </div>
    </div>
</div>
</div>

{% endblock %}
```

Untuk menghubungkan **charts** ini dengan data, kita perlu menggunakan JavaScript dan library chart, seperti **Chart.js**. Ini bisa ki-

Penyempurnaan Halaman Dashboard

ta integrasikan pada tahap berikutnya sesuai dengan data yang ingin ditampilkan.

Perubahan Utama yang Dilakukan pada Dashboard

Pada perbaikan dashboard ini, kami telah menambahkan beberapa komponen penting seperti ***cards***, ***pesan status yang lebih menarik***, dan ***chart placeholders***. Meskipun charts belum terintegrasi dengan data, placeholder ini memberikan gambaran visual yang lebih baik dari potensi data yang akan ditampilkan di masa depan.

1. ***Penambahan Cards:***

- Cards digunakan untuk menampilkan informasi penting secara cepat dan visual. Setiap card memiliki ikon dan footer yang menambah interaktivitas. Misalnya, ada ***primary card*** untuk menampilkan data primer, dan ***success card*** untuk menampilkan data sukses.
- Cards ini memberikan kesan dashboard yang lebih modern dan profesional, sekaligus membantu pengguna memahami data sekilas.

2. ***Pesan Error dan Sukses yang Lebih Serasi:***

- Pesan sukses dan error kini memiliki tampilan yang lebih menarik dan terintegrasi dengan baik. Setiap pesan ditampilkan menggunakan ***alert*** dengan ***tags*** untuk memperjelas status pesan, seperti sukses atau error. Ini membantu pengguna mengenali pesan penting dengan cepat.

Penyempurnaan Halaman Dashboard

- Visualisasi pesan sekarang lebih **harmonis dengan desain keseluruhan dashboard**, memastikan konsistensi dan pengalaman yang lebih baik.

3. **Tabel Log Aktivitas yang Lebih Responsif:**

- Tabel log aktivitas masih ditampilkan, namun sekarang dilengkapi dengan **header yang lebih menonjol** dan ditampilkan di dalam komponen card, membuat tampilannya lebih serasi dengan elemen lainnya di dashboard.
- Tabel ini juga telah dibuat **responsif** untuk memastikan tampilannya tetap baik di berbagai ukuran layar.

4. **Placeholder untuk Charts:**

- Meskipun saat ini charts belum terintegrasi dengan data apapun, dua placeholder **canvas** telah disiapkan untuk menampilkan **grafik aktivitas** dan **log aktivitas**.
- Penambahan charts ini dimaksudkan untuk memberi **ruang untuk visualisasi data** di masa depan, yang akan memperkaya informasi yang dapat disajikan dalam dashboard.
- Dengan **placeholder** tersebut, kita bisa mulai merencanakan integrasi data dengan menggunakan library seperti **Chart.js** atau **ApexCharts** di masa mendatang.

Mengintegrasikan Charts

Untuk mengintegrasikan charts di masa depan, langkah-langkah yang dapat diambil adalah:

Penyempurnaan Halaman Dashboard

1. *Pilih Library Chart yang Tepat:*

- Gunakan **Chart.js** atau **ApexCharts** untuk menambahkan visualisasi data ke dalam dashboard. Kedua library ini mendukung berbagai tipe grafik seperti **bar charts**, **line charts**, dan **pie charts**, yang sangat fleksibel untuk menampilkan data.

2. *Hubungkan Charts dengan Data:*

- Setelah library ditambahkan, hubungkan charts dengan data backend (seperti data statistik pengguna, performa bot, atau log aktivitas) yang diambil dari database. Ini bisa dilakukan melalui AJAX atau API yang disiapkan pada server Django.

3. *Interaktivitas dan Filtering Data:*

- Jika memungkinkan, tambahkan fitur interaktif seperti **filter waktu** atau **pilihan kategori** untuk memberikan kontrol lebih kepada pengguna mengenai data yang ingin mereka lihat.

Dengan desain yang diperbarui ini, dashboard menjadi lebih profesional, modern, dan siap untuk menampung lebih banyak data visual yang dapat membantu pengguna dalam mengelola aktivitas mereka di platform. Pesan yang lebih jelas dan menarik, serta penggunaan elemen interaktif seperti cards dan charts, menjadikan pengalaman pengguna jauh lebih baik dibandingkan sebelumnya.

14.3 Penyempurnaan Halaman Profile

Dalam memperbarui halaman profil `profile.html`, tujuan utama adalah untuk menciptakan tampilan yang lebih profesional dan menarik. Pertama, kami memperluas penggunaan komponen Bootstrap untuk memastikan tata letak yang responsif dan estetika yang lebih baik. Hal ini dilakukan dengan menambahkan ikon dari Bootstrap Icons untuk setiap label, memberikan visual yang lebih informatif dan menarik bagi pengguna.

Berikut kode yang sudah di perbaiki:

```

<!-- apps/templates/users/profile.html -->
{% extends "dashboard/base.html" %}

{% block content %}
<div class="content">
  <div class="row">
    <div class="col-md-8">
      <div class="card bg-light shadow-sm">
        <div class="card-header bg-primary text-white">
          <h5 class="title">
            <i class="bi bi-pencil-square"></i> Edit Profile
          </h5>
        </div>
        <div class="card-body">
          <form method="post"
            enctype="multipart/form-data">
            {% csrf_token %}

            <!-- Username dan Email -->

            <div class="row mb-3">

```

Penyempurnaan Halaman Dashboard

```

        <div class="col-md-6">
            <div
class="form-group">
                <label
class="fw-bold"><i class="bi bi-person-
fill"></i> Username</label>
                {{ form.username }}
            </div>
        </div>
        <div class="col-md-6">
            <div
class="form-group">
                <label
class="fw-bold"><i class="bi bi-envelope-
fill"></i> Email</label>
                {{ form.email }}
            </div>
        </div>
        </div>
        <!-- First Name dan Last
Name -->
        <div class="row mb-3">
            <div class="col-md-6">
                <div
class="form-group">
                    <label
class="fw-bold"><i class="bi bi-file-
person"></i> First Name</label>
                    {{ form.first_name }}
                </div>
            </div>
        </div>
    </div>

```

Penyempurnaan Halaman Dashboard

```

        <div class="col-md-
6">
            <div
class="form-group">
                <label
class="fw-bold"><i class="bi bi-file-person-
fill"></i> Last Name</label>
                {{ form.last_name }}
            </div>
        </div>
    </div>

    <!-- Company -->
    <div class="form-group
mb-3">
        <label class="fw-
bold"><i class="bi bi-building"></i>
Company</label>
        {{ form.company }}
    </div>

    <!-- Address -->
    <div class="form-group
mb-3">
        <label class="fw-
bold"><i class="bi bi-house"></i>
Address</label>
        {{ form.address }}
    </div>

    <!-- City, Country,
Postal Code -->
    <div class="row mb-3">
        <div class="col-md-
4">
            <div
class="form-group">

```

Penyempurnaan Halaman Dashboard

```

                                <label
class="fw-bold"><i class="bi bi-geo-alt-
fill"></i> City</label>
                                {{ form.city
}}

                                </div>
                                </div>
                                <div class="col-md-
4">

                                <div
class="form-group">
                                <label
class="fw-bold"><i class="bi bi-globe2"></i>
Country</label>
                                {{ form.country }}

                                </div>
                                </div>
                                <div class="col-md-
4">

                                <div
class="form-group">
                                <label
class="fw-bold"><i class="bi bi-envelope"></i>
Postal Code</label>
                                {{ form.postal_code }}

                                </div>
                                </div>
                                </div>

                                <!-- About Me -->
                                <div class="form-group
mb-3">

                                <label class="fw-
bold"><i class="bi bi-info-circle"></i> About
Me</label>
                                {{ form.about }}

```

```

        </div>

        <!-- Profile Picture -->
        <div class="form-group"
mb-3">
            <label class="fw-
bold"><i class="bi bi-camera"></i> Profile
Picture</label>
            {{ form.foto_profile
}}

        </div>

        <!-- Submit Button -->
        <button type="submit"
class="btn btn-fill btn-success">
            <i class="bi bi-
save"></i> Save
        </button>
    </form>
</div>
</div>
</div>

<!-- Bagian Preview Profile -->
<div class="col-md-4">
    <div class="card card-user text-
center shadow-sm">
        <div class="card-body">
            <div class="author">
                
                <h5 class="title text-
center">{{ form.username.value }}</h5>
            </div>

```

Penyempurnaan Halaman Dashboard

```
text-center">
    <div class="card-description
    {{ form.about.value }}
    </div>
  </div>
</div>
</div>
</div>
</div>
</div>
{% endblock %}
```

Bagian pertama yang diperhatikan adalah struktur form. Formulir diatur agar setiap label berada di atas input field yang sesuai, dengan menggunakan kelas form-group dari Bootstrap. Ini tidak hanya menciptakan spasi yang konsisten, tetapi juga meningkatkan keterbacaan. Setiap label ditandai dengan kelas fw-bold, sehingga lebih menonjol dan memudahkan pengguna dalam mengidentifikasi setiap field yang harus diisi.

Pada bagian input, kelas form-control diterapkan pada semua elemen input, termasuk TextInput, EmailInput, dan Textarea. Ini memberikan tampilan yang seragam dan meningkatkan pengalaman pengguna saat berinteraksi dengan formulir. Placeholder juga ditambahkan untuk memberikan petunjuk lebih lanjut tentang informasi yang harus dimasukkan.

Selanjutnya, tata letak form diorganisir dengan baik ke dalam beberapa baris. Misalnya, field untuk username dan email dikelompokkan dalam satu baris, sementara field untuk nama depan dan nama belakang dikelompokkan dalam baris berikutnya. Ini tidak hanya menciptakan struktur yang lebih jelas, tetapi juga memaksimalkan penggunaan ruang layar, terutama pada perangkat yang lebih kecil.

Penyempurnaan Halaman Dashboard

Selain itu, kami menambahkan ikon untuk setiap label, yang berfungsi sebagai elemen visual tambahan yang menarik perhatian pengguna. Ikon tersebut memberikan konteks lebih lanjut tentang informasi yang diminta, seperti ikon rumah untuk alamat dan ikon globe untuk negara. Penambahan ini menjadikan formulir lebih intuitif dan memudahkan pengguna dalam memahami tujuan setiap field.

Terakhir, pada bagian tombol simpan, kami memberikan kelas `btn-success` untuk menonjolkan tombol dengan warna yang cerah dan menarik, sehingga mengundang pengguna untuk menekan tombol tersebut setelah mengisi formulir. Ikon disertakan di dalam tombol untuk menambah daya tarik visual.

Dengan semua perubahan ini, halaman profil `profile.html` kini tidak hanya lebih menarik secara visual, tetapi juga lebih fungsional dan ramah pengguna, menciptakan pengalaman yang lebih baik bagi pengguna dalam mengedit profil mereka.

Kesimpulan Bab

Pada Bab ini membahas *penyempurnaan halaman dashboard* dalam proyek Django dengan fokus pada penyempurnaan struktur, tampilan, dan fungsionalitas dashboard. Berikut adalah ringkasan dari setiap subbab:

1. *Memisahkan Struktur Direktori Template*

- ***Membuat Template base.html***: Struktur template `base.html` dirancang untuk menjadi fondasi dari seluruh layout dashboard. Template ini mengatur kerangka dasar yang mencakup elemen-elemen seperti header, footer, dan ruang untuk konten dinamis.
- ***Membuat Template navbar.html***: Template `navbar.html` menyimpan kode untuk navbar, yang merupakan elemen navigasi utama di dashboard. Navbar ini memastikan bahwa navigasi antara berbagai bagian dashboard menjadi mudah dan terstruktur.
- ***Membuat Template sidebar.html***: Template `sidebar.html` mengatur sidebar yang berfungsi sebagai navigasi tambahan untuk fitur-fitur dashboard. Sidebar ini dirancang untuk menampilkan opsi-opsi menu yang sering digunakan.
- ***Membuat Template dashboard.html***: Template `dashboard.html` adalah tempat di mana konten spesifik halaman dashboard ditampilkan. Template ini mengintegrasikan berbagai komponen dan menyediakan ruang untuk menampilkan data dan informasi utama.

Penyempurnaan Halaman Dashboard

2. *Memperbaiki Halaman*

- ***Menyiapkan CSS dan JS untuk Dashboard:*** Kode CSS dan JavaScript dikonfigurasi untuk meningkatkan fungsionalitas dan desain dashboard. Ini melibatkan penyesuaian gaya dan skrip untuk mendukung elemen-elemen seperti grafik, tabel, dan interaksi dinamis.
- ***Memperbaiki Template base.html untuk Dashboard:*** Template `base.html` diperbarui untuk mencerminkan desain dan struktur yang lebih baik. Perubahan termasuk penambahan link ke stylesheet dan skrip yang mendukung tampilan dan fungsi dashboard.
- ***Memperbaiki Sidebar untuk Dashboard:*** Sidebar diperbarui untuk memastikan tampilannya sesuai dengan tema dashboard dan menyediakan navigasi yang efisien. Perubahan ini termasuk pengaturan ulang komponen dan penyesuaian gaya agar lebih konsisten.
- ***Memperbaiki Navbar untuk Dashboard:*** Navbar disesuaikan untuk menyesuaikan dengan desain dan kebutuhan spesifik dashboard. Perubahan ini memastikan bahwa navbar berfungsi dengan baik dan memiliki tampilan yang konsisten.
- ***Meningkatkan Tampilan Dashboard:*** Tampilan dashboard ditingkatkan dengan menambahkan komponen visual seperti cards dan charts, serta memperbaiki desain agar lebih menarik dan fungsional. Ini juga mencakup pengujian untuk

Penyempurnaan Halaman Dashboard

memastikan bahwa tampilan responsif dan user-friendly.

Dengan mengikuti langkah-langkah ini, halaman dashboard dapat disempurnakan secara menyeluruh, menawarkan pengalaman pengguna yang lebih baik dan desain yang lebih modern. Fokus pada pemisahan struktur template dan perbaikan elemen-elemen individual memastikan bahwa dashboard tidak hanya terlihat profesional tetapi juga mudah digunakan dan dinamis.

BAB 15 -Penyempurnaan Halaman Bots

Pada Bagian ini kita akan fokus pada penyempurnaan halaman terkait bot dalam aplikasi Django, yang meliputi halaman untuk membuat, mengelola, dan mengedit bot serta menambah dan mengelola perintah.

Fokus utama adalah meningkatkan antarmuka dan fungsionalitas dari halaman-halaman berikut:

- **Halaman Create Bot:** Tempat di mana pengguna dapat membuat bot baru dengan mengisi formulir.
- **Halaman Manage Bots:** Tempat di mana pengguna dapat melihat daftar bot yang telah dibuat, mengelola, dan melakukan tindakan seperti mengedit atau menghapus bot.
- **Halaman Edit Bot:** Tempat di mana pengguna dapat memperbarui informasi bot yang sudah ada.
- **Halaman Add Command:** Tempat di mana pengguna dapat menambahkan perintah baru ke bot.
- **Halaman Edit Command:** Tempat di mana pengguna dapat mengedit perintah yang sudah ada.

Untuk masing-masing halaman, kita akan memperhatikan peningkatan desain dan fungsionalitas dengan memanfaatkan komponen Bootstrap. Modifikasi ini bertujuan untuk memberikan

Penyempurnaan Halaman Bots

tampilan yang lebih profesional dan pengalaman pengguna yang lebih baik, tanpa perlu menulis CSS tambahan.

Berikut adalah struktur yang akan dibahas dan diperbaiki dalam bab ini:

```
apps/  
├── templates/  
│   └── bots/  
│       ├── create_bot.html  
│       ├── manage_bots.html  
│       ├── edit_bot.html  
│       ├── add_command.html  
│       └── edit_command.html
```

Mari kita mulai dengan pembahasan dan perbaikan pada masing-masing halaman.

15.1 Memperbaiki Halaman Create Bot

Halaman Create Bot adalah fitur penting dalam aplikasi platform_bot, memungkinkan pengguna untuk membuat bot baru dengan memasukkan nama, token, dan memilih platform bot. Desain halaman ini harus responsif, mudah diakses, dan memberikan pengalaman pengguna yang baik. Untuk menyempurnakan halaman ini, kami akan menerapkan beberapa perubahan signifikan yang memanfaatkan komponen Bootstrap dan menambahkan elemen untuk menangani pesan kesalahan.

Peningkatan Desain Halaman

Penyempurnaan Halaman Bots

Pada versi sebelumnya, formulir di halaman ini sudah menggunakan beberapa komponen Bootstrap. Namun, kita dapat meningkatkan tampilannya dengan menambahkan beberapa elemen baru seperti ikon, warna, dan desain yang lebih interaktif. Juga, kita akan menambahkan modal untuk menampilkan pesan kesalahan jika ada.

Perubahan pada Kode

1. *Penambahan Ikon dan Warna:*

- Menambahkan ikon pada label formulir untuk memberikan konteks visual yang lebih baik.
- Menggunakan warna Bootstrap untuk batas dan latar belakang formulir.

2. *Peningkatan Tampilan Formulir:*

- Mengubah tombol submit menjadi tombol besar dengan ikon untuk meningkatkan visibilitas.
- Menambahkan placeholder pada input untuk memberikan petunjuk tambahan kepada pengguna.

3. *Modal Pesan Kesalahan:*

- Menambahkan modal Bootstrap untuk menampilkan pesan kesalahan jika terjadi masalah saat mengirim formulir.
- Modal ini akan muncul secara otomatis jika ada kesalahan pada formulir.

Kode Lengkap setelah Penyempurnaan

```
<!-- File: apps/templates/bots/create_bot.html -->
```

Penyempurnaan Halaman Bots

```
{% extends 'dashboard/base.html' %}

{% block content %}
<div class="container mt-4">
    <!-- Card untuk Formulir -->
    <div class="card shadow-lg" style="border-
radius: 15px;">
        <div class="card-header text-center bg-
primary text-white" style="border-radius: 15px
15px 0 0;">
            <h5 class="card-title"><i class="bi
bi-bot-fill"></i> Buat Bot Baru</h5>
        </div>
        <div class="card-body">
            <!-- Form untuk Bot -->
            <form method="post">
                {% csrf_token %}

                <!-- Form Group untuk Platform
(muncul pertama) -->
                <div class="form-group mb-4">
                    <label class="form-label fw-
bold">

                        <i class="bi bi-device-
screenshot"></i> Platform
                    </label>
                    <div class="btn-group d-flex
justify-content-center" role="group" aria-
label="Platform">

                        <input type="radio"
name="bot_type" id="telegram" value="telegram"
{% if form.bot_type.value == 'telegram'
%}><checked{% endif %}> onclick="toggleFields()"
class="btn-check">

                            <label class="btn btn-
outline-primary" for="telegram"><i class="bi bi-
telegram"></i> Telegram</label>
                    
```

Penyempurnaan Halaman Bots

```

                                <input type="radio"
name="bot_type" id="whatsapp" value="whatsapp"
{% if form.bot_type.value == 'whatsapp'
%}checked{% endif %} onclick="toggleFields()"
class="btn-check">
                                <label class="btn btn-
outline-primary" for="whatsapp"><i class="bi bi-
whatsapp"></i> WhatsApp</label>
                                </div>
                                </div>

                                <!-- Form Group untuk Nama -->
                                <div class="form-group mb-4">
                                    <label
for="{{ form.name.id_for_label }}" class="fw-
bold">
                                    <i class="bi bi-chat-
dots"></i> Nama
                                    </label>
                                    <input type="text"
name="name" id="{{ form.name.id_for_label }}"
value="{{ form.name.value }}" class="form-
control {% if form.name.errors %}is-invalid{%
endif %}">
                                    {% for error in
form.name.errors %}
                                    <div class="invalid-
feedback">{{ error }}</div>
                                    {% endfor %}
                                    </div>

                                <!-- Form Group untuk Token
(hanya untuk Telegram) -->
                                <div class="form-group mb-4"
id="token-field">
                                    <label
for="{{ form.token_telegram.id_for_label }}"
class="fw-bold">
```

Penyempurnaan Halaman Bots

```

                                <i class="bi bi-lock-
fill"></i> Token
                                </label>
                                <input type="text"
name="token_telegram"
id="{{ form.token_telegram.id_for_label }}"
value="{{ form.token_telegram.value }}"
class="form-control {% if
form.token_telegram.errors %}is-invalid{% endif
%}">
                                {% for error in
form.token_telegram.errors %}
                                <div class="invalid-
feedback">{{ error }}</div>
                                {% endfor %}
                                </div>

                                <!-- Button Submit -->
                                <div class="text-center">
                                <button type="submit"
class="btn btn-primary btn-lg">
                                <i class="bi bi-bot-
fill"></i> Buat Bot
                                </button>
                                </div>
                                </form>
                                </div>
                                </div>

                                <!-- Modal untuk Pesan Error -->
                                {% if form.errors %}
                                <div class="modal fade" id="errorModal"
tabindex="-1" aria-labelledby="errorModalLabel"
aria-hidden="true">
                                <div class="modal-dialog">
                                <div class="modal-content">
                                <div class="modal-header bg-
danger text-white">

```


Penyempurnaan Halaman Bots

```
        <h1 class="modal-title fs-5"
id="errorModalLabel">Terjadi Kesalahan</h1>
        <button type="button"
class="btn-close" data-bs-dismiss="modal" aria-
label="Close"></button>
    </div>
    <div class="modal-body">
        <div class="alert alert-
danger">
            {% for field in form %}
                {% for error in
field.errors %}
                    <p>{{ error
}}</p>
                {% endfor %}
            {% endfor %}
            {% for error in
form.non_field_errors %}
                <p>{{ error }}</p>
            {% endfor %}
        </div>
    </div>
    <div class="modal-footer">
        <button type="button"
class="btn btn-secondary" data-bs-
dismiss="modal">Tutup</button>
    </div>
</div>
</div>

<script>
    // Pastikan bahwa Bootstrap JavaScript
sudah di-load dan jQuery jika diperlukan

document.addEventListener('DOMContentLoaded',
function () {
    if
(document.querySelector('#errorModal')) {
```

Penyempurnaan Halaman Bots

```
        var errorModal = new
bootstrap.Modal(document.getElementById('errorMo
dal'));
        errorModal.show();
    }
    });
</script>
{% endif %}
</div>

<script>
    // Fungsi untuk menyembunyikan/menampilkan
    elemen sesuai platform
    function toggleFields() {
        var tokenField =
document.getElementById('token-field');
        var whatsappFields =
document.getElementById('whatsapp-fields');

        if
(document.getElementById('telegram').checked) {
            tokenField.classList.remove('d-
none');
            whatsappFields?.classList.add('d-
none');
        } else if
(document.getElementById('whatsapp').checked) {
            tokenField.classList.add('d-none');
            whatsappFields?.classList.remove('d-
none');
        }
    }

    // Panggil toggleFields() saat halaman
    dimuat

document.addEventListener('DOMContentLoaded',
function () {
    toggleFields();
});
```

Penyempurnaan Halaman Bots

```
});  
</script>  
{% endblock %}
```

Penjelasan Perubahan:

- **Header Card:** Ditambahkan ikon bot di judul card untuk memberikan kesan visual yang lebih menarik.
- **Platform Selection:** Menggunakan tombol radio yang lebih stylish dengan ikon untuk platform Telegram dan WhatsApp.
- **Label dan Input:** Setiap label sekarang memiliki ikon terkait untuk meningkatkan daya tarik visual.
- **Button Submit:** Ikon ditambahkan pada tombol untuk membuatnya lebih informatif dan menarik.
- **Modal Error:** Dibiarkan seperti semula, namun dengan gaya dan fungsi yang konsisten.

Dengan perbaikan ini, tampilan halaman `create_bot.html` akan lebih profesional, modern, dan menarik.

Kekurangan Kode Sebelumnya

Kode sebelumnya sudah menggunakan komponen dasar Bootstrap tetapi tidak memanfaatkan sepenuhnya kemampuan desain yang ditawarkan oleh Bootstrap, seperti ikon dan warna yang bisa memperbaiki pengalaman pengguna. Selain itu, tidak ada penanganan untuk menampilkan pesan kesalahan, yang penting untuk memberikan umpan balik kepada pengguna jika formulir tidak berhasil dikirim. Penambahan modal untuk pesan kesalahan

Penyempurnaan Halaman Bots

membantu memastikan bahwa pengguna diberitahu tentang masalah dengan cara yang jelas dan terstruktur.

15.2 Memperbaiki Halaman Manage Bots

Halaman Manage Bots memberikan pengguna kemampuan untuk melihat dan mengelola daftar bot yang telah ada. Halaman ini harus didesain agar mudah dinavigasi dan memberikan informasi yang jelas tentang status dan tindakan yang dapat dilakukan pada setiap bot.

Peningkatan Desain Halaman

Pada versi sebelumnya, halaman ini menggunakan tabel sederhana untuk menampilkan data bot. Meskipun cukup fungsional, tampilan ini dapat diperbaiki lebih lanjut untuk meningkatkan pengalaman pengguna dengan menambahkan beberapa elemen desain tambahan dan memperbaiki tampilan tabel.

Perubahan pada Kode

1. *Penambahan Card dan Header:*

- Menggunakan komponen card Bootstrap untuk memberikan desain yang lebih menarik dan terstruktur.
- Menambahkan ikon pada header card untuk memberikan konteks visual yang lebih baik.

2. *Tabel Responsif dan Hover:*

Penyempurnaan Halaman Bots

- Menggunakan `table-responsive` untuk memastikan tabel dapat menyesuaikan dengan berbagai ukuran layar.
- Menambahkan kelas `table-hover` untuk memberikan efek hover pada baris tabel, meningkatkan keterbacaan dan interaksi pengguna.

3. *Ikon dan Badge pada Kolom Platform dan Status:*

- Menggunakan ikon untuk menunjukkan platform bot dengan warna yang sesuai.
- Menggunakan badge Bootstrap untuk menampilkan status bot (Active/Inactive) dengan warna yang berbeda.

4. *Tombol Edit dan Delete yang Lebih Informatif:*

- Mengubah tombol Edit dan Delete untuk menyertakan ikon yang sesuai, meningkatkan visibilitas dan pemahaman fungsi tombol.

Kode Lengkap setelah Penyempurnaan

```
<!-- File: apps/templates/bots/manage_bots.html -->

{% extends 'dashboard/base.html' %}

{% block content %}
<div class="container my-5">
  <!-- Header Section -->
  <div class="bg-primary text-white text-center p-4 rounded mb-4">
    <h1 class="display-6"><i class="bi bi-robot"></i> Manage Your Bots</h1>
  </div>
</div>
```

Penyempurnaan Halaman Bots

```
<p class="lead">Easily manage, edit, and
monitor your Telegram and WhatsApp bots</p>
</div>
<!-- Menampilkan pesan error atau sukses -->
{% if messages %}
  <div class="messages">
    {% for message in messages %}
      <div class="alert {{ message.tags }}
shadow-sm rounded">
        {{ message }}
      </div>
    {% endfor %}
  </div>
{% endif %}

<!-- Bots Cards -->
<div class="row">
  {% for bot in bots %}
    <div class="col-md-6 col-lg-4 mb-4">
      <div class="card shadow-sm h-100">
        <div class="card-header bg-
primary text-white d-flex justify-content-
between align-items-center">
          <h5 class="mb-0">
            <i class="bi bi-
robot"></i> {{ bot.name }}
          </h5>
          <span class="badge
{{ bot.is_active|yesno:'bg-success,bg-danger'
}}">
            {{ bot.is_active|
yesno:'Active,Inactive' }}
          </span>
        </div>

        <div class="card-body">
          <!-- Platform -->
          <p><strong>Platform:</strong>
```

Penyempurnaan Halaman Bots

```
{% if bot.bot_type ==
'telegram' %}
    <i class="bi bi-telegram
text-info"></i> Telegram
    {% elif bot.bot_type ==
'whatsapp' %}
    <i class="bi bi-whatsapp
text-success"></i> WhatsApp
    {% endif %}
</p>

<!-- Credentials (hanya
untuk Telegram) -->
    {% if bot.bot_type ==
'telegram' %}

<p><strong>Credentials:</strong></p>
        <ul class="list-unstyled">
            <li><i class="bi bi-key-
fill"></i> <strong>Token:</strong>
{{ bot.token_telegram }}</li>
        </ul>
        {% endif %}

    <!-- Commands -->

<p><strong>Commands:</strong></p>
    <table class="table table-sm
table-bordered">
        <thead>
            <tr>
                <th>Command</th>
                <th>Response</th>
                <th>Actions</th>
            </tr>
        </thead>
        <tbody>
```

Penyempurnaan Halaman Bots

```
bot.commands.all %}
                                {% for command in
                                <tr>

<td><strong>{{ command.command }}</strong></td>
<td>{{ command.response }}</td>
                                <td>
                                <div
                                class="d-flex gap-2">
                                <a
                                href="{% url 'edit_command' command.id %}"
                                class="btn btn-warning btn-sm">
                                <i
                                class="bi bi-pencil"></i> Edit
                                </a>
                                <form
                                method="post" style="display:inline;">
                                {%
                                csrf_token %}

                                <button type="submit" name="delete_command"
                                value="{{ command.id }}" class="btn btn-danger
                                btn-sm" onclick="return confirm('Are you sure
                                you want to delete this command?');">

                                <i class="bi bi-trash"></i> Delete

                                </button>
                                </form>
                                </div>
                                </td>
                                </tr>
                                {% empty %}
                                <tr>
                                <td colspan="3"
                                class="text-center">No commands available.</td>
                                </tr>
                                {% endfor %}
```


Penyempurnaan Halaman Bots

```
        </tbody>
    </table>

    <!-- Add Command Button -->
    <a href="{% url
'add_command' bot.id %}" class="btn btn-success
btn-sm mt-2">
        <i class="bi bi-plus-
circle"></i> Add Command
    </a>
</div>

    <!-- Card Footer: Actions -->
    <div class="card-footer bg-
light">
        <div class="d-flex justify-
content-between">
            <a href="{% url
'edit_bot' bot.id %}" class="btn btn-primary
btn-sm">
                <i class="bi bi-
pencil"></i> Edit Bot
            </a>
            <form method="post"
style="display:inline;">
                {% csrf_token %}
                <button
type="submit" name="delete" value="{ { bot.id } }"
class="btn btn-danger btn-sm" onclick="return
confirm('Are you sure you want to delete this
bot?');">
                    <i class="bi bi-
trash"></i> Delete
                </button>
            </form>
        </div>
    </div>
</div>
</div>
```

Penyempurnaan Halaman Bots

```
{% empty %}
<div class="col-12">
  <div class="alert alert-warning
text-center" role="alert">
    No bots available.
  </div>
</div>
{% endfor %}
</div>

<!-- Button to Create New Bot -->
<div class="text-center mt-4">
  <a href="{% url 'create_bot' %}"
class="btn btn-lg btn-primary">
    <i class="bi bi-plus-circle"></i>
Create New Bot </a>

</div>
</div>
{% endblock %}
```

Penjelasan Perbaikan:

1. **Header Section:** Ditambahkan penekanan menggunakan ikon `<i class="bi bi-robot"></i>` untuk memperkuat tema pengelolaan bot.
2. **Cards yang Lebih Informasional:** Setiap bot ditampilkan dalam card dengan informasi yang jelas, termasuk platform dan status aktif/nonaktif. WhatsApp tidak memiliki kredensial (seperti yang diinginkan), sehingga hanya Telegram yang menampilkan token.
3. **Tabel Command:** Struktur tabel commands lebih teratur dengan header yang lebih jelas dan penggunaan ikon Bootstrap untuk aksi edit dan delete.

Penyempurnaan Halaman Bots

4. **Button dengan Ikon Bootstrap:** Semua aksi seperti edit, delete, dan add command menggunakan ikon Bootstrap agar terlihat lebih profesional dan interaktif.
5. **Tampilan Bot:** Penggunaan badge berwarna untuk menunjukkan status aktif atau tidak aktif dari bot.

Ini membantu membuat halaman `manage_bots.html` terlihat lebih profesional dan terorganisir, sesuai dengan preferensi styling menggunakan Bootstrap tanpa tambahan CSS.

Kekurangan Kode Sebelumnya

Kode sebelumnya menggunakan tabel dasar tanpa desain tambahan yang memudahkan navigasi dan pengelolaan. Dengan menggunakan card, tabel responsif, dan ikon, desain yang baru memberikan pengalaman pengguna yang lebih baik. Selain itu, penambahan badge untuk status dan ikon untuk platform membuat informasi lebih mudah dipahami dan tabel lebih menarik. Tombol aksi dengan ikon juga meningkatkan keterbacaan dan aksesibilitas.

15.3 Memperbaiki Halaman Edit Bot

Halaman Edit Bot adalah tempat di mana pengguna dapat memperbarui detail bot yang telah ada. Untuk memastikan halaman ini efektif dan mudah digunakan, kita perlu memperbaiki tampilan formulir dan daftar perintah bot dengan memberikan elemen desain yang lebih baik serta memperbaiki struktur halaman.

Penyempurnaan Halaman Bots

Peningkatan Desain Halaman

Dalam versi sebelumnya, halaman Edit Bot menggunakan formulir dasar dan tabel sederhana. Untuk meningkatkan pengalaman pengguna, kita dapat menambahkan beberapa elemen desain, seperti card untuk tampilan yang lebih terstruktur, ikon untuk memperjelas fungsi, dan menata ulang elemen untuk memastikan kemudahan penggunaan.

Perubahan pada Kode

1. *Formulir Edit Bot:*

- Menggunakan card untuk membungkus formulir agar lebih terstruktur dan menarik.
- Menambahkan ikon dan label pada setiap grup formulir untuk memberikan konteks yang jelas.
- Menggunakan kelas Bootstrap untuk membuat formulir lebih menarik dan responsif.

2. *Penataan Formulir:*

- Menyusun elemen formulir seperti `bot_type`, `token`, `name`, `description`, `start_message`, dan `help_message` dengan menggunakan grup formulir dan kelas Bootstrap.

Kode Lengkap setelah Penyempurnaan

```
{% extends 'dashboard/base.html' %}

{% block content %}
<div class="container mt-4">
  <div class="card shadow-sm border-0">
```

Penyempurnaan Halaman Bots

```
<div class="card-header bg-primary text-
white">
    <h5 class="card-title">
        <i class="bi bi-pencil-
square"></i> Edit Bot: {{ bot.name }}
    </h5>
</div>
<div class="card-body bg-light">
    <form method="post">
        {% csrf_token %}

        <!-- Bot Name -->
        <div class="mb-3">
            <label for="id_name"
class="form-label"><i class="bi bi-robot"></i>
Bot Name</label>
            {{ form.name }}
        </div>

        <!-- Bot Token (only for
Telegram bots) -->
        {% if bot.bot_type == 'telegram'
%}

            <div class="mb-3">
                <label
for="id_token_telegram" class="form-label"><i
class="bi bi-key-fill"></i> Telegram
Token</label>
                {{ form.token_telegram }}
            </div>
            {% elif bot.bot_type ==
'whatsapp' %}

                <!-- WhatsApp Bot Token Fields
would go here -->
                {% endif %}

            <!-- Start Message -->
            <div class="mb-3">
```

Penyempurnaan Halaman Bots

```
        <label
for="id_start_message" class="form-label"><i
class="bi bi-play-circle"></i> Start
Message</label>
        {{ form.start_message }}
    </div>

    <!-- Help Message -->
    <div class="mb-3">
        <label for="id_help_message"
class="form-label"><i class="bi bi-info-
circle"></i> Help Message</label>
        {{ form.help_message }}
    </div>

    <!-- Description -->
    <div class="mb-3">
        <label for="id_description"
class="form-label"><i class="bi bi-file-earmark-
text"></i> Description</label>
        {{ form.description }}
    </div>

    <!-- Submit and Back Buttons -->
    <div class="d-flex justify-
content-between mt-4">
        <button type="submit"
class="btn btn-success">
            <i class="bi bi-check-
circle"></i> Simpan Perubahan
        </button>
        <a href="{% url
'manage_bots' %}" class="btn btn-secondary">
            <i class="bi bi-arrow-
left-circle"></i> Kembali
        </a>
    </div>
</form>
</div>
```

Penyempurnaan Halaman Bots

```
</div>  
</div>  
{% endblock %}
```

Penjelasan Perubahan

1. *Formulir Edit Bot:*

- `card mb-4 shadow-sm` digunakan untuk membungkus formulir dalam card dengan bayangan untuk tampilan yang lebih bersih.
- Menambahkan ikon pada label formulir (`bi bi-tag`, `bi bi-key`, `bi bi-person`, `bi bi-info-circle`, `bi bi-chat-dots`, `bi bi-question-circle`) untuk meningkatkan pemahaman.
- Menggunakan grup tombol radio (`btn-group`) untuk pilihan `bot_type` dengan ikon yang relevan, memastikan tampilan yang lebih menarik dan interaktif.

2. *Penataan Formulir:*

- Formulir disusun dengan grup formulir yang terpisah untuk setiap elemen, menggunakan kelas Bootstrap (`form-control`, `border border-primary`) untuk styling yang lebih baik.
- `btn btn-success` untuk tombol submit dengan ikon penyimpanan (`bi bi-save`), meningkatkan keterbacaan dan fungsi tombol.

Kekurangan Kode Sebelumnya

Penyempurnaan Halaman Bots

Kode sebelumnya menggunakan formulir dan tabel sederhana tanpa desain tambahan. Dengan menggunakan card dan styling yang lebih baik, tampilan halaman Edit Bot menjadi lebih terstruktur dan menarik. Penambahan ikon dan penggunaan kelas Bootstrap membuat formulir dan daftar perintah lebih mudah digunakan dan dipahami, serta meningkatkan pengalaman pengguna secara keseluruhan.

15.4 Memperbaiki Halaman Add Command

Pada halaman ini, pengguna dapat menambahkan perintah baru untuk bot. Untuk meningkatkan pengalaman pengguna dan estetika halaman, kita perlu melakukan beberapa perbaikan desain. Berikut adalah cara kita memperbaiki tampilan dan fungsionalitas halaman Add Command dengan menggunakan elemen desain yang lebih modern dan menarik.

Peningkatan Desain Halaman

Dalam versi sebelumnya, halaman Add Command menggunakan card dengan desain dasar dan elemen formulir yang sederhana. Untuk meningkatkan tampilan, kita dapat menambahkan ikon, menggunakan kelas Bootstrap untuk styling yang lebih baik, dan memberikan placeholder pada elemen formulir untuk membantu pengguna memahami apa yang harus diisi.

Perubahan pada Kode

1. *Desain Card:*

Penyempurnaan Halaman Bots

- Menggunakan kelas `shadow-sm` untuk memberikan bayangan halus pada card agar lebih menonjol.
- Mengubah warna header card menjadi biru dengan kelas `bg-primary` dan `text-white` untuk kontras yang lebih baik.

2. *Formulir:*

- Menambahkan ikon pada label formulir menggunakan `bi bi-terminal` untuk Command dan `bi bi-chat-dots` untuk Response.
- Menggunakan kelas `border border-primary` untuk memberikan border pada input dan textarea, meningkatkan visibilitas elemen formulir.
- Menambahkan placeholder pada input dan textarea untuk memberikan petunjuk kepada pengguna tentang data yang harus diisi.
- Mengubah tombol submit menjadi `btn btn-success` dengan ikon `bi bi-check-circle` untuk tampilan yang lebih profesional.

Kode Lengkap setelah Penyempurnaan

```
<!-- File: apps/templates/bots/add_command.html -->

{% extends 'dashboard/base.html' %}

{% block content %}
<div class="container mt-4">
    <!-- Header Section -->
```

Penyempurnaan Halaman Bots

```
<div class="bg-primary text-white text-center p-4 rounded mb-4">
    <h1 class="display-6"><i class="bi bi-plus-circle"></i> Tambah Command untuk Bot:
    {{ bot.name }}</h1>
</div>

<div class="card shadow-sm border-0">
    <div class="card-header bg-primary text-white">
        <h5 class="card-title"><i class="bi bi-terminal"></i> Form Tambah Command</h5>
    </div>
    <div class="card-body bg-light">
        <form method="post">
            {% csrf_token %}

            <!-- Form Group for Command
Field -->

            <div class="mb-3">
                <label
for="{{ form.command.id_for_label }}"
class="form-label"><i class="bi bi-code"></i>
Command</label>

                {{ form.command }} <!--
Bootstrap styling default untuk form input -->
            </div>

            <!-- Form Group for Response
Field -->

            <div class="mb-3">
                <label
for="{{ form.response.id_for_label }}"
class="form-label"><i class="bi bi-chat-dots"></i> Response</label>

                {{ form.response }} <!--
Bootstrap styling default untuk textarea -->
            </div>
```

Penyempurnaan Halaman Bots

```
        <!-- Submit Button -->
        <div class="d-flex justify-
content-end">
            <button type="submit"
class="btn btn-success">
                <i class="bi bi-check-
circle"></i> Tambah Command
            </button>
            <a href="{% url
'manage_bots' %}" class="btn btn-secondary ms-
2">
                <i class="bi bi-arrow-
left"></i> Kembali
            </a>
        </div>
    </form>
</div>
</div>
</div>
{% endblock %}
```

Penjelasan Perubahan

1. *Desain Card:*

- `card shadow-sm` digunakan untuk memberikan bayangan halus pada card, menciptakan kedalaman visual yang membuat card terlihat lebih terpisah dari latar belakang.
- `bg-primary` dan `text-white` digunakan untuk header card agar lebih menarik dan menonjol, dengan kontras yang jelas.

2. *Formulir:*

- Ikon ditambahkan pada label formulir (`bi bi-terminal` untuk Command dan `bi bi-`

Penyempurnaan Halaman Bots

chat-dots untuk Response) untuk memberikan konteks visual yang jelas.

- `form-control border border-primary` digunakan untuk input dan textarea agar terlihat lebih menonjol dan jelas dengan border berwarna biru.
- Placeholder ditambahkan pada input dan textarea untuk memberikan petunjuk visual tentang apa yang harus diisi pengguna.
- Tombol submit diubah menjadi `btn btn-success` dengan ikon `bi bi-check-circle`, menambahkan warna hijau dan ikon cek yang menunjukkan tindakan yang berhasil.

Kekurangan Kode Sebelumnya

Kode sebelumnya menggunakan card dengan desain dan elemen formulir yang cukup dasar. Tanpa ikon, border, dan placeholder, tampilan formulir tidak memberikan petunjuk yang cukup bagi pengguna. Dengan penambahan elemen desain seperti ikon, border, placeholder, dan styling tombol, halaman ini kini lebih menarik dan informatif, mempermudah pengguna dalam menambahkan perintah baru untuk bot.

15.5 Memperbaiki Halaman Edit Command

Pada bagian ini, kita akan membahas perbaikan desain halaman *Edit Command* untuk memastikan halaman ini tidak hanya fung-

Penyempurnaan Halaman Bots

sional tetapi juga mudah digunakan dan estetik. Halaman ini memungkinkan pengguna untuk mengedit perintah bot yang sudah ada, dan desain yang baik akan meningkatkan pengalaman pengguna secara keseluruhan.

Perbaikan dan Penjelasan

Kita akan memperbaiki halaman ini dengan menerapkan beberapa perubahan desain untuk meningkatkan estetika dan fungsionalitas:

1. **Penggunaan Kartu dan Header:** Menggunakan kartu dengan header berwarna untuk menyoroti judul halaman dan meningkatkan visualisasi.
2. **Menambahkan Ikon dan Placeholder:** Menambahkan ikon dan placeholder pada input dan textarea untuk memberikan petunjuk yang lebih jelas kepada pengguna.
3. **Styling Form:** Menggunakan kelas-kelas tambahan dari Bootstrap untuk meningkatkan styling elemen form.

Berikut adalah kode yang telah diperbaiki:

```
<!-- File: apps/templates/bots/edit_command.html -->

{% extends 'dashboard/base.html' %}

{% block content %}
<div class="container mt-4">
  <!-- Header Section -->
  <div class="bg-primary text-white text-center p-4 rounded mb-4">
```

Penyempurnaan Halaman Bots

```
<h1 class="display-6"><i class="bi bi-pencil-square"></i> Edit Command untuk Bot:
{{ bot.name }}</h1>
</div>

<div class="card shadow-sm border-0">
  <div class="card-header bg-primary text-white">
    <h5 class="card-title"><i class="bi bi-pencil"></i> Form Edit Command</h5>
  </div>
  <div class="card-body bg-light">
    <form method="post">
      {% csrf_token %}

      <!-- Form Group for Command
Field -->
      <div class="mb-3">
        <label
for="{{ form.command.id_for_label }}"
class="form-label"><i class="bi bi-code-slash"></i> Command</label>
        {{ form.command }} <!--
Menampilkan input field tanpa tambahan class -->
      </div>

      <!-- Form Group for Response
Field -->
      <div class="mb-3">
        <label
for="{{ form.response.id_for_label }}"
class="form-label"><i class="bi bi-chat-dots-fill"></i> Response</label>
        {{ form.response }} <!--
Menampilkan textarea tanpa tambahan class -->
      </div>

      <!-- Submit Button -->
```

Penyempurnaan Halaman Bots

```
        <div class="d-flex justify-  
content-end">  
            <button type="submit"  
class="btn btn-primary">  
                <i class="bi bi-check-  
circle-fill"></i> Update Command  
            </button>  
            <a href="{% url  
'manage_bots' %}" class="btn btn-secondary ms-  
2">  
                <i class="bi bi-arrow-  
left-circle"></i> Kembali  
            </a>  
        </div>  
    </form>  
</div>  
</div>  
</div>  
{% endblock %}
```

Penjelasan Kode:

1. Header dan Kartu:

- `card shadow-sm`: Menggunakan kartu dengan bayangan untuk menambah kedalaman visual.
- `card-header bg-primary text-white`: Menyediakan header yang mencolok dengan latar belakang biru dan teks putih, disertai ikon untuk kontekstualisasi.

2. Formulir Input:

- `form-control border border-primary`: Menambahkan border berwarna pada input dan textarea untuk visibilitas yang lebih baik.

Penyempurnaan Halaman Bots

- `placeholder`: Menambahkan placeholder untuk memberikan instruksi atau contoh tentang data yang diharapkan.

3. Tombol Submit:

- `btn btn-success`: Menggunakan kelas Bootstrap untuk tombol dengan warna hijau yang menonjol, serta menambahkan ikon untuk memberi makna visual pada tombol.

Kekurangan dari kode sebelumnya:

1. **Desain Visual**: Tampilan halaman menggunakan elemen dasar dari Bootstrap, tetapi tidak memanfaatkan sepenuhnya kemampuan styling Bootstrap untuk meningkatkan keterbacaan dan estetika.
2. **Konsistensi Ikon**: Kode ini tidak menggunakan ikon untuk meningkatkan keterbacaan dan memberikan konteks visual yang lebih baik.
3. **Placeholder**: Tidak ada placeholder di dalam input dan textarea, yang bisa membuat pengguna bingung tentang format data yang diharapkan.
4. **Penempatan Elemen**: Elemen form dan tombol dapat disempurnakan untuk meningkatkan penampilan dan keteraturan halaman.

Perubahan ini meningkatkan pengalaman pengguna dengan memberikan desain yang lebih menarik dan fungsionalitas yang lebih jelas, serta memastikan halaman **Edit Command** lebih mudah digunakan dan konsisten dengan halaman lainnya di aplikasi.

Kesimpulan Bab

Bab ini berfokus pada penyempurnaan beberapa halaman kunci dalam aplikasi *platform_bot*, khususnya halaman-halaman yang berkaitan dengan pengelolaan bot dan perintahnya. Dengan melakukan perbaikan pada desain dan fungsionalitas halaman-halaman tersebut, kita telah meningkatkan pengalaman pengguna dan memastikan bahwa aplikasi tidak hanya fungsional tetapi juga menyenangkan untuk digunakan.

Poin-Poin Utama:

1. *Halaman Navbar:*

- **Perubahan:** Mengubah desain navbar untuk meningkatkan keterbacaan dan aksesibilitas, serta memastikan responsivitas yang lebih baik di berbagai perangkat.
- **Tujuan:** Menyediakan navigasi yang jelas dan konsisten, yang penting untuk navigasi yang efektif di dalam dashboard.

2. *Halaman Edit Bot:*

- **Perubahan:** Menerapkan desain kartu dengan styling yang lebih baik, menambahkan ikon untuk memperjelas fungsi, dan menyempurnakan elemen form dengan border dan placeholder.
- **Tujuan:** Menyediakan antarmuka yang lebih profesional dan mudah digunakan untuk mengedit detail bot, serta memudahkan pengguna dalam menginput data yang tepat.

3. *Halaman Add Command:*

Penyempurnaan Halaman Bots

- **Perubahan:** Menggunakan styling kartu dan header yang lebih mencolok, menambahkan ikon dan placeholder untuk elemen form, serta menyempurnakan tombol dengan ikon dan warna yang konsisten.
- **Tujuan:** Mempermudah pengguna dalam menambahkan perintah baru dengan memberikan antarmuka yang jelas dan intuitif, serta meningkatkan pengalaman pengguna dengan desain yang lebih menarik.

4. Halaman Edit Command:

- **Perubahan:** Menambahkan desain kartu dengan header yang lebih menonjol, menggunakan ikon dan placeholder dalam input form, serta memperbaiki tombol dengan styling yang lebih baik.
- **Tujuan:** Meningkatkan keterbacaan dan kemudahan penggunaan halaman edit perintah bot, memberikan pengguna kontrol yang lebih baik dalam mengelola perintah yang ada.

Peningkatan desain pada halaman-halaman ini tidak hanya berfokus pada estetika, tetapi juga pada fungsionalitas dan pengalaman pengguna secara keseluruhan. Dengan menerapkan prinsip-prinsip desain yang konsisten, menggunakan elemen-elemen Bootstrap secara efektif, dan memastikan bahwa antarmuka pengguna mudah digunakan, aplikasi **platform_bot** sekarang menawarkan pengalaman yang lebih baik dan lebih profesional. Desain yang lebih baik dan fungsionalitas yang lebih jelas membantu pengguna dalam mengelola bot dan perintah dengan lebih efisi-

Penyempurnaan Halaman Bots

en, yang pada akhirnya meningkatkan produktivitas dan kepuasan pengguna.

BAB 16 -Menginstall Project ke GitHub dan Deploy Django

Proses pengembangan aplikasi web modern tidak hanya berhenti pada tahap penulisan kode. Untuk memastikan bahwa aplikasi dapat diakses secara online dan dikelola dengan baik, kita memerlukan sistem kontrol versi dan platform hosting yang handal. Salah satu solusi paling populer adalah menggunakan GitHub sebagai platform kontrol versi dan Django sebagai kerangka kerja aplikasi web. Setelah pengembangan selesai, langkah berikutnya adalah meng-online-kan aplikasi agar dapat diakses oleh pengguna melalui internet. Pada bab ini, kita akan membahas secara detail bagaimana menginstal project Django ke GitHub dan kemudian melakukan deployment agar aplikasi kita dapat diakses secara publik.

16.1 Pendahuluan

Mengelola kode tanpa kontrol versi dapat menjadi hal yang rumit, terutama ketika bekerja secara kolaboratif atau jika aplikasi terus berkembang dengan fitur-fitur baru. Inilah alasan mengapa kita memerlukan sistem kontrol versi seperti Git dan GitHub. Git adalah alat untuk melacak perubahan pada kode, sementara GitHub menyediakan platform berbasis cloud untuk menyimpan dan berbagi kode secara online. Dengan GitHub, kita bisa bekerja dalam tim, mengelola versi kode dengan mudah, serta memudahkan rollback jika terjadi kesalahan.

Menginstall Project ke GitHub dan Deploy Django

Menggunakan Git untuk menyimpan dan mengelola project Django kita memberikan banyak keuntungan. Tidak hanya kita bisa melacak perubahan yang terjadi pada project, tetapi juga bisa memanfaatkan GitHub sebagai tempat untuk berbagi kode dengan tim, melakukan kolaborasi, serta menampilkan project kepada publik. Dengan GitHub, kita juga bisa memanfaatkan fitur-fitur seperti *Issues*, *Pull Requests*, dan *GitHub Actions* untuk otomatisasi dalam pipeline pengembangan.

Di sisi lain, deployment merupakan tahapan penting yang memungkinkan aplikasi kita bisa diakses oleh pengguna melalui internet. Tanpa proses ini, aplikasi yang sudah kita kembangkan tidak dapat digunakan secara langsung oleh pengguna. Dengan melakukan deployment, kita mentransformasikan aplikasi Django dari lingkungan lokal ke server yang dapat diakses melalui domain publik. Salah satu cara populer untuk deployment adalah menggunakan platform seperti Heroku, PythonAnywhere, atau server pribadi yang bisa dikonfigurasi.

Dalam bab ini, kita akan melangkah lebih jauh dengan membahas bagaimana kita bisa memanfaatkan GitHub untuk mengelola kode kita dan bagaimana melakukan deployment agar aplikasi Django kita dapat digunakan oleh pengguna secara online.

16.2 Mengupload Project Ke GitHub

Setelah memahami pentingnya kontrol versi dan deployment, langkah pertama yang harus kita lakukan adalah memastikan bahwa kode project Django kita dapat diakses dan dikelola de-

Menginstall Project ke GitHub dan Deploy Django

ngan baik melalui GitHub. Untuk itu, kita perlu mengunggah project tersebut ke repository GitHub. Repository ini akan berfungsi sebagai tempat penyimpanan kode yang terorganisir, dan memungkinkan kita melacak perubahan, serta berkolaborasi dengan pengembang lain jika diperlukan.

16.2.1 Membuat Repository di GitHub

Langkah pertama dalam proses ini adalah membuat repository baru di GitHub. Repository ini akan menjadi wadah untuk semua file dan folder project kita. Sebelum mengunggah project Django ke GitHub, pastikan bahwa kita sudah memiliki akun GitHub dan telah mengatur Git di mesin lokal kita.

Untuk memulai, buka halaman GitHub dan masuk ke akun GitHub kita. Setelah masuk, arahkan ke ikon *plus* di sudut kanan atas halaman, lalu pilih *New Repository*. Di sini, kita akan diminta untuk memberikan nama repository. Nama ini bisa sama dengan nama project Django kita agar lebih mudah dikenali. Misalnya, jika project kita bernama `platform_bot`, maka kita bisa memberi nama repository tersebut `platform_bot` juga.

Setelah menentukan nama, ada beberapa pengaturan penting yang perlu diperhatikan:

1. **Deskripsi (Description)** – Menambahkan deskripsi singkat tentang project. Deskripsi ini opsional, tetapi sangat membantu, terutama jika repository bersifat publik.
2. **Visibility (Public atau Private)** – Pilih antara membuat repository *public* atau *private*. Jika *public*, semua orang

Menginstall Project ke GitHub dan Deploy Django

bisa melihat repository kita, yang bisa berguna untuk project open source. Jika *private*, hanya kita dan kolaborator yang diizinkan dapat mengakses repository tersebut. Misalnya, jika ini adalah project pribadi atau dalam tahap pengembangan, kita bisa memilih opsi *private*.

3. ***Initialize repository with a README*** – Opsi ini akan menambahkan file README secara otomatis ke repository. README adalah file teks yang biasanya berisi informasi penting tentang project, seperti cara instalasi atau penjelasan singkat tentang fungsionalitasnya.

Setelah mengisi semua informasi dan menentukan pengaturan, klik tombol *Create Repository*. Sekarang, kita telah berhasil membuat repository baru di GitHub. Selanjutnya, kita akan menghubungkan repository ini dengan project Django lokal kita dan mengunggah kode ke repository tersebut.

16.2.2 Mengatur Personal Access Token (PAT) di GitHub

Dalam bagian ini, kita akan membahas cara membuat dan mengatur ***Personal Access Token (PAT)*** di GitHub untuk otentikasi ketika berinteraksi dengan repositori Git. Setelah GitHub menghentikan dukungan untuk otentikasi menggunakan kata sandi, kita perlu menggunakan PAT sebagai pengganti kata sandi dalam operasi Git seperti `push`, `pull`, dan `clone` melalui HTTP.

Menginstall Project ke GitHub dan Deploy Django

Personal Access Token adalah semacam kunci digital yang mewakili izin akun kita di GitHub. Dengan PAT, kita dapat mengakses fitur GitHub seperti repositori, tanpa perlu memasukkan kata sandi akun GitHub setiap kali melakukan operasi Git.

Membuat Personal Access Token di GitHub

Langkah pertama adalah membuat token di akun GitHub kita. Berikut cara melakukannya:

Masuk ke Akun GitHub

Pertama, kita perlu masuk ke akun GitHub menggunakan peramban. Setelah masuk, kita dapat mengakses pengaturan akun dengan mengklik foto profil kita di sudut kanan atas, lalu memilih *Settings*.

Membuka Pengaturan Developer

Pada halaman pengaturan, gulir ke bawah di panel sebelah kiri dan temukan bagian *Developer settings*. Klik pada bagian ini untuk melanjutkan ke pengaturan akses token.

Membuat Personal Access Token

- Pada bagian *Developer settings*, klik *Personal access tokens*.
- Setelah itu, klik tombol *Generate new token*.
- Kita akan diminta untuk memberikan nama atau deskripsi untuk token ini. Pilih nama yang jelas seperti "Token untuk Repositori Platform Bot."

Menginstall Project ke GitHub dan Deploy Django

- Tentukan berapa lama token ini akan berlaku. GitHub menawarkan opsi untuk token yang dapat berlaku dalam 30 hari, 60 hari, atau lebih lama.

Jika kita ingin token berlaku tanpa batas waktu, kita bisa memilih opsi tanpa tanggal kedaluwarsa, namun perlu hati-hati dalam menjaga keamanan token ini.

Mengatur Izin Token

Saat membuat token, kita akan diminta untuk memilih hak akses atau **scope** yang diperlukan. Untuk bekerja dengan repositori Git, pastikan opsi **repo** dipilih, karena ini memberi kita izin untuk membaca dan menulis ke repositori kita. Jika kita juga bekerja dengan fitur lain seperti paket, tindakan (actions), atau alat pengelolaan proyek, kita dapat memilih izin tambahan sesuai kebutuhan.

Menghasilkan Token

Setelah semua pengaturan diatur, klik **Generate token**. GitHub kemudian akan menampilkan token yang telah dibuat. Penting untuk **menyalin token ini** segera karena setelah halaman ditutup, token tidak dapat dilihat lagi. Simpan token ini di tempat yang aman, misalnya di pengelola kata sandi.

Menggunakan Personal Access Token

Setelah kita memiliki PAT, langkah berikutnya adalah menggunakannya dalam operasi Git kita. Setiap kali Git meminta otentikasi

Menginstall Project ke GitHub dan Deploy Django

(misalnya saat menjalankan perintah `git push`), masukkan token ini sebagai pengganti kata sandi GitHub kita.

Contoh Penggunaan Personal Access Token di Git

Misalnya, ketika kita mencoba untuk `push` ke repositori, Git akan meminta kita memasukkan username dan kata sandi:

```
$ git push origin main
Username for 'https://github.com': username
Password for 'https://username@github.com':
[Paste your PAT here]
```

Di bagian **Password**, kita *tidak memasukkan kata sandi akun GitHub*, melainkan kita memasukkan token yang telah kita buat tadi. Ini akan berhasil menggantikan otentikasi menggunakan kata sandi.

Menyimpan Personal Access Token Secara Permanen

Agar tidak perlu memasukkan token setiap kali menjalankan operasi Git, kita bisa menyimpan token secara lokal di komputer kita menggunakan fitur **Git Credential Manager**. Dengan demikian, Git akan otomatis menggunakan token ini di masa mendatang.

Mengonfigurasi Git untuk Menyimpan Token

Jalankan perintah berikut di terminal untuk mengonfigurasi Git agar menyimpan informasi otentikasi:

```
$ git config --global credential.helper store
```

Menginstall Project ke GitHub dan Deploy Django

Dengan konfigurasi ini, Git akan menyimpan token kita setelah pertama kali dimasukkan, sehingga kita tidak perlu memasukkannya kembali setiap kali melakukan push, pull, atau clone.

Menggunakan Git Credential Manager (Opsional)

Jika belum diinstal, kita bisa mengunduh dan memasang *Git Credential Manager* dari tautan berikut:

<https://github.com/GitCredentialManager/git-credential-manager>

Setelah Credential Manager terpasang, Git akan otomatis menggunakan token yang tersimpan.

Dengan demikian, kita telah mempelajari cara membuat *Personal Access Token (PAT)* di GitHub dan menggunakannya sebagai pengganti kata sandi dalam otentikasi Git. PAT ini memberikan keamanan yang lebih baik dan menjadi standar baru dalam mengakses repositori GitHub melalui HTTPS. Jangan lupa untuk menyimpan token di tempat yang aman dan selalu menghapusnya jika tidak lagi diperlukan.

16.2.3 Menghubungkan Local Repository dengan GitHub

Setelah membahas pentingnya kontrol versi dan deployment pada proyek Django, langkah selanjutnya adalah menghubungkan proyek lokal kita dengan GitHub, sehingga kita bisa mulai menyimpan dan mengelola perubahan proyek secara online. Untuk melakukan ini, kita perlu menginisialisasi Git di dalam direktori proyek kita. Proses ini bertujuan untuk mengubah proyek yang

Menginstall Project ke GitHub dan Deploy Django

sedang kita kerjakan menjadi repository Git lokal yang nantinya bisa disinkronisasi dengan repository di GitHub.

Pertama, pastikan bahwa Git sudah terinstall di komputer kita. Jika belum, kita bisa mendownload dan menginstall Git dari situs resminya. Setelah Git terinstall, buka terminal atau command prompt, lalu navigasi ke direktori proyek Django kita. Misalnya, jika proyek kita berada di folder `platform_bot`, kita bisa menjalankan perintah berikut untuk masuk ke dalam folder tersebut.

```
$ cd platform_bot
```

Setelah berada di direktori proyek, kita perlu menginisialisasi Git pada folder ini. Perintah yang digunakan adalah sebagai berikut:

```
$ git init
```

Perintah ini akan membuat folder `.git` yang berfungsi sebagai penyimpanan metadata Git di dalam proyek kita. Folder `.git` ini akan menyimpan semua informasi yang dibutuhkan untuk melacak perubahan pada kode, seperti commit, branch, dan history perubahan. Setelah Git terinisialisasi, kita bisa mulai menambahkan file proyek kita ke dalam staging area dengan perintah:

```
$ git add .
```

Perintah ini akan menambahkan semua file di dalam direktori proyek ke staging area, artinya file-file tersebut siap untuk di-

Menginstall Project ke GitHub dan Deploy Django

commit. Pada tahap ini, Git belum melakukan commit, tetapi hanya menandai file-file yang siap dicatat. Jika kita ingin memastikan file mana saja yang telah ditambahkan, kita bisa menggunakan perintah

```
$ git status.
```

Setelah menambahkan file, langkah selanjutnya adalah membuat commit pertama. Commit adalah titik penyimpanan dalam repository yang memungkinkan kita untuk kembali ke versi tertentu dari kode jika diperlukan. Untuk membuat commit, kita bisa menjalankan perintah berikut:

```
$ git commit -m "Initial commit"
```

Commit ini akan menyimpan snapshot dari kode kita pada saat itu, dengan pesan "Initial commit" sebagai deskripsi perubahan. Pesan commit ini sangat penting, karena akan membantu kita dan tim untuk memahami apa yang berubah dalam setiap commit.

Setelah commit dibuat, kita perlu menghubungkan repository lokal ini dengan repository di GitHub. Untuk itu, kita harus membuat repository baru di GitHub terlebih dahulu. Setelah repository dibuat, GitHub akan memberikan URL repository tersebut. Kita perlu menambahkan URL ini ke repository lokal dengan menggunakan perintah:

```
$ git remote add origin  
https://github.com/username/platform_bot.git
```

Menginstall Project ke GitHub dan Deploy Django

Perintah ini akan menghubungkan repository lokal kita dengan repository GitHub yang baru kita buat. `origin` adalah nama default untuk repository jarak jauh (remote repository), dan kita bisa memberinya nama lain jika diperlukan.

Langkah terakhir adalah mengunggah (push) semua commit kita ke repository di GitHub. Untuk itu, kita bisa menggunakan perintah:

```
$ git push -u origin main
```

Ketika menjalankan perintah `git push`, Git akan meminta kita memasukkan username dan password.

Contoh:

```
$ git push origin main
Username for 'https://github.com': username
Password for 'https://username@github.com':
[Paste your PAT here]
```

Di bagian **Password**, kita *tidak memasukkan kata sandi akun GitHub*, melainkan kita memasukkan token yang telah kita buat tadi. Ini akan berhasil menggantikan otentikasi menggunakan kata sandi.

Perintah ini akan mengirim semua commit dari branch utama (`main`) ke repository `origin` yang terhubung dengan GitHub. Opsi `-u` memastikan bahwa branch lokal kita (`main`) akan selalu terhubung dengan branch `main` di GitHub, sehingga di kemu-

Menginstall Project ke GitHub dan Deploy Django

dian hari kita cukup menjalankan `$ git push` untuk mengunggah perubahan baru.

Setelah proses ini selesai, proyek Django kita kini telah berhasil diunggah ke GitHub, dan setiap perubahan yang kita lakukan dapat didokumentasikan dan diatur melalui sistem kontrol versi ini.

16.3 Menyiapkan Server Untuk Deployment

Setelah berhasil mengupload proyek Django kita ke GitHub, langkah selanjutnya adalah menyiapkan server untuk melakukan deployment. Deployment adalah proses yang memungkinkan aplikasi Django kita diakses oleh publik melalui internet. Memilih platform yang tepat untuk melakukan deployment menjadi hal yang penting, karena setiap platform memiliki kelebihan dan kekurangannya masing-masing, tergantung pada kebutuhan dan skala aplikasi yang kita bangun.

16.3.1 Memilih Platform untuk Deployment

Ada beberapa platform yang sering digunakan untuk melakukan deployment aplikasi Django. Platform-platform ini umumnya menyediakan layanan cloud yang memudahkan kita untuk meng-host aplikasi tanpa perlu memikirkan konfigurasi server yang kompleks. Berikut ini adalah beberapa platform yang umum digunakan untuk deployment Django:

Menginstall Project ke GitHub dan Deploy Django

Heroku

Heroku adalah salah satu platform paling populer untuk deployment aplikasi web, termasuk aplikasi Django. Kelebihan Heroku terletak pada kemudahan penggunaannya. Dengan beberapa perintah saja, aplikasi kita bisa langsung dideploy dan diakses secara online. Heroku juga menawarkan integrasi yang baik dengan GitHub, sehingga setiap kali kita melakukan push ke repository GitHub, aplikasi di Heroku dapat diperbarui secara otomatis.

Heroku mendukung add-ons untuk menambah fungsionalitas aplikasi, seperti database PostgreSQL dan layanan pengiriman email. Namun, versi gratis Heroku memiliki keterbatasan, seperti waktu respon yang lebih lambat dan batasan penggunaan sumber daya.

Virtual Private Server (VPS)

Jika kita membutuhkan kontrol penuh atas server tempat aplikasi Django di-host, menggunakan Virtual Private Server (VPS) adalah pilihan yang tepat. VPS memberikan kebebasan penuh untuk mengatur lingkungan server, mulai dari instalasi sistem operasi hingga konfigurasi database dan web server seperti Nginx atau Apache.

VPS menawarkan performa yang lebih stabil dibandingkan platform seperti Heroku karena kita mendapatkan alokasi sumber daya yang lebih dedicated. Namun, menggunakan VPS

Menginstall Project ke GitHub dan Deploy Django

memerlukan pemahaman tentang administrasi server, termasuk tanggung jawab penuh atas keamanan dan performa server.

Railway

Railway adalah platform cloud yang relatif baru namun cepat populer di kalangan pengembang web. Railway menawarkan proses deployment yang sangat mudah dan mirip dengan Heroku. Kita bisa menghubungkan repository GitHub kita ke Railway, dan platform ini akan otomatis melakukan deployment setiap kali ada perubahan di repository tersebut.

Railway menawarkan fleksibilitas dalam penggunaan sumber daya, serta menyediakan layanan database yang sudah terintegrasi. Versi gratisnya memiliki batasan dalam hal alokasi memori dan jumlah proyek yang bisa di-host.

PythonAnywhere

PythonAnywhere adalah platform cloud yang dirancang khusus untuk aplikasi Python, termasuk Django. Kelebihan PythonAnywhere adalah proses setup-nya yang sangat mudah, tanpa perlu melakukan banyak konfigurasi server secara manual.

Kita hanya perlu mengunggah kode proyek, melakukan pengaturan virtual environment, dan mengonfigurasi webhook serta database. Platform ini juga menyediakan layanan HTTPS, yang membuatnya cocok untuk aplikasi Django yang

Menginstall Project ke GitHub dan Deploy Django

membutuhkan webhook, seperti integrasi bot Telegram dan WhatsApp.

PythonAnywhere juga menyediakan opsi deployment yang cepat dan stabil, dengan paket gratis yang sudah cukup untuk proyek kecil. Namun, untuk skala yang lebih besar atau membutuhkan lebih banyak kontrol, paket berbayar mungkin lebih sesuai.

16.3.2 Memilih Platform Berdasarkan Kebutuhan

Ketika memilih platform untuk deployment, beberapa faktor yang perlu dipertimbangkan antara lain skala aplikasi, anggaran, dan tingkat kontrol yang kita butuhkan atas server. Untuk proyek Django yang melibatkan bot dengan webhook,

Berikut adalah beberapa rekomendasi platform yang bisa kita pertimbangkan:

- **Heroku:** Pilihan terbaik jika kita mencari kemudahan deployment dan tidak ingin repot dengan konfigurasi server. Heroku mendukung integrasi webhook melalui HTTPS, sehingga cocok untuk proyek bot seperti Telegram atau WhatsApp.
- **VPS (Virtual Private Server):** Jika kita membutuhkan kontrol penuh atas server, VPS seperti DigitalOcean atau AWS EC2 menawarkan fleksibilitas penuh. VPS juga memungkinkan pengaturan server Nginx atau Apache untuk menangani permintaan webhook dengan efisien.

Menginstall Project ke GitHub dan Deploy Django

- ***Railway***: Railway cocok untuk pengembang yang ingin proses deployment yang cepat dan sederhana, dengan dukungan webhook yang baik.
- ***PythonAnywhere***: PythonAnywhere menawarkan kemudahan untuk deployment aplikasi Django dengan sedikit konfigurasi. Platform ini cocok untuk aplikasi bot berbasis Python yang membutuhkan webhook dan layanan HTTPS dengan pengaturan yang sederhana. PythonAnywhere adalah pilihan yang tepat jika kita menginginkan platform yang ringan, mudah digunakan, dan khusus untuk proyek Python.

16.3.3 Rekomendasi Platform

Jika kita mencari kemudahan deployment tanpa perlu konfigurasi yang rumit, ***Heroku*** atau ***PythonAnywhere*** bisa menjadi pilihan terbaik, terutama untuk proyek skala kecil hingga menengah. Namun, jika kita membutuhkan kontrol lebih atau aplikasi bot kita memerlukan performa tinggi, ***VPS*** akan memberikan fleksibilitas dan skalabilitas lebih.

Menentukan platform deployment adalah langkah penting dalam memastikan aplikasi Django kita dapat diakses secara stabil dan optimal oleh pengguna.

16.4 Deploy Ke PythonAnywhere

Untuk memulai proses deployment di PythonAnywhere, langkah pertama adalah membuat akun di platform tersebut.

Menginstall Project ke GitHub dan Deploy Django

PythonAnywhere menyediakan opsi gratis dengan beberapa batasan yang cocok untuk keperluan pengembangan dan pengujian proyek kecil seperti proyek Django kita. Jika belum memiliki akun, kita dapat mengunjungi situs resmi PythonAnywhere dan mengikuti langkah-langkah pendaftaran. Setelah berhasil mendaftar, login ke dashboard PythonAnywhere untuk melanjutkan.

16.4.1 Meng-clone Repository dari GitHub

Setelah login, kita akan diarahkan ke dashboard utama. Untuk mulai menyiapkan environment deployment, buka terminal bash melalui **New Console** dan pilih opsi **Bash**. Dari terminal ini, kita dapat meng-clone proyek Django dari GitHub ke PythonAnywhere. Misalnya, gunakan perintah berikut untuk menyalin repository dari GitHub:

```
$ git clone  
https://github.com/username/platform_bot.git
```

Ganti `username` dengan nama pengguna GitHub kita, dan `platform_bot` dengan nama repository proyek. Ini akan menyalin semua file dan struktur proyek ke PythonAnywhere.

16.4.2 Mengatur Aplikasi Web di PythonAnywhere

Setelah repository berhasil di-clone, kita bisa melanjutkan pengaturan aplikasi web di PythonAnywhere. Masuk ke tab **Web** pada dashboard, lalu pilih untuk menambahkan aplikasi baru. Ketika diminta memilih framework, pilih **Django** agar

Menginstall Project ke GitHub dan Deploy Django

PythonAnywhere menyiapkan environment Django secara otomatis. Pastikan PythonAnywhere mendeteksi versi Python yang benar sesuai proyek kita.

16.4.3 Mengonfigurasi Static Files dan Templates

Pada bagian ini, penting untuk mengatur static files dan template directories agar Django dapat melayani file seperti CSS, JavaScript, gambar, dan template HTML dengan benar. Konfigurasi ini dilakukan di bagian **Static files** di tab **Web**. Tambahkan konfigurasi berikut untuk file statis:

- **URL:** `/static/`
- **Directory:**
`/home/username/platform_bot/apps/static/assets`

Selain static files, kita juga perlu memastikan direktori template sudah diatur dengan benar. PythonAnywhere harus mengetahui di mana file template aplikasi kita berada. Untuk template, atur directory seperti ini:

- **URL:** `/templates/`
- **Directory:**
`/home/username/platform_bot/apps/templates`

Ganti username dengan nama pengguna PythonAnywhere kita. URL `/static/` digunakan oleh aplikasi untuk melayani file statis, dan `/templates/` digunakan untuk mengakses file HTML template. Pengaturan ini memastikan bahwa baik file

Menginstall Project ke GitHub dan Deploy Django

statis maupun template dapat ditemukan dan dilayani dengan benar oleh server.

16.4.4 Mengatur File WSGI

Langkah berikutnya adalah memastikan bahwa PythonAnywhere mengetahui di mana file WSGI proyek kita berada. Pada bagian **Source code** di tab **Web**, isikan path ke file `wsgi.py` aplikasi kita. Biasanya, path-nya adalah:

```
/home/username/platform_bot/platform_bot/wsgi.py
```

File ini digunakan oleh server untuk menjalankan aplikasi Django.

16.4.5 Reload Aplikasi

Setelah semua konfigurasi selesai—baik static files, template directories, maupun path WSGI—langkah terakhir adalah me-reload aplikasi. Di tab **Web**, klik tombol **Reload** untuk menerapkan semua perubahan. Dengan melakukan reload, PythonAnywhere akan membaca konfigurasi yang baru dan menjalankan aplikasi dengan setup terbaru.

Setelah reload, kita bisa membuka domain aplikasi yang disediakan oleh PythonAnywhere untuk memverifikasi apakah aplikasi berjalan dengan baik dan apakah static files serta template sudah berfungsi sebagaimana mestinya. Jika ada masalah, cek log error yang bisa diakses dari tab **Log** di dashboard PythonAnywhere untuk mendapatkan petunjuk lebih lanjut.

Menginstall Project ke GitHub dan Deploy Django

Dengan mengikuti langkah-langkah ini, kita telah berhasil melakukan deployment aplikasi Django ke PythonAnywhere dan menyiapkan agar file statis serta template dapat diakses dengan benar.

16.5 Deploy Ke Platform Heroku

16.5.1 Instalasi dan Setup Heroku CLI

Setelah menyiapkan server untuk deployment, salah satu platform yang banyak digunakan untuk meng-host aplikasi Django adalah Heroku. Heroku menawarkan kemudahan dalam proses deployment dan manajemen aplikasi, menjadikannya pilihan yang tepat untuk pengembang yang ingin fokus pada pengembangan tanpa terbebani dengan pengaturan server yang kompleks. Untuk memulai, kita perlu menginstal Heroku CLI, yang merupakan alat baris perintah yang memungkinkan kita untuk berinteraksi dengan platform Heroku.

Langkah pertama adalah mengunjungi situs resmi Heroku dan membuat akun jika kita belum memilikinya. Setelah mendaftar, kita dapat melanjutkan untuk menginstal Heroku CLI. Jika kita menggunakan sistem operasi Windows, kita bisa mengunduh installer yang sesuai dari halaman download Heroku. Untuk pengguna macOS atau Linux, kita dapat menggunakan Homebrew atau apt-get. Misalnya, untuk pengguna macOS,

Menginstall Project ke GitHub dan Deploy Django

jalankan perintah berikut di terminal:

```
$ brew tap heroku/brew && brew install heroku # Menginstal Heroku CLI dengan Homebrew
```

Bagi pengguna Ubuntu, kita bisa menggunakan perintah berikut:

```
$ curl https://cli-assets.heroku.com/install.sh | sh # Menginstal Heroku CLI di Ubuntu
```

Setelah instalasi selesai, kita dapat memverifikasi apakah Heroku CLI terpasang dengan benar dengan menjalankan perintah berikut:

```
$ heroku --version # Memeriksa versi Heroku CLI
```

Dengan Heroku CLI yang terinstal, langkah berikutnya adalah melakukan login ke akun Heroku kita. Kita cukup menjalankan perintah berikut:

```
$ heroku login # Masuk ke akun Heroku
```

Setelah menjalankan perintah ini, jendela browser akan terbuka, meminta kita untuk memasukkan kredensial akun Heroku. Setelah berhasil masuk, terminal akan memberikan konfirmasi bahwa kita telah berhasil login.

Selanjutnya, kita perlu menyiapkan aplikasi kita untuk deployment. Pastikan kita berada di direktori project Django kita. Sebelum melakukan deploy, kita perlu membuat file `Procfile`, yang memberi tahu Heroku bagaimana menjalankan aplikasi kita.

Di dalam direktori project, kita dapat membuat file ini dengan perintah:

Menginstall Project ke GitHub dan Deploy Django

```
$ echo "web: gunicorn project_name.wsgi --log-file -" > Procfile
# Membuat Procfile untuk Heroku
```

Gantilah `project_name` dengan nama project kita. Dengan file ini, Heroku akan tahu untuk menjalankan server Gunicorn ketika aplikasi kita dijalankan.

Selanjutnya, kita perlu mengonfigurasi beberapa pengaturan di file `settings.py` untuk memastikan aplikasi dapat berjalan di Heroku. Misalnya, kita perlu menambahkan Heroku ke `ALLOWED_HOSTS`:

```
ALLOWED_HOSTS = ['.herokuapp.com'] #
Menambahkan domain Heroku
```

Dengan semua persiapan ini, kita siap untuk melakukan deploy aplikasi kita ke Heroku. Proses ini akan membimbing kita dalam mengunggah aplikasi dan menjalankannya secara online, memungkinkan bot kita untuk berfungsi dengan baik.

16.5.2 Mengatur Procfile dan requirements.txt

Setelah berhasil login ke Heroku dan membuat aplikasi baru, langkah selanjutnya adalah menyiapkan dua file penting: Procfile dan `requirements.txt`. Kedua file ini berperan krusial dalam menentukan bagaimana aplikasi kita akan berjalan di Heroku.

File `Procfile` adalah file teks yang memberitahukan Heroku bagaimana cara menjalankan aplikasi kita. Untuk aplikasi Django

Menginstall Project ke GitHub dan Deploy Django

yang menggunakan Gunicorn sebagai server, kita perlu memastikan bahwa `Procfile` berisi perintah yang tepat. Untuk membuat file ini, kita bisa menggunakan perintah berikut di terminal, pastikan kita berada di dalam direktori project:

```
$ echo "web: gunicorn project_name.wsgi --log-file -" > Procfile
# Membuat Procfile untuk Gunicorn
```

Gantilah `project_name` dengan nama project Django kita. Dengan isi ini, Heroku akan menjalankan Gunicorn dan memuat aplikasi kita dengan file `wsgi.py`.

Selanjutnya, kita perlu memastikan bahwa semua dependensi yang diperlukan untuk menjalankan aplikasi kita terdaftar di file `requirements.txt`. File ini berisi daftar paket Python yang diperlukan agar aplikasi kita dapat berjalan dengan baik. Untuk menghasilkan file `requirements.txt`, kita dapat menggunakan perintah pip berikut:

```
$ pip freeze > requirements.txt # Menghasilkan requirements.txt
```

Perintah ini akan mencatat semua paket yang terinstal di lingkungan virtual kita ke dalam file `requirements.txt`. Setelah itu, kita perlu memastikan bahwa file ini mencakup Gunicorn dan Django, serta pustaka lain yang kita gunakan dalam aplikasi. Jika kita menambahkan dependensi baru, kita bisa memperbarui file ini dengan cara yang sama.

Sekarang, dengan `Procfile` dan `requirements.txt` yang sudah diatur, kita siap untuk melanjutkan proses deployment. Heroku akan menggunakan informasi dari kedua file ini untuk men-

Menginstall Project ke GitHub dan Deploy Django

jalankan aplikasi kita dengan benar di server mereka. Hal ini sangat penting agar bot kita dapat berfungsi dengan baik di lingkungan produksi.

16.5.3 Deploy ke Heroku

Setelah semua file penting, seperti `Procfile` dan `requirements.txt`, siap, kita dapat melanjutkan ke langkah terakhir, yaitu menghubungkan project Django dengan Heroku dan melakukan deploy.

Pertama, kita harus membuat aplikasi baru di Heroku. Untuk melakukannya, pastikan kita sudah login ke akun Heroku melalui terminal. Kemudian, kita dapat membuat aplikasi baru menggunakan perintah berikut:

```
$ heroku create # Membuat aplikasi baru di Heroku
```

Setelah perintah ini dijalankan, Heroku akan secara otomatis membuat aplikasi baru dengan URL yang unik. URL tersebut akan digunakan untuk mengakses aplikasi kita secara online setelah deployment berhasil.

Langkah selanjutnya adalah menghubungkan aplikasi yang ada di GitHub repository dengan aplikasi yang baru saja kita buat di Heroku. Untuk melakukannya, kita harus menambahkan Heroku sebagai remote pada Git repository lokal kita. Gunakan perintah berikut di dalam direktori project kita:

Menginstall Project ke GitHub dan Deploy Django

```
$ git remote add heroku https://git.heroku.com/nama-app-kita.git  
# Menghubungkan Heroku dengan repository lokal
```

Gantilah nama-app-kita dengan nama aplikasi yang dihasilkan oleh Heroku. Sekarang, aplikasi kita sudah terhubung dengan Heroku, dan kita dapat melakukan deploy dengan mudah.

Untuk mengirim kode yang ada di branch `main` ke Heroku dan memulai proses deployment, kita dapat menjalankan perintah berikut:

```
$ git push heroku main # Melakukan deploy aplikasi ke Heroku
```

Proses ini akan mengunggah semua file project kita ke Heroku dan memulai build aplikasi. Heroku akan membaca file `Procfile` untuk mengetahui cara menjalankan aplikasi, dan `requirements.txt` untuk menginstal semua dependensi yang dibutuhkan. Jika tidak ada kesalahan, aplikasi akan segera berjalan di server Heroku, dan kita dapat mengaksesnya melalui URL yang diberikan sebelumnya.

Dengan perintah ini, aplikasi Django kita telah berhasil di-deploy ke Heroku dan siap diakses secara online. Bot yang kita bangun pun dapat mulai beroperasi, menerima dan merespons permintaan melalui webhook yang telah diatur sebelumnya. Heroku menyediakan solusi sederhana untuk deployment dengan langkah-langkah yang efisien, menjadikan platform ini pilihan yang ideal untuk menjalankan bot berbasis Django.

Menginstall Project ke GitHub dan Deploy Django

16.5.4 Menangani Environment Variables

Pada tahap ini, setelah aplikasi Django kita berhasil di-deploy ke Heroku, ada satu langkah penting yang harus kita lakukan agar aplikasi bisa berjalan dengan aman dan sesuai konfigurasi. Kita perlu mengatur environment variables yang digunakan oleh Django, seperti `SECRET_KEY`, `DEBUG`, dan `DATABASE_URL`. Mengelola variabel-variabel ini secara langsung di kode tidak disarankan karena bisa menimbulkan masalah keamanan, terutama pada `SECRET_KEY`.

Heroku menyediakan cara mudah untuk menangani environment variables melalui fitur yang disebut "Config Vars". Dengan Config Vars, kita dapat menyimpan nilai-nilai sensitif seperti `SECRET_KEY` tanpa perlu menyimpannya dalam kode kita. Untuk mengatur environment variables di Heroku, kita bisa melakukannya dengan beberapa langkah.

Pertama, pastikan kita sudah login ke Heroku melalui terminal, lalu tambahkan environment variable seperti `SECRET_KEY` menggunakan perintah berikut:

```
$ heroku config  
SECRET_KEY=super-secret-key # Mengatur SECRET_KEY
```

Perintah ini akan menambahkan variabel `SECRET_KEY` ke aplikasi di Heroku. Gantilah `super-secret-key` dengan kunci yang benar-benar rahasia dan aman. Selain `SECRET_KEY`, kita juga perlu mengatur beberapa variabel penting lainnya seperti `DEBUG` dan `ALLOWED_HOSTS`. Contoh perintah untuk mengatur

Menginstall Project ke GitHub dan Deploy Django

variabel `DEBUG` agar aplikasi kita berjalan dalam mode produksi adalah:

```
$ heroku config  
DEBUG=False # Mengatur mode produksi
```

Untuk variabel `ALLOWED_HOSTS`, kita bisa menambahkan domain Heroku yang telah dibuat saat kita menjalankan perintah `heroku create`.

Sebagai contoh, jika URL aplikasi kita adalah `my-app.herokuapp.com`, kita bisa mengaturnya seperti ini:

```
$ heroku config  
ALLOWED_HOSTS=my-app.herokuapp.com # Mengatur  
ALLOWED_HOSTS
```

Selain itu, jika aplikasi Django kita menggunakan webhook untuk mengoperasikan bot, kita perlu mengatur variabel yang terkait dengan URL webhook tersebut. Misalnya, jika kita sudah mendefinisikan `WEBHOOK_URL` di `settings.py`, kita bisa menambahkannya ke Heroku dengan perintah:

```
$ heroku config  
WEBHOOK_URL=https://my-app.herokuapp.com/webhook/ #  
Mengatur URL webhook
```

Setelah environment variables ini diset, Heroku akan secara otomatis menggunakannya saat menjalankan aplikasi. Kita tidak perlu lagi khawatir tentang masalah keamanan karena variabel-variabel sensitif ini tidak akan terlihat dalam kode yang kita commit ke GitHub.

Menginstall Project ke GitHub dan Deploy Django

Yang perlu diperhatikan adalah, setiap kali ada perubahan pada environment variables, aplikasi di Heroku akan menggunakan nilai yang terbaru tanpa perlu restart manual. Hal ini membuat proses deployment lebih efisien dan aman, menjaga agar kunci-kunci penting dan konfigurasi lainnya tidak bocor di repositori publik.

Dengan pengaturan ini, aplikasi Django kita sudah siap untuk beroperasi dengan konfigurasi yang tepat, baik dari segi keamanan maupun performa.

16.6 Mendapatkan Webhook URL

Webhook, sebagaimana sudah dibahas di bab sebelumnya, adalah mekanisme yang memungkinkan aplikasi untuk menerima notifikasi secara real-time dari server eksternal ketika ada kejadian tertentu. Dalam konteks bot, webhook berguna untuk menerima update atau perintah langsung dari platform yang mengirimkannya, seperti Telegram atau layanan lainnya. Hal ini memungkinkan bot kita merespons perintah atau notifikasi dengan cepat tanpa harus melakukan polling secara terus-menerus.

Pada intinya, webhook akan "menghubungkan" aplikasi bot kita dengan server eksternal, dan setiap kali ada aktivitas yang relevan, server akan mengirimkan data ke URL webhook yang kita tentukan. Oleh karena itu, URL webhook ini sangat penting karena menjadi pintu masuk bagi komunikasi antara bot kita dengan server.

Menginstall Project ke GitHub dan Deploy Django

16.6.1 Cara Mendapatkan Webhook di Heroku

Setelah aplikasi bot kita di-deploy ke Heroku, platform ini secara otomatis menyediakan URL yang dapat digunakan sebagai webhook. URL tersebut memiliki format seperti `https://nama-aplikasi.herokuapp.com/`. Inilah yang akan menjadi alamat webhook yang kita daftarkan pada platform bot seperti Telegram.

Untuk mendapatkan webhook di Heroku, langkah pertama yang perlu dilakukan adalah memastikan aplikasi sudah di-deploy dan bisa diakses secara online. Setelah aplikasi berjalan, kita dapat menggunakan URL yang diberikan oleh Heroku sebagai webhook.

Sebagai contoh, jika aplikasi kita di Heroku diberi nama `my-bot-app`, maka URL webhook kita akan seperti ini:

```
https://my-bot-app.herokuapp.com/webhook/
```

URL ini yang kemudian akan didaftarkan pada platform bot, misalnya Telegram. Untuk Telegram, kita perlu mengirimkan perintah API ke endpoint Telegram dengan format sebagai berikut:

```
$ curl -F "url=https://my-bot-app.herokuapp.com/webhook/"  
https://api.telegram.org/bot<TOKEN>/setWebhook
```

Pastikan `<TOKEN>` diganti dengan token API bot yang sudah didapatkan saat membuat bot di Telegram. Perintah ini akan mengaitkan webhook kita dengan bot, sehingga setiap pesan atau

Menginstall Project ke GitHub dan Deploy Django

perintah yang dikirimkan ke bot akan diteruskan ke aplikasi kita melalui webhook URL tersebut.

Selain Telegram, proses ini bisa diterapkan ke platform lainnya dengan format webhook dan endpoint yang sesuai. Dengan menggunakan webhook, aplikasi bot kita akan selalu siap menerima notifikasi secara real-time dari layanan yang digunakan.

Setelah webhook terpasang dan terhubung dengan aplikasi di Heroku, kita bisa memverifikasi apakah webhook berfungsi dengan baik dengan mengirimkan pesan percobaan ke bot dan melihat apakah aplikasi kita merespons dengan benar.

16.6.2 Mengaktifkan Webhook untuk Bot

Setelah kita mendapatkan URL webhook dari Heroku, langkah selanjutnya adalah menghubungkan webhook tersebut dengan bot yang sudah kita buat. Proses ini melibatkan pendaftaran URL webhook ke server bot yang digunakan, seperti Telegram.

Dengan mengaktifkan webhook, bot akan menerima pesan atau notifikasi dari server setiap kali ada aktivitas terkait, misalnya pengguna mengirimkan pesan ke bot.

Untuk bot Telegram, kita dapat menggunakan Telegram's API untuk mendaftarkan webhook dengan cepat. Telegram menyediakan endpoint untuk melakukan hal ini, dan kita dapat mengirimkan permintaan API untuk mengaitkan URL webhook dengan bot kita.

Menginstall Project ke GitHub dan Deploy Django

Langkah pertama adalah memastikan kita memiliki URL webhook dari Heroku, yang memiliki format seperti:

```
https://<app_name>.herokuapp.com/<bot_endpoint>/
```

Di mana <app_name> adalah nama aplikasi kita di Heroku dan <bot_endpoint> adalah rute di dalam Django yang kita tetapkan untuk menerima notifikasi dari webhook.

Untuk mengaktifkan webhook, kita dapat menggunakan perintah `curl` untuk mengirim permintaan ke API Telegram. Pastikan untuk mengganti token bot dan URL sesuai dengan konfigurasi kita:

```
$ curl -F "url=https://my-bot-app.herokuapp.com/webhook/"  
https://api.telegram.org/bot<token>/setWebhook
```

Dalam perintah ini, <token> adalah token unik yang kita dapatkan ketika membuat bot di Telegram, dan `https://my-bot-app.herokuapp.com/webhook/` adalah URL yang akan menerima update dari Telegram.

Setelah menjalankan perintah ini, Telegram akan mengirimkan pesan-pesan yang diterima bot ke URL tersebut. Kita bisa memeriksa apakah webhook sudah terhubung dengan baik melalui respons dari Telegram API. Jika webhook berhasil diaktifkan, respons dari Telegram API akan memberikan status sukses.

Verifikasi bahwa webhook berfungsi dapat dilakukan dengan mengirimkan pesan percobaan ke bot dan melihat apakah aplikasi

Menginstall Project ke GitHub dan Deploy Django

menerima dan memprosesnya sesuai yang diharapkan. Jika semuanya sudah berfungsi, bot kita siap menerima pesan secara real-time melalui webhook, tanpa perlu polling terus-menerus.

Dengan langkah ini, bot kita sudah sepenuhnya terhubung dengan webhook dan siap digunakan secara online, merespons setiap interaksi yang dikirimkan ke URL webhook yang terdaftar.

Menginstall Project ke GitHub dan Deploy Django

Kesimpulan Bab

Pada Bab ini, kita telah mempelajari langkah-langkah penting dalam mengelola dan mendistribusikan proyek Django secara online, mulai dari mengunggah project ke GitHub hingga melakukan deployment ke platform seperti Heroku. Langkah pertama yang kita lakukan adalah menghubungkan project lokal dengan repository di GitHub, yang memberikan keuntungan besar dalam hal kontrol versi dan kolaborasi.

Selanjutnya, kita menjelajahi proses deployment, termasuk pemilihan platform seperti Heroku, menyiapkan lingkungan deployment, serta mengatur `Procfile` dan `requirements.txt` untuk memastikan aplikasi dapat berjalan dengan baik di server produksi. Kita juga menekankan pentingnya pengaturan variabel lingkungan, seperti `SECRET_KEY`, pada platform deployment untuk menjaga keamanan aplikasi.

Tidak kalah pentingnya, kita membahas cara mendapatkan dan mengaktifkan webhook, yang menjadi bagian krusial dalam pengoperasian bot. Webhook memastikan bahwa aplikasi dapat menerima dan memproses pesan secara real-time dari pengguna, tanpa perlu polling yang intensif. Dengan menghubungkan URL webhook ke bot melalui API yang disediakan oleh Telegram, kita telah mengonfigurasi bot untuk dapat berjalan secara efisien di lingkungan produksi.

Menginstall Project ke GitHub dan Deploy Django

Keseluruhan bab ini memberikan panduan menyeluruh tentang bagaimana sebuah aplikasi Django, khususnya aplikasi bot, dapat diatur dan dijalankan secara online. Proses deployment ini membuka jalan bagi aplikasi kita untuk diakses secara global, memperluas jangkauan dan memastikan fungsionalitas bot berjalan optimal di server yang stabil. Pada akhirnya, kita siap meluncurkan project Django ke dunia nyata dengan menggunakan alat yang tersedia dan teknik yang efektif untuk memastikan semuanya berjalan lancar.

PENUTUP

Dalam buku ini, kita telah menjelajahi berbagai aspek dalam pengembangan aplikasi Django, terutama dalam konteks membangun platform manajemen bot. Dari memahami dasar-dasar Django hingga implementasi fitur-fitur yang lebih kompleks seperti validasi token dan integrasi berbagai platform bot, setiap bab telah dirancang untuk memberikan pembaca pemahaman yang komprehensif dan praktis tentang pengembangan aplikasi berbasis Django.

Selama perjalanan ini, kita juga membahas bagaimana memperbaiki tampilan dashboard, menyusun sistem log aktivitas, dan melakukan deployment proyek ke platform seperti GitHub dan Heroku. Kita telah belajar cara mengoptimalkan tampilan antarmuka pengguna dengan menggunakan berbagai komponen Bootstrap, serta bagaimana menyusun arsitektur proyek yang baik untuk mendukung pengembangan berkelanjutan. Dengan panduan yang terperinci ini, diharapkan pembaca, baik yang baru belajar Django maupun yang sudah berpengalaman, dapat mengembangkan proyek mereka sendiri dengan lebih terstruktur dan efisien.

Selain itu, buku ini juga menekankan pentingnya mengikuti perkembangan terbaru dalam ekosistem Django dan teknologi web secara umum. Dengan memanfaatkan dokumentasi resmi, tutorial, dan forum komunitas, pembaca didorong untuk terus belajar

dan mengeksplorasi kemungkinan baru dalam pengembangan aplikasi. Dalam dunia yang selalu berubah ini, kemampuan untuk beradaptasi dan mengembangkan keterampilan baru adalah kunci untuk sukses.

Dengan demikian, buku ini bukan hanya sekadar panduan teknis, tetapi juga diharapkan dapat menjadi sumber inspirasi bagi siapa pun yang ingin mengeksplorasi lebih lanjut tentang dunia pengembangan web menggunakan Django. Semoga setiap bab yang telah dibahas dapat membantu memantik ide-ide baru dan mendorong pembaca untuk menciptakan aplikasi yang inovatif dan bermanfaat. Saya berharap buku ini dapat menjadi pijakan yang solid dalam perjalanan kalian menuju pengembangan aplikasi yang lebih maju dan berdampak.

Harapan Untuk Pembaca

Saya berharap buku ini dapat memberikan wawasan dan pengetahuan yang berharga bagi pembaca dalam mengembangkan aplikasi berbasis Django, khususnya dalam konteks manajemen bot. Dengan memahami konsep dan praktik yang telah dijelaskan, saya berharap pembaca dapat:

1. ***Mendapatkan Inspirasi:*** Menggunakan informasi dan teknik yang disampaikan untuk mengembangkan ide-ide baru dan inovatif dalam proyek mereka sendiri. Dunia pengembangan web menawarkan banyak peluang, dan saya berharap pembaca dapat menemukan semangat untuk terus mengeksplorasi dan bereksperimen.
2. ***Meningkatkan Keterampilan:*** Setiap bab dirancang untuk tidak hanya memberikan pengetahuan teori, tetapi juga mempraktikkan keterampilan dalam pengembangan

aplikasi. Saya berharap pembaca dapat menerapkan apa yang mereka pelajari di dunia nyata dan memperdalam keterampilan teknis mereka.

3. ***Berinteraksi dengan Komunitas:*** Saya mendorong pembaca untuk terlibat dengan komunitas Django dan pengembangan lainnya. Berbagi pengalaman, bertanya, dan berkolaborasi dapat membuka jalan untuk pembelajaran yang lebih mendalam dan memperluas jaringan profesional mereka.
4. ***Menghadapi Tantangan:*** Dalam perjalanan pengembangan aplikasi, pasti akan ada tantangan yang dihadapi. Saya berharap pembaca dapat melihat tantangan sebagai kesempatan untuk belajar dan tumbuh, serta mampu menghadapi setiap rintangan dengan percaya diri.
5. ***Berinovasi:*** Akhirnya, saya berharap pembaca dapat menjadi inovator dalam bidang mereka. Dengan pengetahuan dan keterampilan yang diperoleh dari buku ini, saya percaya bahwa pembaca dapat menciptakan aplikasi yang bermanfaat dan memiliki dampak positif dalam komunitas mereka.

Dengan harapan ini, saya ingin mengingatkan bahwa perjalanan dalam pengembangan perangkat lunak adalah proses yang berkelanjutan. Semoga buku ini menjadi langkah awal bagi pembaca untuk terus belajar dan berkembang di dunia teknologi yang dinamis ini.

Struktur Akhir Proyek

Proyek Django ini bernama ***platform_bot***, yang dirancang untuk membangun dan mengelola platform manajemen bot. Struktur

proyek ini terbagi menjadi beberapa bagian penting yang masing-masing memiliki fungsionalitas spesifik.

Berikut adalah struktur proyek Django yang telah dibangun, yang diharapkan dapat membantu pembaca memahami bagaimana setiap komponen berinteraksi satu sama lain:

```
platform_bot/
├── apps/
│   ├── context_processors.py          # Berisi
fungsi untuk menambahkan konteks tambahan ke
dalam template.
│   ├── authentication/                # Aplikasi
untuk mengelola proses autentikasi pengguna
(login, register).
│   ├── dashboard/                    # Aplikasi
yang mengatur tampilan dashboard bagi pengguna.
│   ├── users/                        # Aplikasi
untuk mengelola profil pengguna.
│   ├── webhook/                      # Aplikasi
untuk menangani webhook dari bot.
│   ├── bots/                         # Aplikasi
yang mengelola semua aspek terkait bot.
│   │   ├── services/                 # Berisi
layanan untuk menangani integrasi dengan
platform bot.
│   │   │   ├── telegram.py           # Layanan
untuk interaksi dengan API Telegram.
│   │   │   ├── whatsapp.py           # Layanan
untuk interaksi dengan API WhatsApp.
│   │   ├── static/                   # Menyimpan
file statis seperti CSS, JavaScript, dan gambar.
│   │   │   ├── assets/                # Folder
untuk menyimpan aset statis.
│   │   │   ├── css/                  # Berisi
file CSS untuk styling.
│   │   │   │   ├── style.css          # File CSS
utama untuk aplikasi.
```

```

| | | | | js/ # Berisi
file JavaScript untuk interaktivitas.
| | | | | script.js # File
JavaScript utama untuk aplikasi.
| | | templates/ # Menyimpan
file template HTML untuk rendering.
| | | | | home/ # Folder
untuk halaman utama.
| | | | | base.html # Template
dasar untuk halaman utama.
| | | | | home.html # Halaman
utama aplikasi.
| | | | | account/ # Folder
untuk halaman terkait akun pengguna.
| | | | | register.html # Halaman
untuk registrasi pengguna baru.
| | | | | login.html # Halaman
untuk login pengguna.
| | | | | dashboard/ # Folder
untuk halaman dashboard.
| | | | | base.html # Template
dasar untuk dashboard.
| | | | | sidebar.html # Template
untuk sidebar di dashboard.
| | | | | navbar.html # Template
untuk navbar di dashboard.
| | | | | dashboard.html # Halaman
utama dashboard.
| | | | | users/ # Folder
untuk halaman profil pengguna.
| | | | | profile.html # Halaman
untuk menampilkan dan mengedit profil pengguna.
| | | | | bots/ # Folder
untuk halaman yang terkait dengan bot.
| | | | | create_bot.html # Halaman
untuk membuat bot baru.
| | | | | manage_bots.html # Halaman
untuk mengelola daftar bot.
| | | | | edit_bot.html # Halaman
untuk mengedit informasi bot yang sudah ada.
| | | | | add_command.html # Halaman
untuk menambahkan perintah baru ke bot.

```


- ***platform_bot/***: Ini adalah folder utama proyek yang menyimpan pengaturan dan konfigurasi inti aplikasi Django, termasuk file pengaturan, URL, dan manajemen server.
- ***manage.py***: Skrip ini digunakan untuk menjalankan perintah manajemen proyek, seperti menjalankan server, melakukan migrasi database, dan banyak lagi.
- ***db.sqlite3***: Database SQLite yang menyimpan semua data aplikasi, termasuk informasi pengguna, bot, dan perintah yang ditambahkan.

Dengan memahami struktur proyek ini, pembaca diharapkan dapat melihat bagaimana berbagai komponen bekerja sama untuk menciptakan platform manajemen bot yang fungsional dan efisien. Setiap bagian dari proyek memiliki peran penting dan saling terintegrasi, memungkinkan pengembangan aplikasi yang lebih terstruktur dan mudah dipelihara.

Link Proyek GitHub

Sebagai tambahan, kalian dapat melihat proyek platform manajemen bot yang telah kita bahas dalam buku ini pada tautan GitHub berikut:

[GitHub Project - Platform Bot](#)

Melalui repositori tersebut, kalian dapat mengakses kode sumber lengkap dari proyek yang kita kembangkan dalam buku ini, serta

berkontribusi jika kalian tertarik untuk mengembangkannya lebih lanjut.

UCAPAN TERIMA KASIH

Saya ingin mengucapkan terima kasih kepada semua pihak yang telah mendukung saya dalam proses penulisan buku ini. Terima kasih kepada keluarga, teman, dan rekan-rekan pengembang yang selalu memberikan motivasi dan inspirasi. Dukungan moral dan kehadiran mereka dalam perjalanan ini sangat berarti bagi saya, dan saya tidak akan bisa menyelesaikan buku ini tanpa semangat yang mereka berikan.

Ucapan terima kasih khusus juga saya sampaikan kepada komunitas Django yang selalu aktif berbagi pengetahuan dan sumber daya yang sangat bermanfaat. Komunitas ini bukan hanya menjadi tempat untuk bertanya dan belajar, tetapi juga menjadi sumber ide dan inovasi yang tidak ternilai. Tanpa komunitas yang kuat, pengembangan aplikasi open-source seperti Django tidak akan sebesar dan seberdampak seperti sekarang. Keberadaan forum, dokumentasi, dan berbagai tutorial dari anggota komunitas telah banyak membantu saya dalam memperdalam pemahaman dan keterampilan saya.

Saya juga berterima kasih kepada semua pembaca yang telah meluangkan waktu untuk membaca buku ini. Keterlibatan dan umpan balik dari kalian sangat berarti bagi saya. Saya berharap buku ini dapat memberikan manfaat besar bagi perjalanan kalian dalam pengembangan aplikasi Django. Semoga setiap konsep yang dibahas, mulai dari dasar hingga fitur-fitur lanjutan, dapat

menjadi alat yang berguna dalam mencapai tujuan pengembangan kalian.

Akhir kata, saya mendorong pembaca untuk terus mengeksplorasi dan belajar. Dunia teknologi informasi terus berkembang, dan menjadi pembelajar seumur hidup adalah salah satu kunci untuk tetap relevan. Jangan ragu untuk berbagi pengetahuan yang telah kalian dapatkan dengan orang lain, karena kolaborasi dan berbagi informasi adalah cara yang efektif untuk memperkuat komunitas kita. Semoga kita semua dapat bersama-sama berkontribusi untuk dunia pengembangan web yang lebih baik dan lebih inovatif. Terima kasih.

Terima kasih telah mengikuti perjalanan ini, dan semoga sukses dalam semua proyek kalian di masa depan!

dccccxxv

MEMBANGUN PLATFORM BOT DENGAN DJANGO

**PANDUAN LANGKAH DEMI LANGKAH UNTUK MEMBANGUN
PLATFORM PEMBUATAN BOT DARI AWAL**

BUKU INI MEMBAHAS SECARA MENDALAM TENTANG CARA MEMBANGUN PLATFORM BOT MENGGUNAKAN FRAMEWORK DJANGO. DENGAN PENDEKATAN PRAKTIS DAN CONTOH NYATA, PEMBACA AKAN DIAJAK UNTUK MEMPELAJARI PENGEMBANGAN BOT YANG DAPAT BERKOMUNIKASI MELALUI PLATFORM SEPERTI TELEGRAM DAN WHATSAPP, MENGGUNAKAN WEBHOOK SEBAGAI METODE INTERAKSI YANG EFEKTIF. MULAI DARI PERANCANGAN, IMPLEMENTASI, HINGGA DEPLOYMENT KE SERVER, BUKU INI ADALAH PANDUAN YANG TEPAT UNTUK SIAPA PUN YANG INGIN MEMULAI ATAU MEMPERDALAM PEMAHAMAN MEREKA TENTANG PENGEMBANGAN BOT DARI AWAL HINGGA DEPLOYMENT DENGAN CONTOH NYATA YANG BISA LANGSUNG DITERAPKAN.

