

MODUL PRAKTIKUM

ARSITEKTUR DAN DESAIN PERANGKAT LUNAK

S1 REKAYASA PERANGKAT LUNAK



Published by school of computing

Our official Instagram



@informaticslab_telu

Lembar Pengesahan

Saya yang bertanda tangan di bawah ini:

Nama : Dana Sulistyo Kusumo, S.T., M.T., Ph.D.
NIK : 02780011
Koordinator Mata Kuliah : Arsitektur Desain Perangkat Lunak
Prodi : S1 Rekayasa Perangkat Lunak

Menerangkan dengan sesungguhnya bahwa modul ini digunakan untuk pelaksanaan praktikum di Semester Genap Tahun Ajaran 2024/2025 di Laboratorium Informatika Fakultas Informatika Universitas Telkom.

Bandung, 14 Februari 2025



Mengesahkan,
Koordinator Mata Kuliah

Dana Sulistyo Kusumo, S.T., M.T., Ph.D.
NIP. 02780011



Fakultas Informatika
School of Computing
Telkom Unive

Mengetahui,
Kaprodi S1

Dr. Mira Kania Sabariah, S.T., M.T.
NIP. 14770011

Peraturan Praktikum Laboratorium Informatika 2024/2025

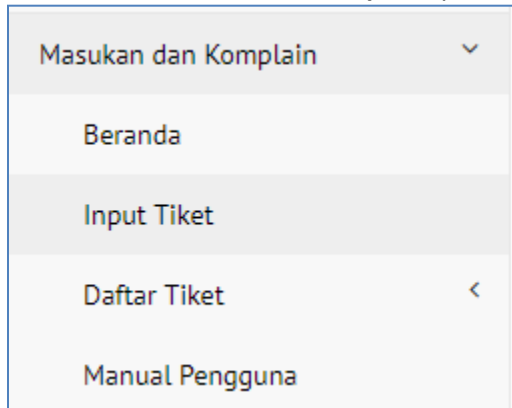
1. Praktikum diampu oleh dosen kelas dan dibantu oleh asisten laboratorium dan asisten praktikum.
2. Praktikum dilaksanakan di Gedung TULT (Telkom University Landmark Tower) lantai 6 dan 7 sesuai jadwal yang ditentukan.
3. Praktikan wajib membawa modul praktikum, kartu praktikum, dan alat tulis.
4. Praktikan wajib mengecek kehadiran di igracias dan sheet yang dibagikan asisten.
5. Durasi kegiatan praktikum S-1 = 2 jam (100 menit).
6. Jumlah pertemuan praktikum:
 - 16 kali pertemuan
7. Praktikan wajib hadir minimal 75% dari seluruh pertemuan praktikum di lab.
8. Praktikan yang datang terlambat :
 - ≤ 5 menit : diperbolehkan mengikuti praktikum tanpa tambahan praktikum
 - ≥ 30 menit : tidak diperbolehkan mengikuti praktikum
9. Saat praktikum berlangsung, asisten praktikum dan praktikan:
 - Wajib menggunakan seragam sesuai aturan institusi.
 - Wajib mematikan/ mengkondisikan semua alat komunikasi.
 - Dilarang membuka aplikasi yang tidak berhubungan dengan praktikum yang berlangsung.
 - Dilarang mengubah pengaturan *software* maupun *hardware* komputer tanpa ijin.
 - Dilarang membawa makanan maupun minuman di ruang praktikum.
 - Dilarang memberikan jawaban ke praktikan lain.
 - Dilarang menyebarkan soal praktikum.
 - Dilarang membuang sampah di ruangan praktikum.
 - Wajib meletakkan alas kaki dengan rapi pada tempat yang telah disediakan.
10. Setiap praktikan dapat mengikuti praktikum susulan maksimal dua modul untuk satu mata kuliah praktikum.
 - Praktikan yang dapat mengikuti praktikum susulan hanyalah praktikan yang memenuhi syarat sesuai ketentuan institusi, yaitu: sakit (dibuktikan dengan surat keterangan medis), tugas dari institusi (dibuktikan dengan surat dinas atau dispensasi dari institusi), atau mendapat musibah atau keduakaan (menunjukkan surat keterangan dari orangtua/wali mahasiswa.)
 - Persyaratan untuk praktikum susulan diserahkan sesegera mungkin kepada asisten laboratorium untuk keperluan administrasi.
 - Praktikan yang diijinkan menjadi peserta praktikum susulan ditetapkan oleh Lab Informatika dan tidak dapat diganggu gugat.
11. a. Ketidakhadiran pada kelas praktikum
 - **Nilai modul = 0**
- b. Meminta, mendapatkan, dan menyebarluaskan soal dan atau kunci jawaban praktikum:
 - **Penyebarnya soal dan kunci jawaban : Pengajuan sanksi kepada Komisi Disiplin Fakultas**
 - **Penerima soal dan kunci jawaban: Nilai '0' pada (seluruh assessment) praktikum**
- c. Lupa menghapus file praktikum
 - **Pengurangan nilai modul 20%**
- d. Memicu kegaduhan, sehingga membuat situasi tidak kondusif (jalan-jalan, mengganggu teman, mengobrol, dll), asisten praktikum diwajibkan menegur sebanyak 3x.

- Pengurangan nilai modul 50%
- e. MENYALAHGUNAKAN FITUR LMS
- Siap menerima sanksi dari lab

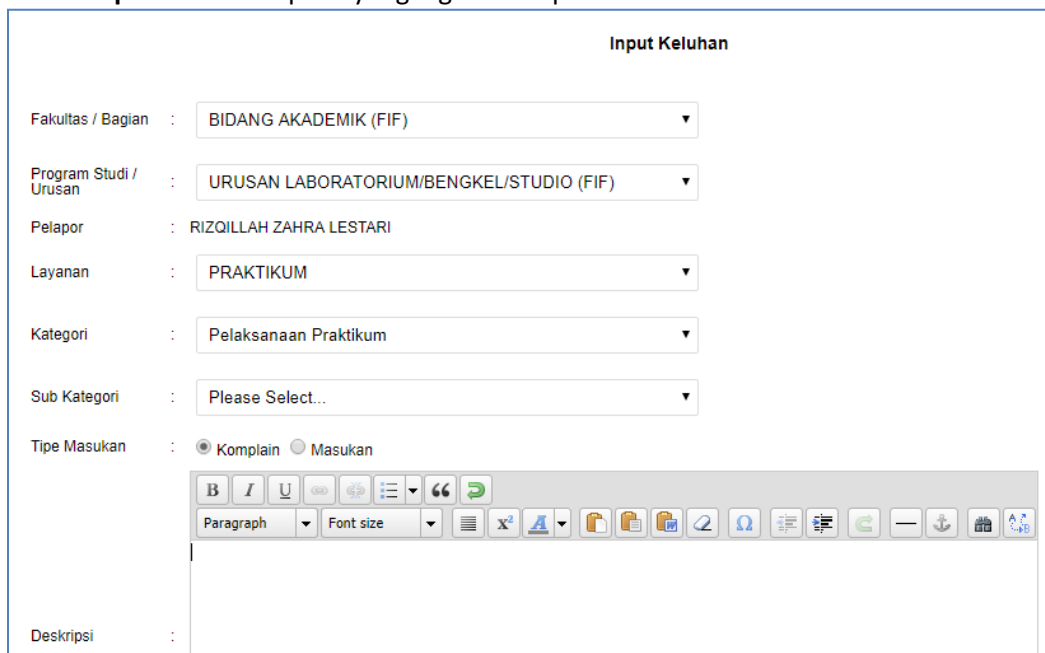


Tata Cara Komplain Praktikum IFLAB Melalui IGRACIAS

1. Login Igracias.
2. Pilih Menu **Masukan dan Komplain**, pilih **Input Tiket**.



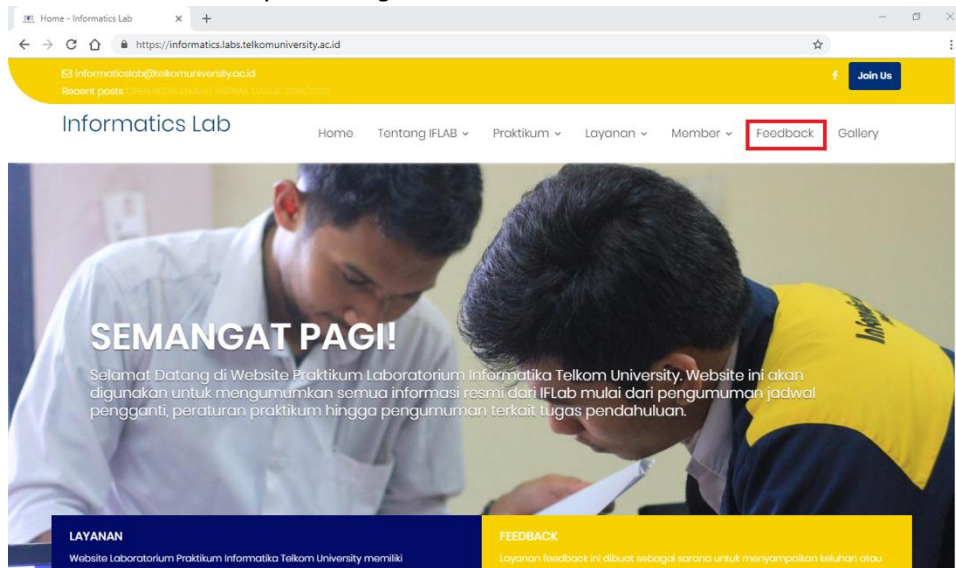
3. Pilih Fakultas/Bagian: **Bidang Akademik (FIF)**.
4. Pilih Program Studi/Urusan: **Urusan Laboratorium/Bengkel/Studio (FIF)**.
5. Pilih Layanan: **Praktikum**.
6. Pilih Kategori: **Pelaksanaan Praktikum**, lalu pilih **Sub Kategori**.
7. Isi **Deskripsi** sesuai komplain yang ingin disampaikan.

A screenshot of the 'Input Keluhan' form. The form contains several dropdown menus and a text input field. The fields are: 'Fakultas / Bagian' (set to 'BIDANG AKADEMIK (FIF)'), 'Program Studi / Urusan' (set to 'URUSAN LABORATORIUM/BENGKEL/STUDIO (FIF)'), 'Pelapor' (set to 'RIZQILLAH ZAHRA LESTARI'), 'Layanan' (set to 'PRAKTIKUM'), 'Kategori' (set to 'Pelaksanaan Praktikum'), 'Sub Kategori' (set to 'Please Select...'), and 'Tipe Masukan' (with radio buttons for 'Komplain' and 'Masukan', where 'Komplain' is selected). Below these fields is a rich text editor with a toolbar containing icons for bold, italic, underline, bulleted list, numbered list, link, unlink, and other formatting options. The 'Deskripsi' field is currently empty.

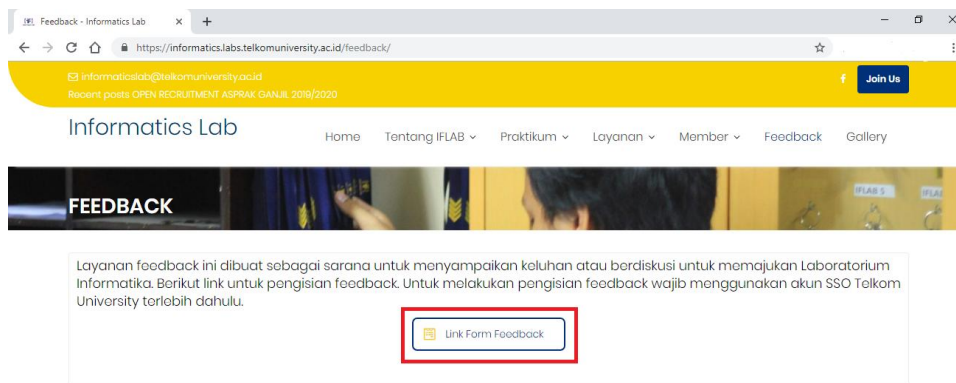
8. Lampirkan *file* jika perlu. Lalu klik Kirim.

Tata Cara Komplain Praktikum IFLAB Melalui Website

1. Buka website <https://informatics.labs.telkomuniversity.ac.id/> melalui browser.
2. Pilih menu **Feedback** pada *navigation bar website*.



3. Pilih tombol **Link Form Feedback**.



4. Lakukan *login* menggunakan akun **SSO Telkom University** untuk mengakses *form feedback*.
5. Isi *form* sesuai dengan *feedback* yang ingin diberikan.

Daftar Isi

Lembar Pengesahan.....	i
Peraturan Praktikum Laboratorium Informatika 2024/2025	ii
Tata Cara Komplain Praktikum IFLAB Melalui IGRACIAS	iv
Tata Cara Komplain Praktikum IFLAB Melalui <i>Website</i>	v
Daftar Isi.....	vi
Daftar Gambar	x
Modul 1 Running Modul.....	12
1.1 Pengenalan Java.....	12
1.1.1 Sejarah Singkat Java	12
1.1.2 Karakteristik Java	12
1.1.3 Garbage Collection.....	12
1.2 Instalasi Eclipse	13
1.2.1 Instalasi JDK (Java Development Kit)	13
1.2.2 Eclipse	14
Modul 2 Graphical User Interface (GUI) Part 1	19
2.1 AWT dan Swing	19
2.2 Komponen utama dalam GUI.....	19
2.3 The Basic User Interface Components with Swing	20
2.4 Top Level Container	20
2.4.1 JFrame	20
2.4.2 JDialog	21
2.5 GUI Drag and Drop.....	22
2.6 Pengaturan Layout dalam Swing.....	23
2.6.1 BorderLayout	23
2.6.2 BoxLayout.....	24
2.6.3 CardLayout	25
2.6.4 FlowLayout.....	26
2.6.5 GridLayout.....	27
2.7 Event Handling	28
2.7.1 Button (Jbutton).....	28
2.7.2 ComboBox (JComboBox).....	29
2.7.3 CheckBox (JCheckBox).....	30
2.7.4 RadioButton (JRadioButton)	31
2.7.5 Table (JTable)	31
Modul 3 Graphical User Interface (GUI) Part 2 & MVC.....	34
3.1 Pendahuluan	34

3.2	Model	35
3.3	View	36
3.4	Controller	37
3.5	Main	38
Modul 4	GUI Part 3 & Java Database Connectivity (JDBC)	39
4.1	Memulai JDBC	39
4.2	Menghubungkan Java dengan Database	39
4.2.1	Cara Kerja JDBC	39
4.2.2	Menambahkan JDBC Driver di Eclipse	40
4.3	Menggunakan Database dalam Java	41
4.3.1	Class Database untuk Koneksi	41
4.3.2	Insert / Create Data	42
4.3.3	Read Data	43
4.3.4	Edit/ Update Data	44
4.3.5	Delete Data	44
4.3.6	Contoh Penerapan MVC dengan JDBC (CRUD)	45
Modul 5	Prinsip SOLID Part 1	49
5.1	Pendahuluan tentang SOLID	49
5.2	Single Responsibility Principle (SRP)	49
5.2.1	Definisi dan Konsep	49
5.2.2	Contoh Kode Tanpa SRP (Kode Buruk)	50
5.2.3	Contoh Kode yang Mematuhi SRP (Kode Baik)	50
5.3	Open-Closed Principle (OCP)	51
5.3.1	Definisi dan Konsep	51
5.3.2	Contoh Kode Tanpa OCP (Kode Buruk)	51
5.3.3	Contoh Kode yang Mematuhi OCP (Kode Baik)	51
5.4	Liskov Substitution Principle (LSP)	52
5.4.1	Definisi dan Konsep	52
5.4.2	Contoh Kode yang Melanggar LSP (Kode Buruk)	53
5.4.3	Contoh Kode yang Mematuhi LSP (Kode Baik)	54
Modul 6	Prinsip SOLID Part 2	55
6.1	Pendahuluan	55
6.2	Interface Segregation Principle (ISP)	55
6.2.1	Definisi dan Konsep	55
6.2.2	Contoh Kode yang Melanggar ISP (Kode Buruk)	55
6.2.3	Contoh Kode yang Mematuhi ISP (Kode Baik)	56
6.3	Dependency Inversion Principle (DIP)	57

6.3.1	Definisi dan Konsep.....	57
6.3.2	Contoh Kode yang Melanggar DIP (Kode Buruk)	57
6.3.3	Contoh Kode yang Mematuhi DIP (Kode Baik)	58
Modul 7	Penerapan SOLID Dalam Studi Kasus Pemrograman	60
7.1	Pendahuluan	60
7.2	Studi Kasus dan Full Code	60
7.3	Single Responsibility Principle (SRP)	63
7.4	Open-Closed Principle (OCP).....	63
7.5	Liskov Substitution Principle (LSP)	64
7.6	Interface Segregation Principle (ISP).....	64
7.7	Dependency Inversion Principle (DIP).....	65
Modul 8	COTS Penerapan Tubes 1	66
8.1	COTS Penerapan Tugas Besar 1	66
Modul 9	SonarQube	67
9.1	Pendahuluan	67
9.2	Prequisite	67
9.3	Hardware Requirement	67
9.4	Instalasi SonarQube	68
9.5	Membuat dan Menganalisis Project	69
9.6	Project Tabs.....	74
9.6.1	Overview Tab	74
9.6.2	Issues Tab.....	74
9.6.3	Security Hotspot Tab.....	76
9.6.4	Measures Tab.....	77
9.6.5	Code Tab	77
9.6.6	Activity Tab.....	78
Modul 10	REST API Spring Boot.....	79
10.1	Pengenalan API	79
10.1.1	Apa itu API?	79
10.1.2	Fungsi dan Manfaat API?	79
10.1.3	Prinsip Desain API yang Baik	79
10.2	REST API	80
10.2.1	Apa itu REST API?	80
10.2.2	Prinsip Desain REST API.....	81
10.2.3	Bagaimana REST API bekerja?	81
10.2.4	Metode HTTP dalam REST API	82
10.2.5	Keuntungan dan Tantangan REST API.....	82

10.3	REST API Spring Boot di Eclipse.....	82
Modul 11	Architectural Pattern: Monolithic.....	92
11.1	Monolithic Architecture.....	92
11.1.1	Karakteristik Monolithic Architecture.....	92
11.1.2	Kelebihan dan Kekurangan Monolithic Architecture.....	93
11.2	Implementasi Monolithic dengan Spring Boot	93
Modul 12	Architectural Pattern: Microservice.....	102
12.1	Microservices Architecture	102
12.1.1	Karakteristik Microservices Architecture	102
12.1.2	Kelebihan dan Kekurangan Microservices Architecture	103
12.2	Implementasi Microservices dengan Spring Boot	103
12.2.1	Membuat Spring Boot Project untuk Setiap Layanan di Eclipse	103
12.2.2	Komunikasi Antar Service	112
Modul 13	JMETER Part 1 Performance Testing, Load Testing, & Stress Testing.....	115
13.1	Pendahuluan	115
13.2	Prequisite	115
13.3	Instalasi JMeter	115
13.4	Antarmuka JMeter	115
13.4.1	Thread Group.....	116
13.4.2	Listener.....	116
13.4.3	Sampler.....	117
13.4.4	Assertion	117
13.4.5	Timer	117
13.4.6	Processor.....	117
13.5	Performance Test.....	117
13.5.1	Load Test	119
13.5.2	Stress Test	119
Modul 14	JMETER Part 2 Functional Testing & API Testing	121
14.1	Functional Testing dengan JMeter.....	121
14.2	API Testing dengan JMeter	121
14.2.1	Membuat Functional Test pada API menggunakan JMeter.....	121
Modul 15	COTS Penerapan Tubes 2	125
15.1	COTS Penerapan Tugas Besar 2	125
	Daftar Pustaka.....	126

Daftar Gambar

Gambar 1-1 <i>Download</i> JDK pada laman Oracle pada platform Windows	13
Gambar 1-2 Tampilan awal instalasi Java (JDK 23.0.2).	14
Gambar 1-3 Pemilihan tempat penginstalan Java.	14
Gambar 1-4 <i>Download Eclipse installer</i>	15
Gambar 1-5 <i>Eclipse installer</i>	15
Gambar 1-6 Eclipse Marketplace	16
Gambar 1-7 Ekstensi Spring Tools dan WindowBuilder.....	16
Gambar 1-8 Project baru	17
Gambar 1-9 Membuat Class baru	17
Gambar 1-10 Tampilan Run Main Project.	18
Gambar 1-11 Tampilan hasil <i>compile</i> dan <i>run</i> program <i>hello world</i>	18
Gambar 2-1 Membuat JFrame Generate File	21
Gambar 2-2 Membuat JDialog Generate File	22
Gambar 2-3 Komponen GUI Builder	23
Gambar 2-4 Border Layout	23
Gambar 2-5 Box Layout.....	24
Gambar 2-6 Card Layout.....	25
Gambar 2-7 Flow Layout.....	26
Gambar 2-8 Grid Layout.....	27
Gambar 2-9 Button	28
Gambar 2-10 ComboBox.....	29
Gambar 2-11 CheckBox.....	30
Gambar 2-12 RadioButton	31
Gambar 2-13 Table.....	32
Gambar 3-1 Susunan Direktori.....	35
Gambar 4-1 Berhasil terkoneksi ke Database	41
Gambar 9-1 Cek Versi Java.....	68
Gambar 9-2 Landing Page SonarQube	68
Gambar 9-3 Halaman Update Password.....	69
Gambar 9-4 Halaman Utama SonarQube	69
Gambar 9-5 Inisialisasi Proyek Baru 1.....	69
Gambar 9-6 Inisialisasi Proyek Baru 2.....	70
Gambar 9-7 Menggunakan proyek lokal.....	70
Gambar 9-8 <i>Generate Token</i>	71
Gambar 9-9 Download SonarScanner.....	71
Gambar 9-10 <i>Edit Environment Variables</i>	72
Gambar 9-11 Buka CMD	72
Gambar 9-12 Copy Script Scanner	72
Gambar 9-13 Proses Analisis Berhasil.....	72
Gambar 9-14 Halaman Utama dengan Proyek Berhasil di Analisis	72
Gambar 9-15 Ringkasan Hasil Analisis 1	73
Gambar 9-16 Ringkasan Hasil Analisis 2	74
Gambar 9-17 Issues Tab 1.....	74
Gambar 9-18 Issues Tab 2.....	74
Gambar 9-19 Issues Tab 3.....	75
Gambar 9-20 Issues Tab 4.....	75
Gambar 9-21 Issues Tab 5.....	75
Gambar 9-22 Issues Tab 6.....	75

Gambar 9-23 Issues Tab 7	76
Gambar 9-24 Security Hotspot Tab.....	77
Gambar 9-25 Measures Tab.....	77
Gambar 9-26 Code Tab	78
Gambar 9-27 Activity Tab	78
Gambar 10-1 Menu XAMPP	85
Gambar 10-2 Query Membuat Database	85
Gambar 10-3 Query Membuat Table Product	86
Gambar 10-4 Query Memasuk-kan Data kedalam Table Product.....	86
Gambar 10-5 Uji API GET	89
Gambar 10-6 Uji API POST	90
Gambar 10-7 Uji API PUT	90
Gambar 10-8 Uji API DELETE	91
Gambar 11-1. Monolith Architecture	92
Gambar 12-1. Microservices Architecture	102
Gambar 12-2. Folder-folder project.....	103
Gambar 12-3 Eureka Server Workspace.....	104
Gambar 12-4 Membuat Proyek Spring 1	104
Gambar 12-5 Isi Detail Proyek	105
Gambar 12-6 Menambah Dependencies Proyek 1.....	105
Gambar 12-7 Membuat Proyek Spring 2	107
Gambar 12-8 Menambah Dependencies Proyek 1.....	107
Gambar 12-9 Struktur Folder.....	108
Gambar 13-1 Tampilan Utama JMETER.....	116
Gambar 13-2 Menambahkan Tread Group.....	117
Gambar 13-3 Pengaturan Parameter.....	118
Gambar 13-4 Graph Hasil.....	118
Gambar 13-5 Result Tree	118
Gambar 13-6 Ringkasan Hasil Pengujian	119
Gambar 14-1 Menambahkan Tread Group.....	122
Gambar 14-2Menambahkan Sampler.....	122
Gambar 14-3 Menambahkan Header Manager.....	122
Gambar 14-4 Response Assertion.....	123
Gambar 14-5 JSON Assertion	123
Gambar 14-6 Pengujian Berhasil.....	124
Gambar 14-7 Pengujian Gagal.	124

Modul 1 Running Modul

Tujuan Praktikum

1. Mengetahui dan mengenal bahasa pemrograman java
2. Mengetahui cara instalasi dan konfigurasi java pada platform Eclipse

1.1 Pengenalan Java

1.1.1 Sejarah Singkat Java



Java dibuat dan diperkenalkan pertama kali oleh sebuah tim Sun Microsystem yang dipimpin oleh **Patrick Naughton** dan **James Gosling** pada tahun 1991 dengan nama kode **Oak**. Tahun 1995 Sun mengubah nama Oak tersebut menjadi **Java**.

Teknologi java diadopsi oleh Netscape tahun 1996. JDK 1.1 diluncurkan tahun 1996, kemudian JDK 1.2, berikutnya J2EE (Java 2 Enterprise Edition) yang berbasis J2SE yaitu servlet, EJB dan JSP. Dan yang terakhir adalah J2ME (Java 2 Micro Edition) yang diadopsi oleh Nokia, Siemens, Motorola, Samsung, dan Sony Ericson



Tahukah kamu?

Ide pertama kali kenapa Java dibuat adalah karena adanya motivasi untuk membuat sebuah bahasa pemrograman yang bersifat *portable* dan *platform independent* (tidak tergantung mesin dan sistem operasi) yang dapat digunakan untuk membuat piranti lunak yang dapat ditanamkan (*embedded*) pada berbagai macam peralatan *electronic consumer* biasa, seperti *microwave*, *remote control*, telepon, *card reader* dan sebagainya.

1.1.2 Karakteristik Java

Karakteristik Java adalah sebagai berikut :

- 1) Berorientasi Objek, Java telah menerapkan konsep pemrograman berorientasi objek yang modern dalam implementasinya.
- 2) *Robust*, java mendorong pemrograman yang bebas dari kesalahan dengan bersifat *strongly typed* dan memiliki *run-time checking*
- 3) *Portable*, Program Java dapat dieksekusi di platform manapun selama tersedia Java Virtual Machine untuk *platform* tersebut.
- 4) *Multithreading*, Java mendukung penggunaan *multithreading* yang telah terintegasi langsung dalam bahasa Java
- 5) Dinamis, Program Java dapat melakukan suatu tindakan yang ditentukan pada saat eksekusi program dan bukan pada saat kompilasi
- 6) Sederhana, Java menggunakan bahasa yang sederhana dan mudah dipelajari.
- 7) Terdistribusi, java didesain untuk berjalan pada lingkungan yang terdistribusi seperti halnya internet.
- 8) Aman, Java memiliki arsitektur yang kokoh dan pemrograman yang aman.

1.1.3 Garbage Collection

Teknik yang digunakan Java untuk penanganan objek yang telah tidak diperlukan untuk dimusnahkan (dikembalikan ke *pool memory*) disebut garbage collection. *Java Interpreter* melakukan pemeriksaan untuk mengetahui apakah objek di memori masih diacu oleh program.

Java Interpreter akan mengetahui objek yang telah tidak dipakai oleh program. Ketika *Java interpreter* mengetahui objek itu, maka *Java interpreter* akan memusnahkan secara aman.

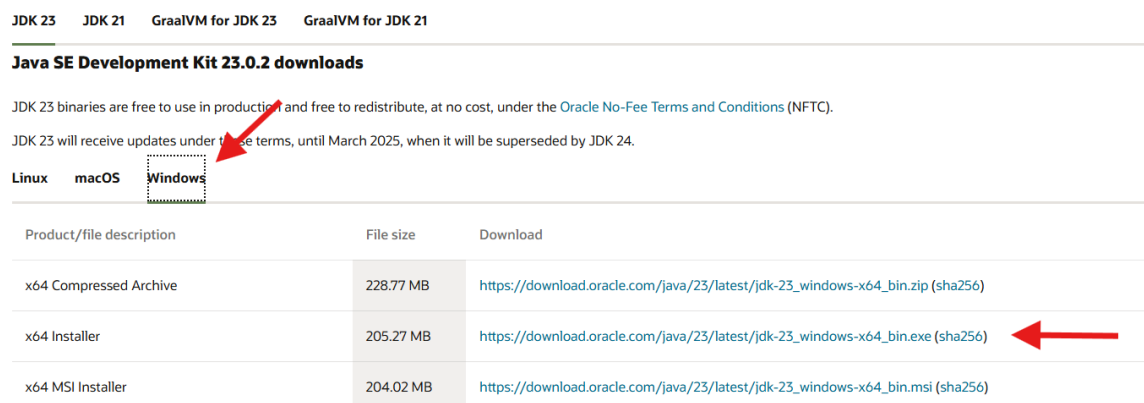
Java garbage collector berjalan sebagai thread berprioritas rendah dan melakukan kerja saat tidak ada kerja lain di program. Umumnya, *Java garbage collector* bekerja saat idle pemakai menunggu masukan dari *keyboard* atau *mouse*. *Garbage collector* akan bekerja dengan prioritas tinggi saat interpreter kekurangan memori. Hal ini jarang terjadi karena *thread* berprioritas rendah telah melakukan tugas dengan baik secara background.

1.2 Instalasi Eclipse

Sebelum *install* Eclipse Kita perlu *install* JDK terlebih dahulu.

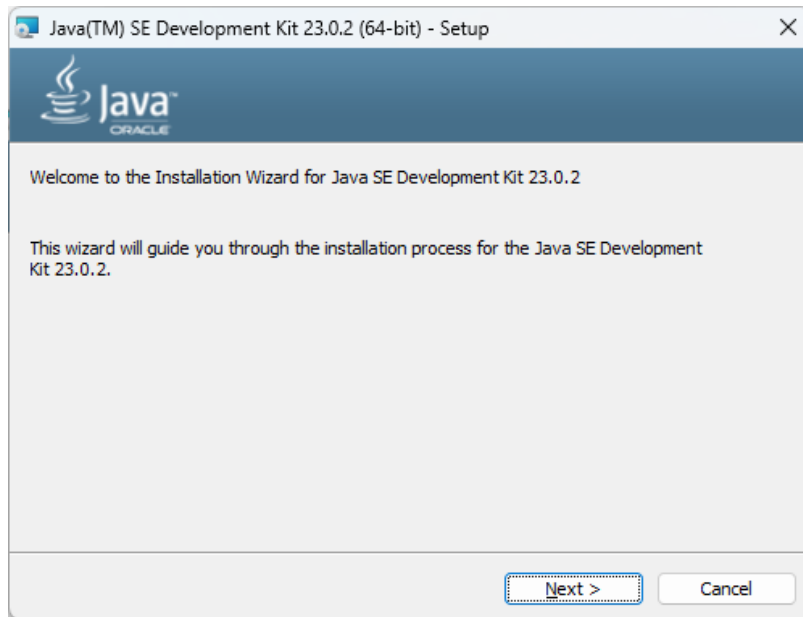
1.2.1 Instalasi JDK (Java Development Kit)

Pada Windows, *file* instalasi berupa **self-installing file**, yaitu *file exe* yang bila dijalankan akan mengekstrak dirinya untuk dikopikan ke direktori. File **exe** ini dapat didownload melalui laman resmi Oracle, yaitu pada *link* berikut <https://www.oracle.com/id/java/technologies/downloads/>. Prosedur instalasi akan menawarkan *default* untuk direktori instalasi seperti jdk 17.0.12, jdk 21.0.4, atau yang terbaru jdk 23.0.2. Kita sebaiknya tidak mengubahnya karena Kita dapat mengkoleksi beragam versi JDK, secara *default*.

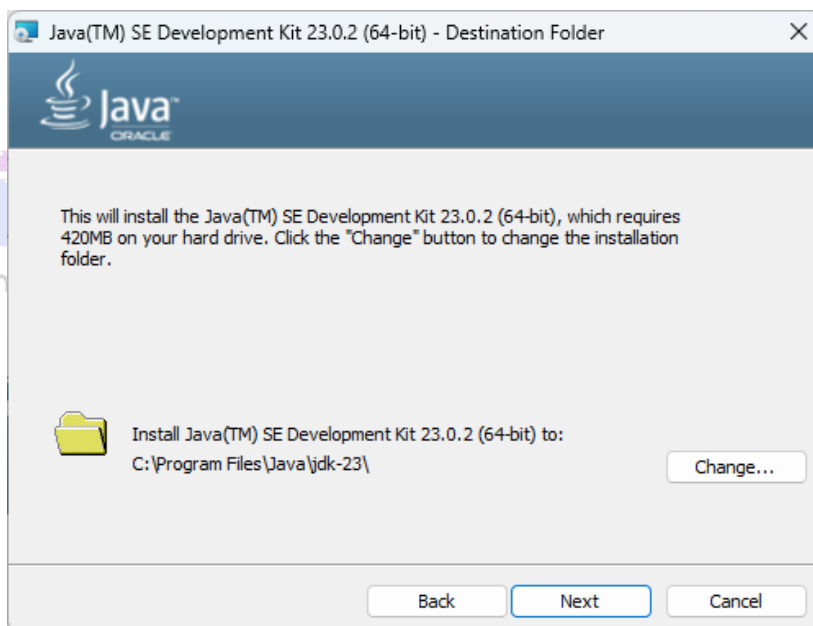


JDK 23	JDK 21	GraalVM for JDK 23	GraalVM for JDK 21
Java SE Development Kit 23.0.2 downloads			
JDK 23 binaries are free to use in production and free to redistribute, at no cost, under the Oracle No-Fee Terms and Conditions (NFTC).			
JDK 23 will receive updates under these terms, until March 2025, when it will be superseded by JDK 24.			
Linux	macOS	Windows	
Product/file description		File size	Download
x64 Compressed Archive		228.77 MB	https://download.oracle.com/java/23/latest/jdk-23_windows-x64_bin.zip (sha256)
x64 Installer		205.27 MB	https://download.oracle.com/java/23/latest/jdk-23_windows-x64_bin.exe (sha256)
x64 MSI Installer		204.02 MB	https://download.oracle.com/java/23/latest/jdk-23_windows-x64_bin.msi (sha256)

Gambar 1-1 *Download* JDK pada laman Oracle pada platform Windows



Gambar 1-2 Tampilan awal instalasi Java (JDK 23.0.2).

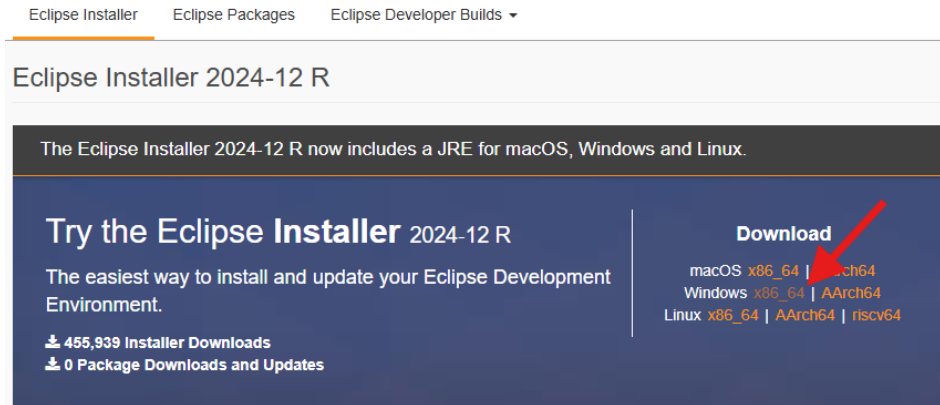


Gambar 1-3 Pemilihan tempat penginstalan Java.

Selanjutnya klik “Next” dan tinggal mengikuti semua alur instalasi sampai proses instalasi selesai.

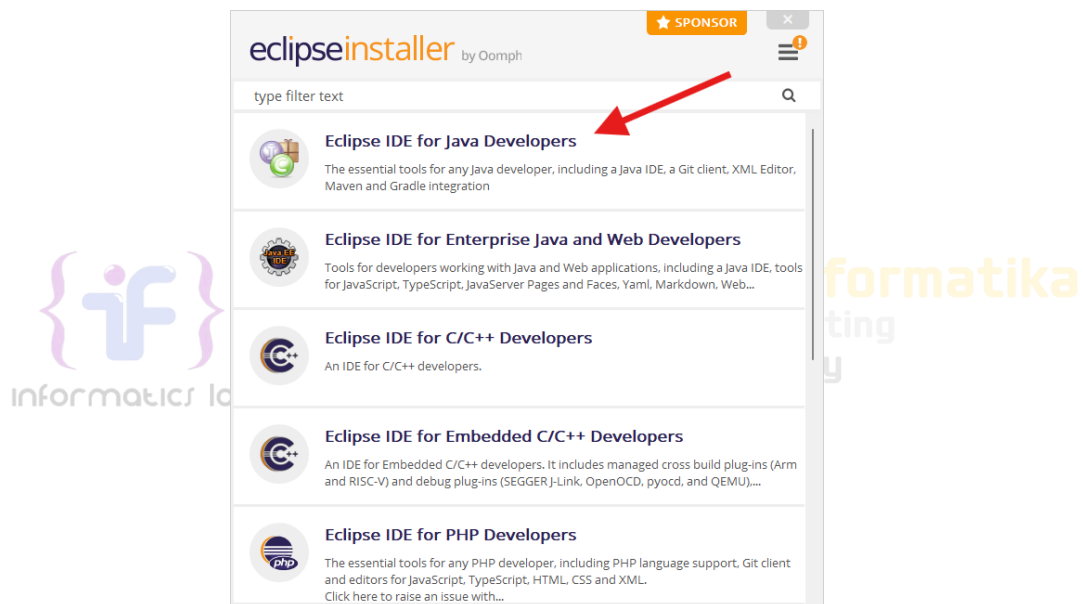
1.2.2 Eclipse

Setelah instalasi JDK selesai, tahap selanjutnya adalah melakukan instalasi Eclipse terbaru yang dapat diunduh di <https://www.eclipse.org/downloads/packages/installer>.



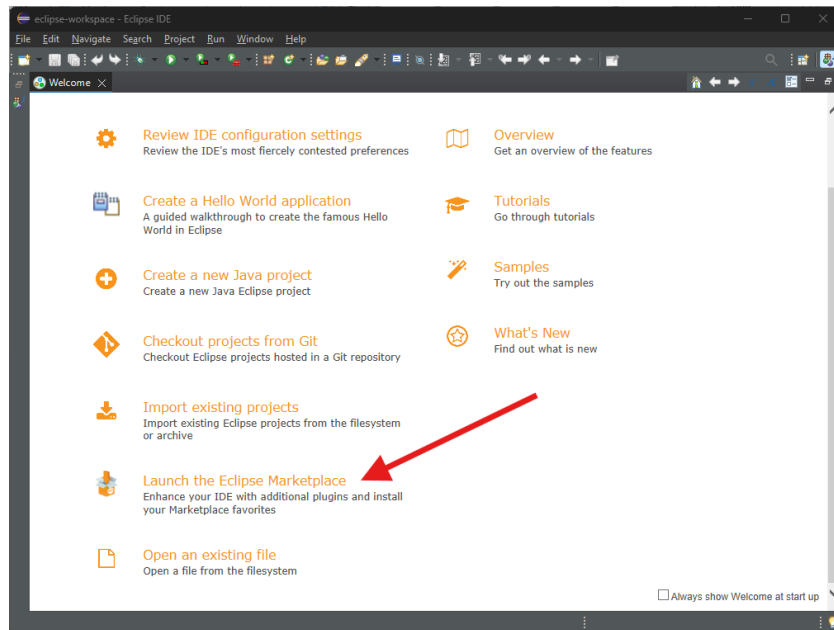
Gambar 1-4 Download Eclipse installer

Setelah selesai mengunduh *installer*, maka lanjutkan instalasi dengan membuka file **exe** hingga menunjukkan Gambar 1-5. Pilih menu “Eclipse IDE for Java Developers” dan ikuti proses Instalasinya.



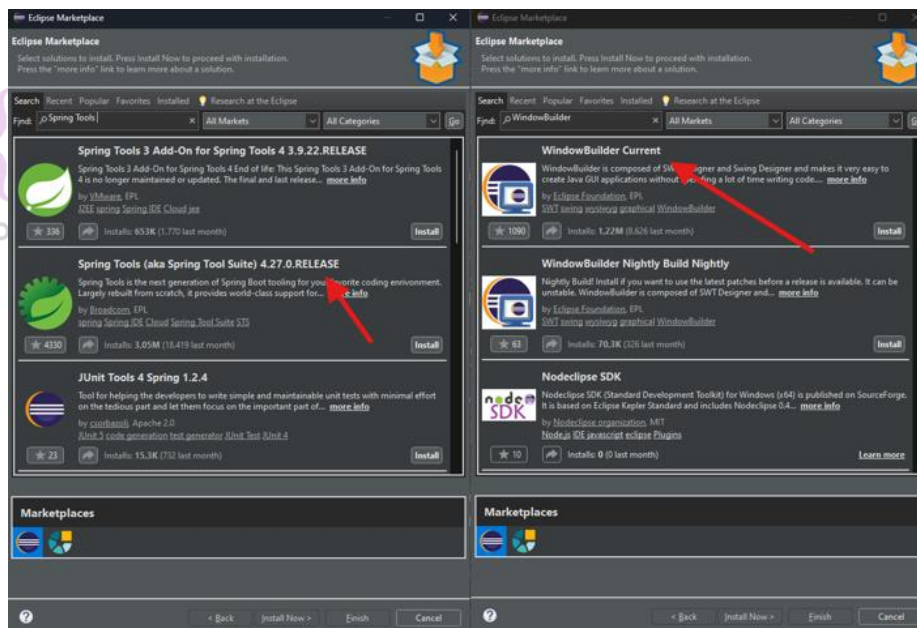
Gambar 1-5 Eclipse installer.

Silahkan buka aplikasi Eclipse yang sudah selesai diinstal. Sebelum membuat proyek, kita akan memerlukan beberapa ekstensi untuk keperluan modul-modul selanjutnya. Modul tersebut yaitu untuk Java GUI dan Java Framework Spring. Hal ini dapat dilakukan dengan menekan menu “Launch the Eclipse Marketplace” seperti digambar 1-6.



Gambar 1-6 Eclipse Marketplace

Lakukan pencarian dengan kata “WindowBuilder” dan “Spring Tools” seperti Gambar 1-7. Lanjutkan dengan melakukan instalasi pada dua ekstensi tersebut.



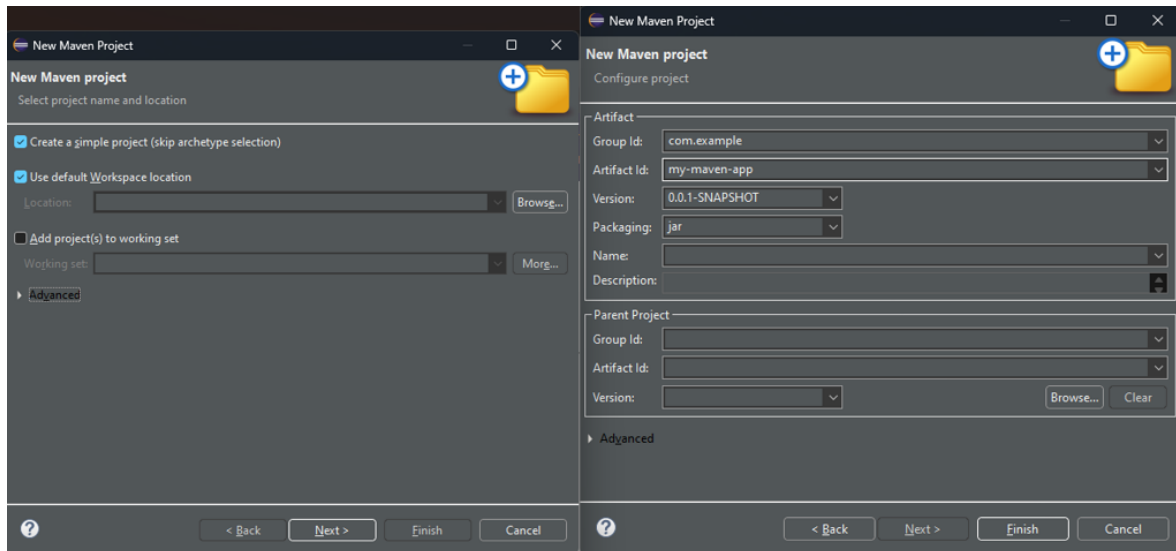
Gambar 1-7 Ekstensi Spring Tools dan WindowBuilder

Di dalam eclipse, semua perancangan dan pemrograman dilakukan di dalam kerangka sebuah proyek. Proyek eclipse merupakan sekumpulan *file* yang dikelompokkan di dalam suatu kesatuan.

1) Membuat proyek Java Maven

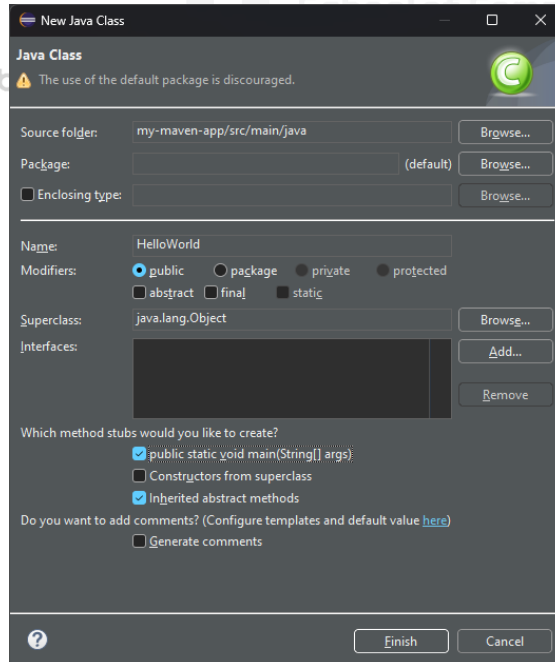
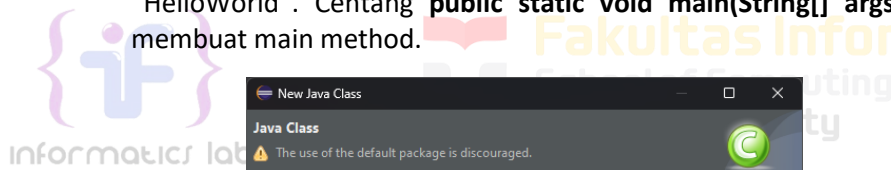
Sebelum membuat program Java dengan menggunakan eclipse, langkah pertama adalah membuat proyek terlebih dahulu.

- a. Jalankan **menu File → New → Maven Project** untuk membuka suatu dialog **New Java Project**.



Gambar 1-8 Project baru

- b. Dalam dialog, silahkan sesuaikan dengan gambar 1-7 sebagai bantuan Anda dalam membuat project baru.
- c. Kemudian klik tombol “Finish” untuk membuat proyek Java.
- d. Setelah project terbuka, buat class java baru dengan cara **Klik kanan** pada folder `src/main/java` → **New** → **Class**.
- e. Beri nama class sesuai dengan yang diinginkan. Dalam contoh, maka class bernama “HelloWorld”. Centang **public static void main(String[] args)** untuk otomatis membuat main method.



Gambar 1-9 Membuat Class baru

- f. Setelah selesai, maka File HelloWorld.java akan terbuat.

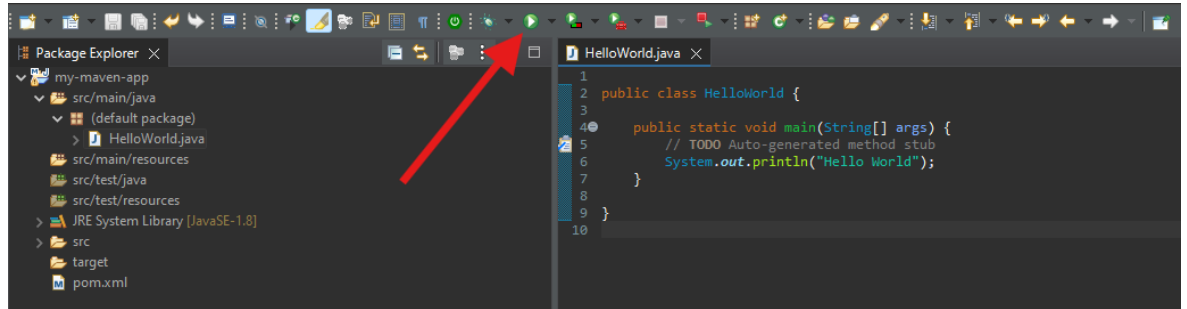
2) Memulai membuat program sederhana di Java

Setelah membuat proyek di Netbeans, selanjutnya membuat program sederhana. Di bawah ini merupakan contoh program yang akan menampilkan “Hello World”.

```
public class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("Hello World");  
    }  
}
```

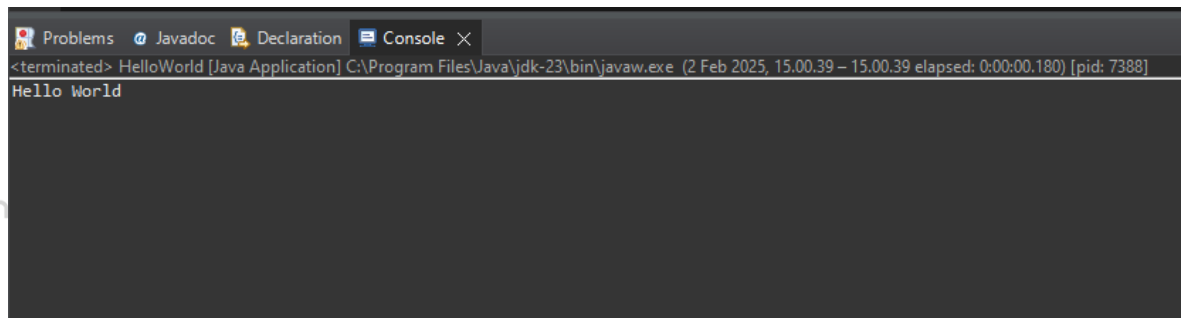
3) Meng-*compile* dan menjalankan program Java di Netbeans

Untuk meng-*compile* dan menjalankan program, maka klik tombol panah hijau, seperti gambar dibawah ini:



Gambar 1-10 Tampilan Run Main Project.

Jika programnya sukses maka akan tampil tulisan *hello word* seperti di bawah ini :



Gambar 1-11 Tampilan hasil *compile* dan *run* program *hello world*.

Modul 2 Graphical User Interface (GUI) Part 1

Tujuan Praktikum

1. Menenal komponen-komponen dasar dalam pembuatan *user interface* dalam Java.
2. Mengerti konsep *Graphic User Interface* dalam Java dan dapat mengimplementasikannya dalam sebuah program sederhana.

2.1 AWT dan Swing

AWT (Abstract Windowing Toolkit) dan Swing pada dasarnya adalah dua teknologi yang digunakan untuk membangun GUI (Graphical User Interface) dalam bahasa pemrograman Java. AWT berada dalam package `java.awt`, yang menyediakan komponen-komponen GUI yang bersifat platform-dependent, yaitu tergantung pada sistem operasi yang digunakan. Sebaliknya, Swing berada dalam package `javax.swing` dan lebih bersifat *lightweight*, artinya komponen-komponen ini tidak bergantung pada platform tertentu dan dapat diaplikasikan secara lintas platform (*multiplatform*).

Dalam perkembangannya, Swing lebih mendominasi dibandingkan AWT, karena menawarkan lebih banyak komponen dan kontrol yang lebih fleksibel dalam pembangunan antarmuka pengguna. Perbedaan utamanya adalah bahwa AWT mengandalkan komponen native (komponen dari sistem operasi yang digunakan), sementara Swing menggunakan model komponen yang digambar secara langsung oleh Java, memungkinkan tampilan yang konsisten di berbagai platform.

Beberapa fasilitas yang disediakan oleh kedua toolkit tersebut antara lain:

1. **Pengaturan tata letak (*layout management*):** Mengatur bagaimana komponen-komponen GUI ditempatkan di dalam sebuah container.
2. **Event handling:** Menyediakan mekanisme untuk mendeteksi aksi atau peristiwa yang dilakukan oleh pengguna (seperti klik, penekanan tombol, dll.) dan menentukan respons yang sesuai dari aplikasi.
3. **Manipulasi grafis komponen:** Memungkinkan pengaturan grafis komponen seperti font, warna, ikon, dan elemen visual lainnya.

2.2 Komponen utama dalam GUI

1. **Containers:** tempat dimana *componen* lain bisa di tempatkan didalamnya, contohnya adalah panel.
2. **Canvases:** Digunakan untuk menampilkan *image* atau pembuatan program yang berbasis grafik. Kita bisa menggambar titik, lingkaran dan sebagainya.
3. **UI components:** contohnya *buttons*, *lists*, *simple pop-up menus*, *check boxes*, *text fields*, dan elemen khusus lain untuk *user interface*.
4. **Window construction components:** Contohnya *windows*, *frames*, *menu bars*, dan *dialog boxes*

2.3 The Basic User Interface Components with Swing

Secara umum, terdapat 3 jenis komponen dasar dalam **Swing** yang sering digunakan, yang dapat dibagi ke dalam beberapa kategori sebagai berikut:

1. **Top-Level Container**

Container utama yang digunakan untuk menampung dan menampilkan komponen lainnya. Komponen ini biasanya menjadi dasar dari antarmuka pengguna dalam aplikasi.

Contoh: `JFrame`, `JDialog`, dan `JApplet`. Komponen-komponen ini bertindak sebagai container utama untuk komponen lain yang ada di dalam aplikasi.

2. **Canvases**

Container perantara yang digunakan untuk mengelompokkan komponen lain di dalamnya. Container ini sering kali digunakan untuk mengorganisasi layout dari komponen antarmuka pengguna.

Contoh: `JPanel` (digunakan untuk meletakkan dan mengelompokkan komponen lainnya), serta container lain seperti `JScrollPane` dan `JTabbedPane` yang sering digunakan untuk memberikan fungsi scroll atau tab pada aplikasi.

3. **Atomic Component**

Komponen-komponen dasar yang memiliki fungsi spesifik dan biasanya berinteraksi langsung dengan pengguna. Komponen ini adalah elemen yang terlihat di antarmuka dan memungkinkan pengguna untuk berinteraksi dengan aplikasi.

Contoh: `JButton` (tombol), `JLabel` (label teks), `JTextField` (input teks), `JTextArea` (area teks), dan komponen lainnya yang digunakan untuk input dan output data.

2.4 Top Level Container

2.4.1 JFrame

`JFrame` adalah salah satu Top-Level Container dalam Swing yang digunakan sebagai **window utama** dalam aplikasi GUI. `JFrame` berfungsi sebagai wadah yang menampung berbagai komponen GUI seperti tombol (`JButton`), teks (`JTextField`), label (`JLabel`), dan lainnya.

Terdapat 2 cara untuk membuat `JFrame`:

1. **Pembuatan JFrame langsung**

Buatlah file class java baru dengan main method. Kemudian coba code sebagai berikut:

```
import javax.swing.*;

public class JFrameExample {
    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            JFrame frame = new JFrame("Contoh JFrame");
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.setSize(400, 200);
            frame.setVisible(true);
        });
    }
}
```

Penjelasan Code:

1. `JFrame frame = new JFrame("Contoh JFrame");` → Membuat frame dengan judul.
2. `setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);` → Menutup aplikasi saat window ditutup.
3. `setSize(400, 200);` → Mengatur ukuran frame.
4. `setVisible(true);` → Menampilkan frame di layar.

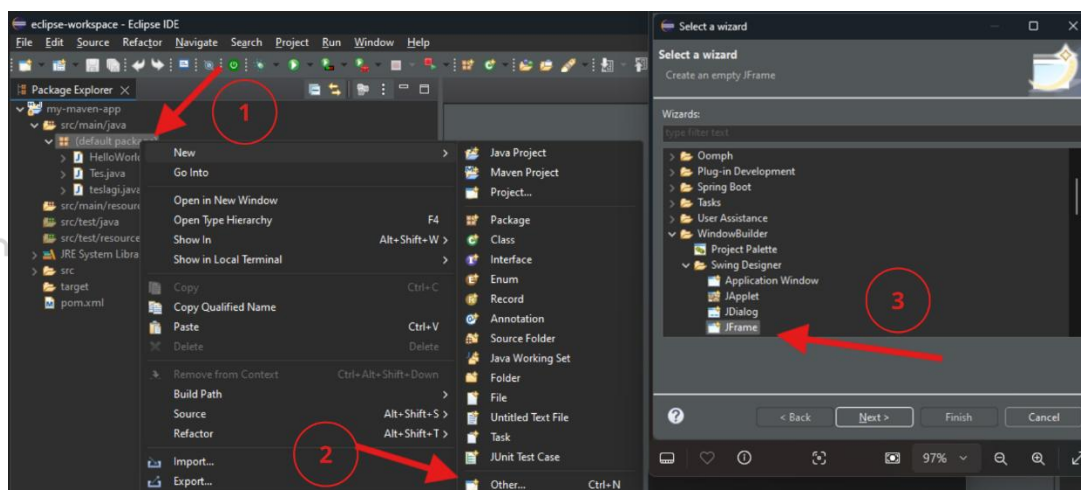
JFrame memiliki empat mode penutupan yang dapat digunakan:

Tabel 2-1

Mode	Keterangan
DO_NOTHING_ON_CLOSE	Tidak melakukan apa pun saat frame ditutup. Harus ditangani secara manual.
HIDE_ON_CLOSE (Default)	Menyembunyikan frame tetapi tetap ada di memori.
DISPOSE_ON_CLOSE	Menghapus tampilan frame dan membebaskan resource-nya.
EXIT_ON_CLOSE	Menghentikan program saat frame utama ditutup.

2. Pembuatan JFrame dengan Eclipse (Generate File)

Selain dengan kode langsung, Eclipse menyediakan cara otomatis untuk membuat JFrame. Hal ini dapat dilakukan dengan **memilih package → Klik kanan → New → Other.. → Window Builder → Swing Designer → JFrame**. Untuk cara lebih jelas, Anda dapat melihat gambar berikut:



Gambar 2-1 Membuat JFrame Generate File

2.4.2 JDialog

JDialog adalah Top-Level Container yang digunakan untuk menampilkan **popup atau kotak dialog** dalam aplikasi GUI Java. Tidak seperti JFrame, JDialog bersifat **modal**, yang berarti akan memblokir interaksi dengan jendela utama (JFrame) sampai dialog tersebut ditutup.

Terdapat 2 cara untuk membuat JFrame:

1. Pembuatan JFrame langsung

Buatlah file class java baru dengan main method. Kemudian coba code sebagai berikut:

```
import javax.swing.*;

public class JDialogExample {
    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            JFrame frame = new JFrame("Frame Utama");
        });
    }
}
```

```

        frame.setSize(400, 300);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);

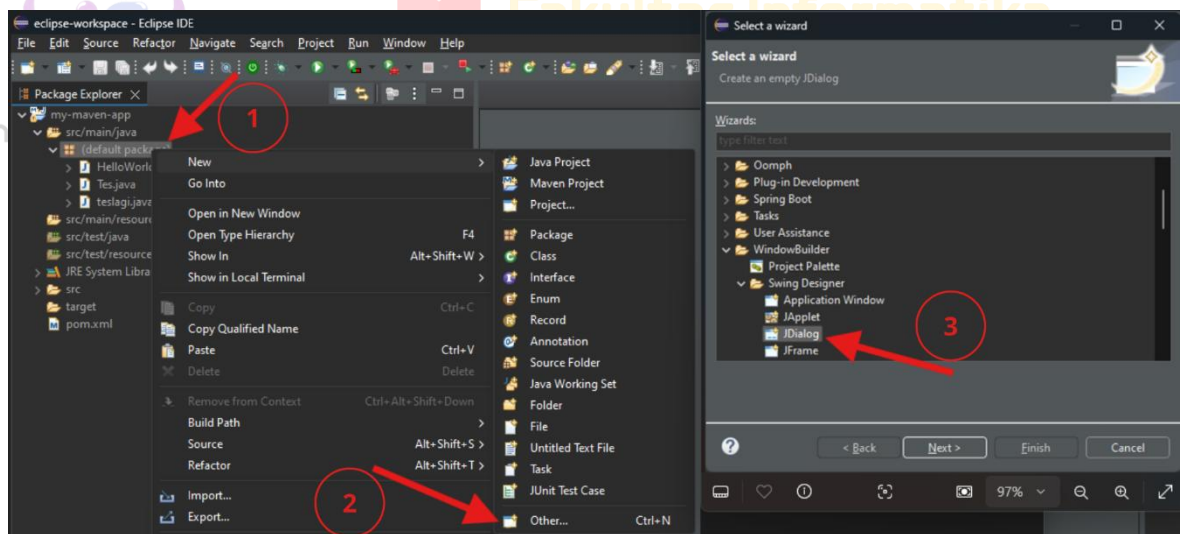
        // Membuat dialog
        JDialog dialog = new JDialog(frame, "Contoh Dialog", true);
        dialog.setSize(250, 150);
        dialog.setLocationRelativeTo(frame);
        dialog.setVisible(true);
    });
}
}

```

Penjelasan Code:

1. JDialog dialog = new JDialog(frame, "Contoh Dialog", true);
 - frame → JFrame sebagai **parent**.
 - "Contoh Dialog" → Judul dialog.
 - true → Dialog bersifat **modal** (memblokir input ke JFrame).
 2. dialog.setSize(250, 150); → Mengatur ukuran dialog.
 3. dialog.setVisible(true); → Menampilkan dialog di atas JFrame.
2. **Pembuatan JDialog dengan Eclipse (Generate File)**

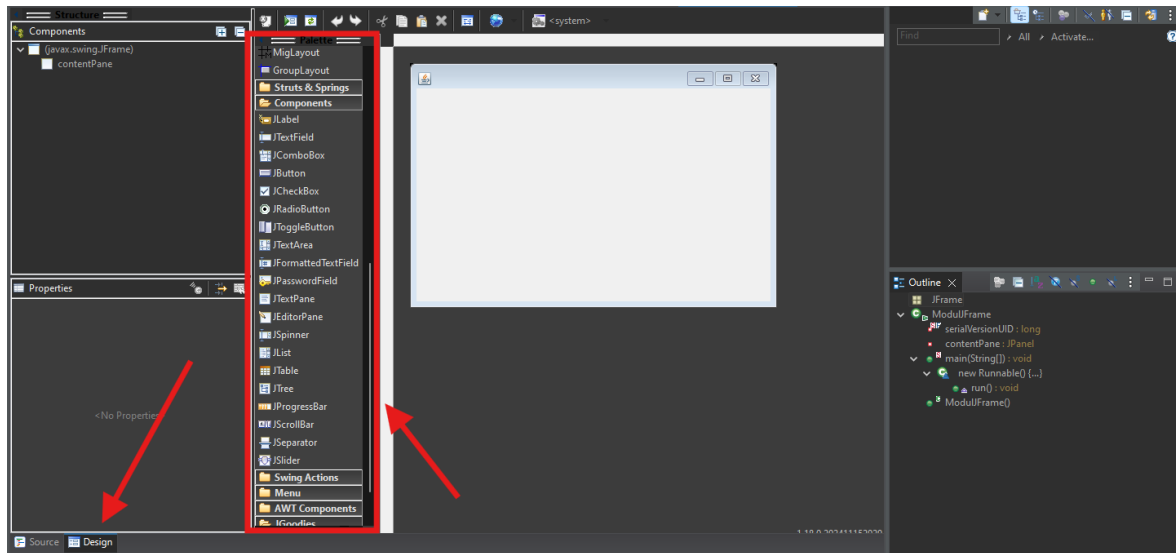
Selain dengan kode langsung, Eclipse menyediakan cara otomatis untuk membuat JDialog. Hal ini dapat dilakukan dengan **memilih package → Klik kanan → New → Other.. → Window Builder → Swing Designer → JFrame**. Untuk cara lebih jelas, Anda dapat melihat gambar berikut:



Gambar 2-2 Membuat JDialog Generate File

2.5 GUI Drag and Drop

Ketika Anda telah berhasil membuat JFrame dengan cara Generate File (menggunakan WindowBuilder), maka Anda akan dapat menggunakan fitur **Drag and Drop** untuk memudahkan Anda dalam membuat tampilan. Anda dapat memilih menu “Design”, seperti di Gambar 2-3. Maka akan muncul tampilan GUI beserta komponen-komponen yang bisa Anda pilih dan masukkan ke tampilan utama.



Gambar 2-3 Komponen GUI Builder

2.6 Pengaturan Layout dalam Swing

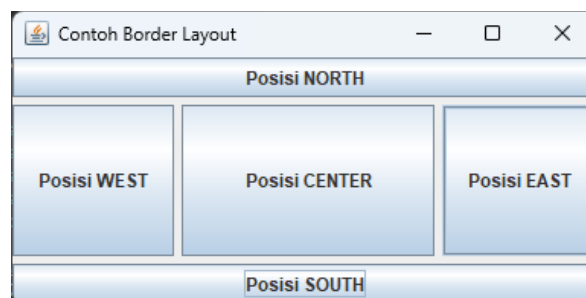
Layout Manager adalah mekanisme yang digunakan dalam Java Swing untuk mengatur **penempatan dan ukuran** komponen dalam container (JFrame, JPanel, dll.). Java Swing menyediakan beberapa layout manager yang umum digunakan untuk mengatur tampilan GUI.

Berikut adalah beberapa jenis **Layout Manager** yang akan dibahas:

1. **BorderLayout** → Menempatkan komponen berdasarkan arah mata angin.
2. **BoxLayout** → Menyusun komponen dalam satu baris atau satu kolom.
3. **CardLayout** → Menyusun komponen seperti tumpukan kartu, hanya satu yang terlihat dalam satu waktu.
4. **FlowLayout** → Menyusun komponen dalam satu baris dari kiri ke kanan.
5. **GridLayout** → Menyusun komponen dalam bentuk grid (tabel).

2.6.1 BorderLayout

BorderLayout adalah layout yang menempatkan komponen berdasarkan arah mata angin seperti **NORTH, SOUTH, EAST, WEST, dan CENTER**. Jika tidak ditentukan posisinya, maka komponen akan ditempatkan di **CENTER** secara default. Berikut adalah contoh tampilan-nya:



Gambar 2-4 Border Layout

Source Code-nya:

```
import javax.swing.*;
```

```
import java.awt.*;

public class BorderLayoutDemo {
    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            JFrame frame = new JFrame("Contoh Border Layout");
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.setSize(400, 200);

            frame.setLayout(new BorderLayout(5, 5)); // Mengatur jarak antar
            komponen

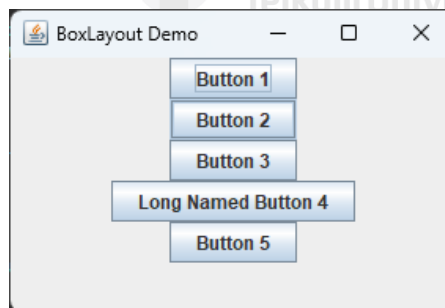
            // Membuat tombol di setiap posisi
            JButton btnNorth = new JButton("Posisi NORTH");
            JButton btnSouth = new JButton("Posisi SOUTH");
            JButton btnEast = new JButton("Posisi EAST");
            JButton btnWest = new JButton("Posisi WEST");
            JButton btnCenter = new JButton("Posisi CENTER");

            // Menambahkan tombol ke frame sesuai posisi
            frame.add(btnNorth, BorderLayout.NORTH);
            frame.add(btnSouth, BorderLayout.SOUTH);
            frame.add(btnEast, BorderLayout.EAST);
            frame.add(btnWest, BorderLayout.WEST);
            frame.add(btnCenter, BorderLayout.CENTER);

            frame.setVisible(true);
        });
    }
}
```

2.6.2 BoxLayout

BoxLayout menyusun komponen dalam satu baris (horizontal) atau satu kolom (vertikal). Berikut adalah contoh tampilan-nya:



Gambar 2-5 Box Layout

Source Code-nya:

```
import javax.swing.*;
import java.awt.*;

public class BoxLayoutDemo {
    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            JFrame frame = new JFrame("BoxLayout Demo");
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.setSize(300, 200);

            JPanel panel = new JPanel();
            panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS)); // Layout
            vertikal

            JButton btn1 = new JButton("Button 1");
            JButton btn2 = new JButton("Button 2");
```

```

        JButton btn3 = new JButton("Button 3");
        JButton btn4 = new JButton("Long Named Button 4");
        JButton btn5 = new JButton("Button 5");

        btn1.setAlignmentX(Component.CENTER_ALIGNMENT);
        btn2.setAlignmentX(Component.CENTER_ALIGNMENT);
        btn3.setAlignmentX(Component.CENTER_ALIGNMENT);
        btn4.setAlignmentX(Component.CENTER_ALIGNMENT);
        btn5.setAlignmentX(Component.CENTER_ALIGNMENT);

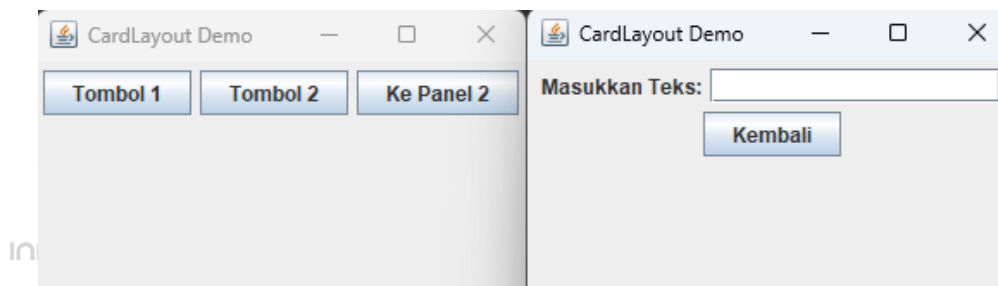
        panel.add(btn1);
        panel.add(btn2);
        panel.add(btn3);
        panel.add(btn4);
        panel.add(btn5);

        frame.add(panel);
        frame.setVisible(true);
    });
}
}

```

2.6.3 CardLayout

CardLayout digunakan untuk menampilkan komponen seperti **tumpukan kartu**, sehingga hanya **satu komponen** yang terlihat pada satu waktu. Berikut adalah contoh tampilan-nya:



Gambar 2-6 Card Layout

Source Code-nya:

```

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class CardLayoutDemo {
    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            // Membuat frame utama
            JFrame frame = new JFrame("CardLayout Demo");
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.setSize(300, 200);

            // Membuat panel utama dengan CardLayout
            JPanel cardPanel = new JPanel();
            CardLayout cardLayout = new CardLayout();
            cardPanel.setLayout(cardLayout);

            // Panel pertama dengan tombol
            JPanel panel1 = new JPanel();
            JButton btnNext = new JButton("Ke Panel 2");
            panel1.add(new JButton("Tombol 1"));
            panel1.add(new JButton("Tombol 2"));
            panel1.add(btnNext);
        });
    }
}

```



```

        // Panel kedua dengan TextField
        JPanel panel2 = new JPanel();
        JTextField textField = new JTextField(15);
        JButton btnBack = new JButton("Kembali");
        panel2.add(new JLabel("Masukkan Teks:"));
        panel2.add(textField);
        panel2.add(btnBack);

        // Menambahkan panel ke CardLayout
        cardPanel.add(panel1, "Panel 1");
        cardPanel.add(panel2, "Panel 2");

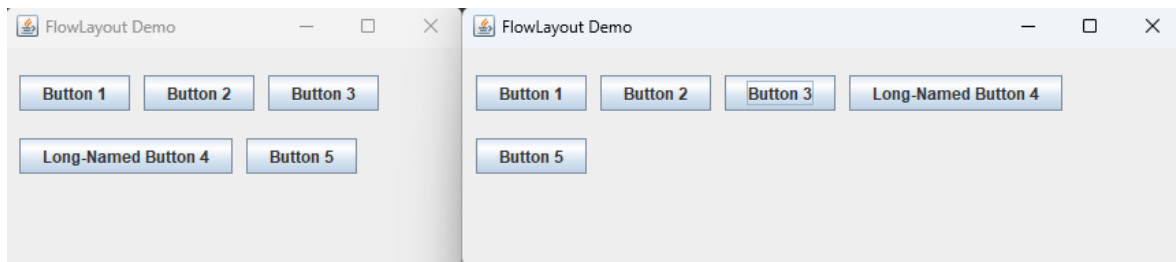
        // Menambahkan aksi untuk berpindah panel
        btnNext.addActionListener(e -> cardLayout.show(cardPanel, "Panel 2"));
        btnBack.addActionListener(e -> cardLayout.show(cardPanel, "Panel 1"));

        // Menambahkan CardLayout ke frame
        frame.add(cardPanel);
        frame.setVisible(true);
    });
}
}

```

2.6.4 FlowLayout

FlowLayout menyusun komponen dari kiri ke kanan dan dari atas ke bawah secara otomatis, mirip seperti cara teks ditulis dalam dokumen. Berikut adalah contoh tampilan-nya:



Gambar 2-7 Flow Layout

Source Code-nya:

```

import javax.swing.*;
import java.awt.*;

public class FlowLayoutDemo {
    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            // Membuat frame utama
            JFrame frame = new JFrame("FlowLayout Demo");
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.setSize(350, 200);

            // Mengatur layout menjadi FlowLayout dengan rata kiri dan jarak antar
            // komponen
            FlowLayout layout = new FlowLayout(FlowLayout.LEFT, 10, 20);
            JPanel panel = new JPanel();
            panel.setLayout(layout);

            // Menambahkan tombol ke dalam panel
            JButton btn1 = new JButton("Button 1");
            JButton btn2 = new JButton("Button 2");
            JButton btn3 = new JButton("Button 3");
            JButton btn4 = new JButton("Long-Named Button 4");
            JButton btn5 = new JButton("Button 5");

            panel.add(btn1);

```

```

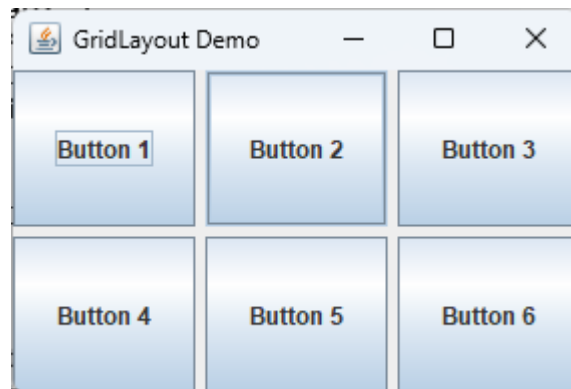
        panel.add(btn2);
        panel.add(btn3);
        panel.add(btn4);
        panel.add(btn5);

        // Menambahkan panel ke dalam frame
        frame.add(panel);
        frame.setVisible(true);
    });
}

```

2.6.5 GridLayout

GridLayout menyusun komponen dalam bentuk tabel, di mana setiap sel memiliki ukuran yang sama besar. Berikut adalah contoh tampilan-nya:



Gambar 2-8 Grid Layout

Source Code-nya:

```

import javax.swing.*;
import java.awt.*;

public class GridLayoutDemo {
    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            JFrame frame = new JFrame("GridLayout Demo");
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.setSize(300, 200);

            JPanel panel = new JPanel();
            panel.setLayout(new GridLayout(2, 3, 5, 5)); // 2 baris, 3 kolom

            JButton btn1 = new JButton("Button 1");
            JButton btn2 = new JButton("Button 2");
            JButton btn3 = new JButton("Button 3");
            JButton btn4 = new JButton("Button 4");
            JButton btn5 = new JButton("Button 5");
            JButton btn6 = new JButton("Button 6");

            panel.add(btn1);
            panel.add(btn2);
            panel.add(btn3);
            panel.add(btn4);
            panel.add(btn5);
            panel.add(btn6);

            frame.add(panel);
            frame.setVisible(true);
        });
    }
}

```

2.7 Event Handling

Untuk mendeteksi atas apa yang dilakukan oleh user, diperlukan **penanganan khusus** terhadap **event** (peristiwa yang di-stimulasikan/di-trigger) yang diperlukan oleh *user* terhadap **komponen GUI** tertentu. Penanganan ini disebut **Event Handling Component**. *Event Handling Component* dibagi atas 2 bagian, yaitu **Event Listener** dan **Event Handler**. Untuk mengenal 2 *method* Event tersebut dapat diilustrasikan sbb:

Bila mengklik suatu *Object button*, maka tercipta *event*. *Event* ini ditangkap oleh *Event Listener* dan secara otomatis akan diberi ID (identitas) seperti `button1`, `button2`, `textfield1`, dll atau Kita dapat memberikan ID atas komponen tersebut dengan memberi nama suatu *Object*. Sehingga Java mengenal *event* mana yang ditangkap.

Selanjutnya, tentukan **Event Handler** dari komponen tersebut, berupa **blok program/statement** yang **memproses** bila terjadi suatu *event* yang ditangkap oleh **Event Listener**.

Berikut adalah beberapa **Event Listener** yang sering digunakan dalam Java Swing:

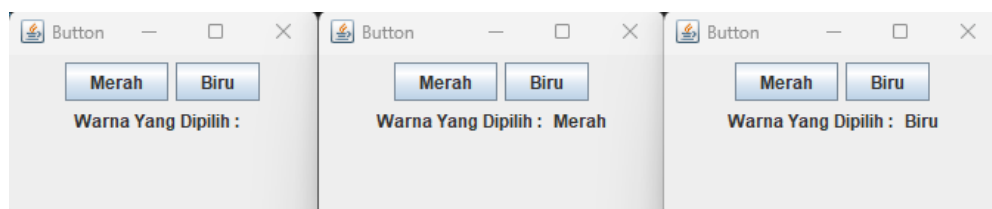
Event Class	Listener Interface	Deskripsi
ActionEvent	ActionListener	Digunakan untuk menangani event seperti menekan tombol (JButton), memilih item di JComboBox, atau menekan Enter di JTextField.
MouseEvent	MouseListener / MouseMotionListener	Digunakan untuk menangani event yang berkaitan dengan mouse, seperti klik, hover, atau geser.
KeyEvent	KeyListener	Digunakan untuk menangani event ketika pengguna menekan tombol pada keyboard.
ItemEvent	ItemListener	Digunakan untuk menangani perubahan status pada komponen seperti JCheckBox dan JComboBox.
WindowEvent	WindowListener	Digunakan untuk menangani event yang berhubungan dengan jendela (JFrame), seperti saat ditutup atau dibuka.

Berikut merupakan contoh program dari *event handling* yang menggunakan *JTable*, *JButton*, *JRadioButton*, *JCheckBox*, dan *JComboBox*.

2.7.1 Button (Jbutton)

JButton digunakan untuk menangkap aksi tombol yang diklik oleh pengguna. Untuk menangani event dari tombol, kita menggunakan ActionListener.

Dalam contoh berikut, kita akan membuat dua tombol: "Merah" dan "Biru". Saat pengguna menekan tombol "Merah", teks label akan berubah menjadi "Merah", dan jika tombol "Biru" ditekan, teks label akan berubah menjadi "Biru".



Gambar 2-9 Button

Source Code-nya:

```
import javax.swing.*;
import java.awt.*;
```

```

public class ButtonEventDemo {
    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            JFrame frame = new JFrame("Button");
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.setSize(300, 150);
            frame.setLayout(new FlowLayout(FlowLayout.Center));

            JLabel label = new JLabel("Warna Yang Dipilih : ");
            JLabel labelWarna = new JLabel("");

            JButton btnMerah = new JButton("Merah");
            JButton btnBiru = new JButton("Biru");

            btnMerah.addActionListener(e -> labelWarna.setText("Merah"));
            btnBiru.addActionListener(e -> labelWarna.setText("Biru"));

            frame.add(btnMerah);
            frame.add(btnBiru);
            frame.add(label);
            frame.add(labelWarna);

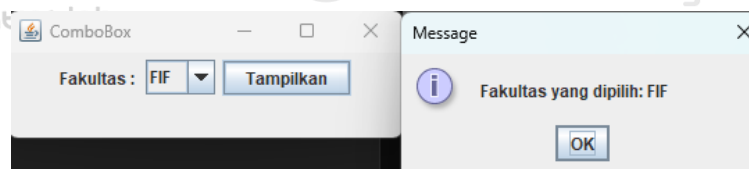
            frame.setVisible(true);
        });
    }
}

```

2.7.2 ComboBox (JComboBox)

JComboBox digunakan untuk membuat daftar pilihan (dropdown menu). Untuk menangani event saat pengguna memilih item, kita menggunakan ActionListener.

Dalam contoh berikut, kita akan membuat ComboBox berisi daftar fakultas. Jika pengguna memilih salah satu fakultas, teks yang dipilih akan ditampilkan dalam JOptionPane.



Gambar 2-10 ComboBox

Source Code-nya:

```

import javax.swing.*;
import java.awt.*;

public class ComboBoxEventDemo {
    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            JFrame frame = new JFrame("ComboBox");
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.setSize(300, 100);
            frame.setLayout(new FlowLayout());

            JLabel label = new JLabel("Fakultas : ");
            String[] fakultas = {"FIF", "FKB", "FIT", "FEB", "FTE", "FRI", "FIK"};
            JComboBox<String> comboBox = new JComboBox<>(fakultas);
            JButton btnTampilkan = new JButton("Tampilkan");

            btnTampilkan.addActionListener(e -> {
                JOptionPane.showMessageDialog(frame, "Fakultas yang dipilih: " +
                    comboBox.getSelectedItem());
            });
        });
    }
}

```

```

        frame.add(label);
        frame.add(comboBox);
        frame.add(btnTampilkan);

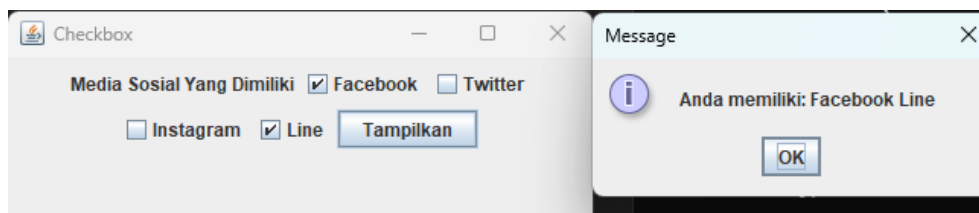
        frame.setVisible(true);
    });
}
}

```

2.7.3 CheckBox (JCheckBox)

JCheckBox digunakan untuk membuat pilihan yang dapat dipilih lebih dari satu. Untuk menangani event, kita menggunakan ItemListener.

Dalam contoh berikut, kita akan membuat CheckBox untuk memilih platform media sosial. Saat pengguna mencentang atau menghapus centang dari CheckBox, daftar media sosial yang dipilih akan ditampilkan dalam JOptionPane.



Gambar 2-11 CheckBox

Source Code-nya:

```

import javax.swing.*;
import java.awt.*;

public class CheckBoxEventDemo {
    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            JFrame frame = new JFrame("Checkbox");
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.setSize(400, 150);
            frame.setLayout(new FlowLayout());

            JLabel label = new JLabel("Media Sosial Yang Dimiliki");
            JCheckBox cbFacebook = new JCheckBox("Facebook");
            JCheckBox cbTwitter = new JCheckBox("Twitter");
            JCheckBox cbInstagram = new JCheckBox("Instagram");
            JCheckBox cbLine = new JCheckBox("Line");

            JButton btnTampilkan = new JButton("Tampilkan");
            btnTampilkan.addActionListener(e -> {
                StringBuilder hasil = new StringBuilder("Anda memiliki: ");
                if (cbFacebook.isSelected()) hasil.append("Facebook ");
                if (cbTwitter.isSelected()) hasil.append("Twitter ");
                if (cbInstagram.isSelected()) hasil.append("Instagram ");
                if (cbLine.isSelected()) hasil.append("Line ");
                JOptionPane.showMessageDialog(frame, hasil.toString());
            });

            frame.add(label);
            frame.add(cbFacebook);
            frame.add(cbTwitter);
            frame.add(cbInstagram);
            frame.add(cbLine);
            frame.add(btnTampilkan);

            frame.setVisible(true);
        });
    }
}

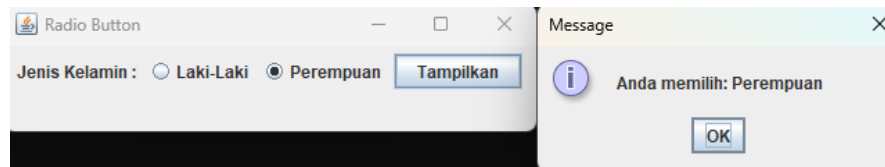
```

```
}  
}
```

2.7.4 RadioButton (JRadioButton)

JRadioButton digunakan untuk membuat pilihan satu dari banyak. Jika beberapa JRadioButton dikelompokkan dalam ButtonGroup, hanya satu yang dapat dipilih dalam satu waktu.

Dalam contoh berikut, kita akan membuat RadioButton untuk memilih jenis kelamin. Jika pengguna memilih "Laki-Laki" atau "Perempuan", pilihan tersebut akan ditampilkan dalam JOptionPane.



Gambar 2-12 RadioButton

Source Code-nya:

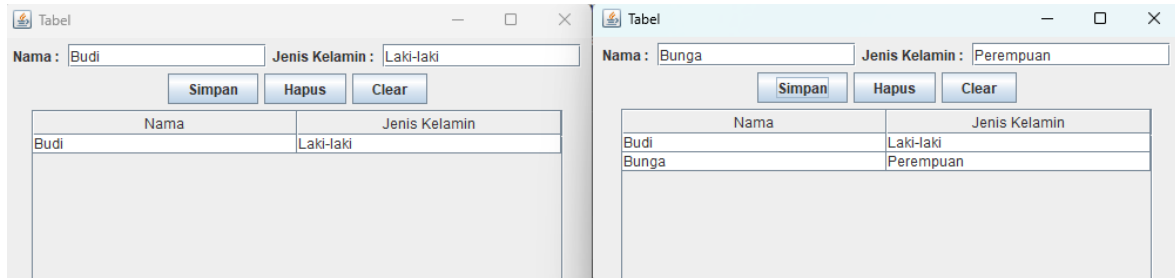
```
import javax.swing.*;  
import java.awt.*;  
  
public class RadioButtonEventDemo {  
    public static void main(String[] args) {  
        SwingUtilities.invokeLater(() -> {  
            JFrame frame = new JFrame("Radio Button");  
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
            frame.setSize(400, 100);  
            frame.setLayout(new FlowLayout());  
  
            JLabel label = new JLabel("Jenis Kelamin :");  
            JRadioButton rbLaki = new JRadioButton("Laki-Laki");  
            JRadioButton rbPerempuan = new JRadioButton("Perempuan");  
  
            ButtonGroup group = new ButtonGroup();  
            group.add(rbLaki);  
            group.add(rbPerempuan);  
  
            JButton btnTampilkan = new JButton("Tampilkan");  
            btnTampilkan.addActionListener(e -> {  
                String pilihan = rbLaki.isSelected() ? "Laki-Laki" :  
rbPerempuan.isSelected() ? "Perempuan" : "Tidak ada yang dipilih";  
                JOptionPane.showMessageDialog(frame, "Anda memilih: " + pilihan);  
            });  
  
            frame.add(label);  
            frame.add(rbLaki);  
            frame.add(rbPerempuan);  
            frame.add(btnTampilkan);  
  
            frame.setVisible(true);  
        });  
    }  
}
```

2.7.5 Table (JTable)

JTable digunakan untuk menampilkan dan mengelola data dalam bentuk tabel. Event handling pada JTable digunakan untuk menambah, menghapus, atau memperbarui data di dalam tabel.

Dalam contoh berikut, kita akan membuat tabel dinamis yang berisi daftar nama dan jenis kelamin. Pengguna dapat:

- Menambahkan data baru ke dalam tabel.
- Menghapus baris yang dipilih.
- Menghapus semua data dalam tabel.



Gambar 2-13 Table

Source Code-nya:

```
import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.*;

public class TableEventDemo {
    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            JFrame frame = new JFrame("Tabel");
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.setSize(400, 250);
            frame.setLayout(new FlowLayout());

            JLabel labelNama = new JLabel("Nama : ");
            JTextField textNama = new JTextField(15);

            JLabel labelGender = new JLabel("Jenis Kelamin : ");
            JTextField textGender = new JTextField(15);

            DefaultTableModel model = new DefaultTableModel(new String[]{"Nama",
"Jenis Kelamin"}, 0);
            JTable table = new JTable(model);
            JScrollPane scrollPane = new JScrollPane(table);

            JButton btnSimpan = new JButton("Simpan");
            JButton btnHapus = new JButton("Hapus");
            JButton btnClear = new JButton("Clear");

            btnSimpan.addActionListener(e -> model.addRow(new
Object[]{textNama.getText(), textGender.getText()}));
            btnHapus.addActionListener(e ->
model.removeRow(table.getSelectedRow()));
            btnClear.addActionListener(e -> model.setRowCount(0));

            frame.add(labelNama);
            frame.add(textNama);
            frame.add(labelGender);
            frame.add(textGender);
            frame.add(btnSimpan);
            frame.add(btnHapus);
            frame.add(btnClear);
            frame.add(scrollPane);

            frame.setVisible(true);
        });
    }
}
```



Modul 3 Graphical User Interface (GUI) Part 2 & MVC

Tujuan Praktikum
1. Mengetahui konsep Model-View-Controller (MVC) dalam pengembangan aplikasi.
2. Memisahkan Model, View, dan Controller dalam aplikasi berbasis GUI.
3. Mengembangkan aplikasi berbasis MVC dalam Java.

3.1 Pendahuluan

Model-View-Controller (MVC) adalah pola desain perangkat lunak yang memisahkan aplikasi menjadi tiga komponen utama: **Model**, **View**, dan **Controller**. Pemisahan ini bertujuan untuk **memisahkan logika bisnis dari antarmuka pengguna**, sehingga memudahkan pengembangan, pemeliharaan, dan pengembangan aplikasi.

Komponen MVC dalam Java yaitu sebagai berikut:

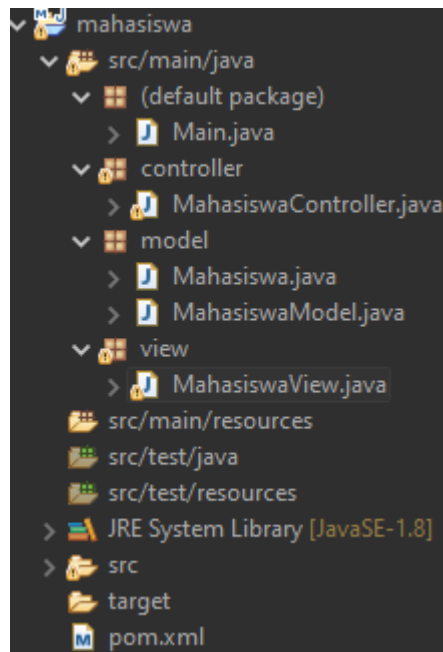
1. **Model** → Mewakili data dan logika bisnis aplikasi. Model bertanggung jawab untuk mengelola data, aturan bisnis, dan logika aplikasi.
2. **View** → Menyediakan representasi visual dari data yang ada di Model. View bertanggung jawab untuk menampilkan data kepada pengguna dan menerima input dari pengguna.
3. **Controller** → Menghubungkan Model dan View. Controller menerima input dari pengguna melalui View, memprosesnya (misalnya, dengan memvalidasi data), dan memperbarui Model atau View sesuai kebutuhan.

Dalam konteks aplikasi Java Swing, penerapan MVC membantu memisahkan logika bisnis dari kode antarmuka pengguna, sehingga meningkatkan modularitas dan kemudahan pemeliharaan. Sebagai contoh, Model dapat berupa kelas yang mewakili data mahasiswa, View dapat berupa antarmuka pengguna berbasis GUI yang menampilkan data mahasiswa, dan Controller dapat berupa kelas yang menangani interaksi antara pengguna dan data mahasiswa.

Dalam penerapan **MVC dalam Java Swing**, sangat penting untuk memiliki **struktur folder yang terorganisir** agar memudahkan pengelolaan kode dan pemisahan tanggung jawab antara **Model**, **View**, dan **Controller**. Dengan struktur yang baik, kita dapat lebih mudah memahami bagaimana data dikelola, ditampilkan, dan dikontrol dalam aplikasi.

Struktur proyek yang rapi juga membantu dalam **pemeliharaan jangka panjang** dan memudahkan tim pengembang untuk bekerja secara kolaboratif tanpa mencampur kode antara **logika bisnis (Model)**, **tampilan (View)**, dan **interaksi pengguna (Controller)**.

Agar implementasi MVC dalam Java lebih jelas dan terstruktur, berikut adalah susunan direktori dan file yang umum digunakan dalam proyek berbasis **MVC dengan Java Swing**.



Gambar 3-1 Susunan Direktori

3.2 Model

Model bertanggung jawab atas data dan logika bisnis. Kita akan membuat dua kelas:

- Mahasiswa.java → Representasi objek mahasiswa.
- MahasiswaModel.java → Bertanggung jawab untuk menyimpan data mahasiswa dalam **ArrayList**.

Source code Mahasiswa.java:

```
package model;

public class Mahasiswa {
    private String nama;
    private String jenisKelamin;

    public Mahasiswa(String nama, String jenisKelamin) {
        this.nama = nama;
        this.jenisKelamin = jenisKelamin;
    }

    public String getNama() {
        return nama;
    }

    public String getJenisKelamin() {
        return jenisKelamin;
    }
}
```

Source code MahasiswaModel.java:

```
package model;

import javax.swing.table.DefaultTableModel;
import java.util.ArrayList;
```

```

import java.util.List;

public class MahasiswaModel {
    private final List<Mahasiswa> mahasiswaList;
    private final DefaultTableModel tableModel;

    public MahasiswaModel() {
        mahasiswaList = new ArrayList<>();
        tableModel = new DefaultTableModel(new String[]{"Nama", "Jenis Kelamin"},
0);
    }

    public void addMahasiswa(String nama, String jenisKelamin) {
        Mahasiswa mahasiswa = new Mahasiswa(nama, jenisKelamin);
        mahasiswaList.add(mahasiswa);
        tableModel.addRow(new Object[]{nama, jenisKelamin});
    }

    public void removeMahasiswa(int index) {
        if (index >= 0 && index < mahasiswaList.size()) {
            mahasiswaList.remove(index);
            tableModel.removeRow(index);
        }
    }

    public void clearMahasiswa() {
        mahasiswaList.clear();
        tableModel.setRowCount(0);
    }

    public DefaultTableModel getTableModel() {
        return tableModel;
    }
}

```

3.3 View

View bertanggung jawab untuk **menampilkan GUI** kepada pengguna dan menerima input. Kita menggunakan **Swing (JFrame)** untuk menampilkan form mahasiswa dengan tabel.

Source code MahasiswaView.java:

```

package view;

import javax.swing.*;
import java.awt.*;

public class MahasiswaView extends JFrame {
    private final JTextField textNama;
    private final JTextField textGender;
    private final JButton btnSimpan;
    private final JButton btnHapus;
    private final JButton btnClear;
    private final JTable table;
    private final JScrollPane scrollPane;

    public MahasiswaView() {
        setTitle("Tabel Mahasiswa");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 250);
        setLayout(new FlowLayout());

        // Label dan Input Field
        JLabel labelNama = new JLabel("Nama : ");
        textNama = new JTextField(15);
        JLabel labelGender = new JLabel("Jenis Kelamin : ");
    }
}

```

```

        textGender = new JTextField(15);

        // Tombol
        btnSimpan = new JButton("Simpan");
        btnHapus = new JButton("Hapus");
        btnClear = new JButton("Clear");

        // Tabel
        table = new JTable();
        scrollPane = new JScrollPane(table);

        // Menambahkan Komponen ke Frame
        add(labelNama);
        add(textNama);
        add(labelGender);
        add(textGender);
        add(btnSimpan);
        add(btnHapus);
        add(btnClear);
        add(scrollPane);

        setVisible(true);
    }

    public JTextField getTextNama() {
        return textNama;
    }

    public JTextField getTextGender() {
        return textGender;
    }

    public JButton getBtnSimpan() {
        return btnSimpan;
    }

    public JButton getBtnHapus() {
        return btnHapus;
    }

    public JButton getBtnClear() {
        return btnClear;
    }

    public JTable getTable() {
        return table;
    }
}

```

3.4 Controller

Controller bertanggung jawab untuk menghubungkan View dan Model serta menangani event dari pengguna.

Source code MahasiswaController.java:

```

package controller;

import model.MahasiswaModel;
import view.MahasiswaView;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class MahasiswaController {
    private final MahasiswaModel model;
    private final MahasiswaView view;
}

```

```

public MahasiswaController(MahasiswaModel model, MahasiswaView view) {
    this.model = model;
    this.view = view;

    // Set tabel di View sesuai dengan model
    view.getTable().setModel(model.getTableModel());

    // Event Listener untuk tombol Simpan
    view.getBtnSimpan().addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            String nama = view.getTextNama().getText();
            String jenisKelamin = view.getTextGender().getText();
            model.addMahasiswa(nama, jenisKelamin);
            view.getTextNama().setText("");
            view.getTextGender().setText("");
        }
    });

    // Event Listener untuk tombol Hapus
    view.getBtnHapus().addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            int selectedRow = view.getTable().getSelectedRow();
            model.removeMahasiswa(selectedRow);
        }
    });

    // Event Listener untuk tombol Clear
    view.getBtnClear().addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            model.clearMahasiswa();
        }
    });
}

```

3.5 Main

Controller bertanggung jawab untuk menghubungkan View dan Model serta menangani event dari pengguna.

Source code Main.java:

```

import model.MahasiswaModel;
import view.MahasiswaView;
import controller.MahasiswaController;

public class Main {
    public static void main(String[] args) {
        MahasiswaModel model = new MahasiswaModel();
        MahasiswaView view = new MahasiswaView();
        new MahasiswaController(model, view);
    }
}

```


Modul 4 GUI Part 3 & Java Database Connectivity (JDBC)

Tujuan Praktikum

1. Mengetahui konsep JDBC dan cara menghubungkan Java dengan database.
2. Menggunakan JDBC Driver dalam aplikasi Java.
3. Mengimplementasikan operasi CRUD (Create, Read, Update, Delete) dalam aplikasi berbasis GUI.

4.1 Memulai JDBC

Java menyediakan fungsi untuk menghubungkan suatu aplikasi yang dibangun dengan bahasa pemrograman **Java** ke dalam **database**. Salah satu cara untuk menghubungkannya adalah dengan menggunakan **JDBC (Java Database Connectivity)**.

Dengan **JDBC**, programmer dapat menggunakan **SQL Statement** untuk membuat, memanipulasi, atau memelihara suatu data dalam database. JDBC memungkinkan komunikasi antara aplikasi Java dengan berbagai **Relational Database Management System (RDBMS)** seperti **MySQL, Oracle, PostgreSQL, Microsoft SQL Server**, dan lainnya.

JDBC merupakan **versi Java dari ODBC (Open Database Connectivity)** yang dibuat oleh **Sun Microsystems**. Cara kerjanya mirip dengan ODBC, tetapi JDBC **sebenarnya dibangun menggunakan Java API**, sehingga dapat digunakan di berbagai **platform (cross-platform)**. Sedangkan ODBC dikembangkan dalam bahasa **C**, sehingga bergantung pada platform tertentu.

Seperti halnya ODBC, JDBC didasarkan pada **X/Open SQL Call Level Interface (CLI)**, yang memungkinkan aplikasi untuk berkomunikasi dengan database menggunakan **perintah SQL standar**.

Dengan **JDBC API**, kita dapat mengakses database dari berbagai vendor ternama seperti **Oracle, MySQL, Sybase, Informix, dan Interbase**. Untuk menghubungkan aplikasi dengan database, diperlukan **JDBC Driver** yang sesuai dengan jenis database yang digunakan. Setiap vendor database memiliki driver yang berbeda, yang dapat diunduh dari situs resmi vendor tersebut.

Dalam Java, **kelas dan interface JDBC API** tersedia dalam paket:

- **java.sql** → (Core API) untuk pengelolaan database dasar.
- **javax.sql** → (Standard Extension API) untuk fitur tambahan seperti **connection pooling** dan **row sets**.

4.2 Menghubungkan Java dengan Database

4.2.1 Cara Kerja JDBC

JDBC bekerja dengan **empat komponen utama**, yaitu:

1. **JDBC API**
 - Sekumpulan kelas dan interface dalam **java.sql** yang digunakan untuk berkomunikasi dengan database.
2. **JDBC Driver**
 - Driver yang menerjemahkan perintah Java ke dalam bahasa yang dapat dimengerti oleh database tertentu.

3. Database Connection

- Koneksi antara aplikasi dan database untuk mengirim dan menerima data.

4. Query Execution

- Perintah SQL dieksekusi melalui aplikasi untuk mengelola data dalam database.

4.2.2 Menambahkan JDBC Driver di Eclipse

Agar Java dapat berkomunikasi dengan database, kita perlu menambahkan **MySQL JDBC Driver** ke dalam proyek. Terdapat **dua cara** untuk menambahkan JDBC Driver di Eclipse:

Opsi 1: Menggunakan JAR Secara Manual

1. Unduh MySQL Connector/J

- Kunjungi situs resmi MySQL: <https://dev.mysql.com/downloads/connector/j/>
- Unduh versi terbaru dari **MySQL Connector/J**.

2. Tambahkan Library JDBC di Eclipse

- Buka Eclipse dan buka proyek yang sedang dikerjakan.
- Klik kanan pada proyek → **Properties**.
- Pilih **Java Build Path** → **Libraries**.
- Klik **Add External JARs...** dan pilih file **mysql-connector-java-xx.jar** yang telah diunduh.
- Klik **Apply and Close**.

Opsi 2: Menggunakan Maven (pom.xml)

Jika proyek menggunakan **Maven**, maka **tidak perlu menambahkan driver secara manual**. Cukup tambahkan dependensi berikut di dalam **pom.xml**:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>cobaPOM</groupId>
  <artifactId>cobaPOM</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>cobaPOM</name>
  <dependencies>
    <!-- JDBC Driver untuk MySQL -->
    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>8.0.33</version>
    </dependency>
  </dependencies>
</project>
```

Pastikan bahwa Anda menggunakan versi terbaru dari MySQL JDBC Driver dengan mengeceknya di <https://mvnrepository.com/artifact/mysql/mysql-connector-java>. Setelah menambahkan dependensi, perbarui Maven Dependencies di Eclipse dengan cara Klik kanan proyek → **Maven** → **Update Project (Alt + F5)**

Setelah melakukan salah satu dari kedua cara tersebut, maka Anda dapat membuat main class dan mencoba code ini untuk memastikan bahwa project Anda telah terhubung dengan database.

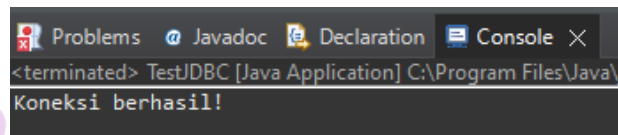
```
package cobaPOM;

import java.sql.Connection;
import java.sql.DriverManager;

public class TestJDBC {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/moduladpl"; // Sesuaikan dengan database Anda
        String user = "root"; // Sesuaikan dengan username database
        String password = ""; // Sesuaikan dengan password database

        try (Connection conn = DriverManager.getConnection(url, user, password)) {
            System.out.println("Koneksi berhasil!");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Jika berhasil dijalankan, maka seharusnya akan muncul pesan seperti gambar dibawah ini.



Gambar 4-1 Berhasil terkoneksi ke Database

4.3 Menggunakan Database dalam Java

Untuk mengelola data dalam database menggunakan **JDBC**, operasi **CRUD (Create, Read, Update, Delete)** perlu diterapkan. Sesuai dengan konsep **MVC**, operasi ini ditempatkan dalam **kelas Model**, sehingga logika bisnis tetap terpisah dari antarmuka pengguna.

4.3.1 Class Database untuk Koneksi

Sebelum melakukan operasi CRUD, diperlukan sebuah kelas Database yang bertanggung jawab atas koneksi ke database MySQL. Berikut adalah implementasi kelas Database, yang mencakup membuka dan menutup koneksi, serta mengeksekusi query:

```
package model;

import java.sql.*;

public class Database {
    static final String url = "jdbc:mysql://localhost:3306/moduladpl";
    static final String user = "root";
    static final String pass = "";
    static Connection conn;
    public static Statement stmt;
    public static ResultSet rs;

    public void connect() {
        try {
            conn = DriverManager.getConnection(url, user, pass);
            stmt = conn.createStatement();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```

        System.out.println("Koneksi Gagal: " + e.getMessage());
    }
}

public void query(String sql) {
    try {
        stmt.executeUpdate(sql);
    } catch (SQLException ex) {
        System.out.println("Kesalahan Query: " + ex.getMessage());
    }
}

public ResultSet view(String sql) {
    try {
        rs = stmt.executeQuery(sql);
    } catch (SQLException ex) {
        System.out.println("Kesalahan View: " + ex.getMessage());
    }
    return rs;
}

public void disconnect() {
    try {
        conn.close();
    } catch (SQLException ex) {
        System.out.println("Kesalahan saat menutup koneksi: " +
ex.getMessage());
    }
}
}

```

1. **connect()** → Mewakili data dan logika bisnis aplikasi. Model bertanggung jawab untuk mengelola data, aturan bisnis, dan logika aplikasi.
2. **query(String sql)** → Menyediakan representasi visual dari data yang ada di Model. View bertanggung jawab untuk menampilkan data kepada pengguna dan menerima input dari pengguna.
3. **view(String sql)** → Menghubungkan Model dan View. Controller menerima input dari pengguna melalui View, memprosesnya (misalnya, dengan memvalidasi data), dan memperbarui Model atau View sesuai kebutuhan.
4. **disconnect()** → Menghubungkan Model dan View. Controller menerima input dari pengguna melalui View, memprosesnya (misalnya, dengan memvalidasi data), dan memperbarui Model atau View sesuai kebutuhan.

4.3.2 Insert / Create Data

Operasi **Create (INSERT)** digunakan untuk menambahkan data baru ke dalam tabel database.

Berikut merupakan contoh kode Penjualan.java (Model – Insert Data).

```

package model;

public class Penjualan {
    private String kode;
    private String nama;
    private int harga;

    public Penjualan(String kode, String nama, int harga) {
        this.kode = kode;
        this.nama = nama;
        this.harga = harga;
    }
}

```

```

    }

    public void insertPenjualan() {
        Database db = new Database();
        db.connect();
        String sql = "INSERT INTO tabel_penjualan (kode, nama, harga) VALUES ('" +
this.kode + "', '" + this.nama + "', " + this.harga + ")";
        db.query(sql);
        db.disconnect();
    }
}

```

1. **InsertPenjualan()** → digunakan untuk memasukkan data ke database dengan perintah INSERT INTO.
2. Koneksi ke database dibuat melalui kelas Database, sehingga kode lebih modular.

4.3.3 Read Data

Operasi **Read (SELECT)** digunakan untuk mengambil data dari database.

Berikut merupakan contoh kode Penjualan.java (Model – Read Data).

```

package model;

import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

public class Penjualan {
    private String kode;
    private String nama;
    private int harga;

    public Penjualan(String kode, String nama, int harga) {
        this.kode = kode;
        this.nama = nama;
        this.harga = harga;
    }

    public static List<Penjualan> getAllPenjualan() {
        List<Penjualan> daftarPenjualan = new ArrayList<>();
        Database db = new Database();
        db.connect();
        String sql = "SELECT * FROM tabel_penjualan";
        ResultSet rs = db.view(sql);

        try {
            while (rs.next()) {
                Penjualan p = new Penjualan(rs.getString("kode"),
rs.getString("nama"), rs.getInt("harga"));
                daftarPenjualan.add(p);
            }
        } catch (SQLException e) {
            System.out.println("Kesalahan saat membaca data: " + e.getMessage());
        }

        db.disconnect();
        return daftarPenjualan;
    }
}

```

1. **SELECT * FROM tabel_penjualan** digunakan untuk membaca semua data dalam tabel.

2. Hasil yang diperoleh disimpan dalam bentuk **List<Penjualan>**, sehingga lebih mudah digunakan dalam antarmuka pengguna.

4.3.4 Edit/ Update Data

perasi **Update (UPDATE)** digunakan untuk memperbarui data yang sudah ada dalam database.

Berikut merupakan contoh kode Penjualan.java (Model – Update Data).

```
package model;

public class Penjualan {
    private String kode;
    private String nama;
    private int harga;

    public Penjualan(String kode, String nama, int harga) {
        this.kode = kode;
        this.nama = nama;
        this.harga = harga;
    }

    public void updatePenjualan(String kodeBaru) {
        Database db = new Database();
        db.connect();
        String sql = "UPDATE tabel_penjualan SET kode='" + kodeBaru + "', nama='"
+ this.nama + "', harga=" + this.harga + " WHERE kode='" + this.kode + "'";
        db.query(sql);
        db.disconnect();
    }
}
```

1. **UPDATE tabel_penjualan SET ... WHERE kode = 'kodeLama'** digunakan untuk memperbarui data dalam tabel.
2. Perubahan data dilakukan berdasarkan **kode barang lama** yang akan diperbarui.

4.3.5 Delete Data

Operasi **Delete (DELETE)** digunakan untuk menghapus data dari database berdasarkan kode barang.

Berikut merupakan contoh kode Penjualan.java (Model – Delete Data).

```
package model;

public class Penjualan {
    private String kode;
    private String nama;
    private int harga;

    public Penjualan(String kode, String nama, int harga) {
        this.kode = kode;
        this.nama = nama;
        this.harga = harga;
    }

    public void deletePenjualan() {
        Database db = new Database();
    }
```

```

        db.connect();
        String sql = "DELETE FROM tabel_penjualan WHERE kode='" + this.kode + "'";
        db.query(sql);
        db.disconnect();
    }
}

```

1. **InsertPenjualan()** → digunakan untuk memasukkan data ke database dengan perintah INSERT INTO.
2. Koneksi ke database dibuat melalui kelas Database, sehingga kode lebih modular.

4.3.6 Contoh Penerapan MVC dengan JDBC (CRUD)

Berikut merupakan contoh kode pengimplementasian JDBC dengan arsitektur MVC.

Model Penjualan.Java

```

package Model;

import java.sql.ResultSet;

public class Penjualan {
    private String kode;
    private String nama;
    private int harga;

    public Penjualan(String kode, String nama, int harga) {
        this.kode = kode;
        this.nama = nama;
        this.harga = harga;
    }

    Getter & setter

    public static List<Penjualan> getAllPenjualan() {
        List<Penjualan> daftarPenjualan = new ArrayList<>();
        Database db = new Database();
        db.connect();
        ResultSet rs = db.view("SELECT * FROM tabel_penjualan");

        try {
            while (rs.next()) {
                daftarPenjualan.add(new Penjualan(rs.getString("kode"),
rs.getString("nama"), rs.getInt("harga")));
            }
        } catch (SQLException e) {
            System.out.println("Kesalahan saat membaca data: " + e.getMessage());
        }

        db.disconnect();
        return daftarPenjualan;
    }

    public void save() {
        Database db = new Database();
        db.connect();
        ResultSet rs = db.view("SELECT * FROM tabel_penjualan WHERE kode='" + kode
+ "'");

        try {
            if (rs.next()) {
                db.query("UPDATE tabel_penjualan SET nama='" + nama + "', harga="
+ harga + " WHERE kode='" + kode + "'");
            }
        }
    }
}

```

```

        } else {
            db.query("INSERT INTO tabel_penjualan (kode, nama, harga) VALUES
('" + kode + "', '" + nama + "', " + harga + ")");
        }
    } catch (SQLException e) {
        System.out.println("Kesalahan: " + e.getMessage());
    }

    db.disconnect();
}

public static void delete(String kode) {
    Database db = new Database();
    db.connect();
    db.query("DELETE FROM tabel_penjualan WHERE kode='" + kode + "'");
    db.disconnect();
}
}

```

View PenjualanView.java

```

package view;

import javax.swing.*;

public class PenjualanView extends JFrame {
    private final JTextField textKode, textNama, textHarga;
    private final JTable table;
    private final DefaultTableModel tableModel;
    private final JButton btnSimpan, btnHapus;

    public PenjualanView() {
        setTitle("Manajemen Data Penjualan");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(600, 400);
        setLayout(new BorderLayout());

        JPanel panelForm = new JPanel(new GridLayout(4, 2));
        panelForm.add(new JLabel("Kode:"));
        textKode = new JTextField();
        panelForm.add(textKode);

        panelForm.add(new JLabel("Nama:"));
        textNama = new JTextField();
        panelForm.add(textNama);

        panelForm.add(new JLabel("Harga:"));
        textHarga = new JTextField();
        panelForm.add(textHarga);

        btnSimpan = new JButton("Simpan");
        btnHapus = new JButton("Hapus");
        panelForm.add(btnSimpan);
        panelForm.add(btnHapus);

        tableModel = new DefaultTableModel(new String[]{"Kode", "Nama", "Harga"},
0);

        table = new JTable(tableModel);
        JScrollPane scrollPane = new JScrollPane(table);

        add(panelForm, BorderLayout.NORTH);
        add(scrollPane, BorderLayout.CENTER);

        setVisible(true);
    }
}

```



```

    public JTextField getTextKode() { return textKode; }
    public JTextField getTextNama() { return textNama; }
    public JTextField getTextHarga() { return textHarga; }
    public JButton getBtnSimpan() { return btnSimpan; }
    public JButton getBtnHapus() { return btnHapus; }
    public JTable getTable() { return table; }
    public DefaultTableModel getTableModel() { return tableModel; }

    public void addSimpanListener(ActionListener listener) {
    btnSimpan.addActionListener(listener); }
    public void addHapusListener(ActionListener listener) {
    btnHapus.addActionListener(listener); }
}

```

Controller PenjualanController.java

```

package controller;

import model.Penjualan;

public class PenjualanController {
    private final PenjualanView view;

    public PenjualanController(PenjualanView view) {
        this.view = view;
        loadTable();

        view.addSimpanListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String kode = view.getTextKode().getText();
                String nama = view.getTextNama().getText();
                int harga = Integer.parseInt(view.getTextHarga().getText());
                new Penjualan(kode, nama, harga).save();
                loadTable();
            }
        });

        view.addHapusListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                int selectedRow = view.getTable().getSelectedRow();
                if (selectedRow != -1) {
                    String kode = (String)
view.getTableModel().getValueAt(selectedRow, 0);
                    Penjualan.delete(kode);
                    loadTable();
                } else {
                    JOptionPane.showMessageDialog(view, "Pilih data yang ingin
dihapus!");
                }
            }
        });
    }

    private void loadTable() {
        DefaultTableModel tableModel = view.getTableModel();
        tableModel.setRowCount(0);
        for (Penjualan p : Penjualan.getAllPenjualan()) {
            tableModel.addRow(new Object[]{p.getKode(), p.getNama(),
p.getHarga()});
        }
    }
}

```

```
}
```

Main.java

```
package Main;
import Controller.PenjualanController;

public class Main {
    public static void main(String[] args) {
        PenjualanView view = new PenjualanView();
        new PenjualanController(view);
    }
}
```

Module-info.java

```
module penjualan {
    requires java.desktop;
    requires java.sql;
}
```

Table Query

```
CREATE TABLE tabel_penjualan (
    kode VARCHAR(10) PRIMARY KEY,
    nama VARCHAR(100) NOT NULL,
    harga INT NOT NULL
);

INSERT INTO tabel_penjualan (kode, nama, harga) VALUES
('P001', 'Laptop', 10000000),
('P002', 'Mouse', 150000),
('P003', 'Keyboard', 350000);
```

Modul 5 Prinsip SOLID Part 1

Tujuan Praktikum

1. Mengetahui prinsip SOLID dalam pemrograman.
2. Memahami dan menerapkan Single Responsibility, Open-Closed, dan Liskov Substitution Principle dalam Java.

5.1 Pendahuluan tentang SOLID

Dalam pengembangan perangkat lunak berbasis **Object-Oriented Programming (OOP)**, prinsip **SOLID** merupakan sekumpulan pedoman desain yang bertujuan untuk meningkatkan kualitas kode. Prinsip ini membantu menghindari kode yang sulit dikelola, memperbaiki keterbacaan, serta meningkatkan skalabilitas dan fleksibilitas aplikasi.

Prinsip **SOLID** diperkenalkan oleh **Robert C. Martin** dalam esainya yang berjudul *Design Principles and Design Patterns* pada tahun 2000. Istilah **SOLID** sendiri pertama kali diperkenalkan oleh **Michael Feathers**, yang membentuk akronim dari lima prinsip utama desain perangkat lunak:

1. **Single Responsibility Principle (SRP)**
2. **Open-Closed Principle (OCP)**
3. **Liskov Substitution Principle (LSP)**
4. **Interface Segregation Principle (ISP)**
5. **Dependency Inversion Principle (DIP)**

Ketika suatu perangkat lunak berkembang, kompleksitasnya juga meningkat. Jika tidak didesain dengan baik, perangkat lunak akan menjadi **rigid** (sulit diubah), **fragile** (mudah rusak saat ada perubahan), **immobile** (sulit digunakan kembali), dan **vicious** (membuat perubahan kecil berdampak besar).

Tujuan utama dari **prinsip SOLID** adalah **mengurangi ketergantungan antar komponen** sehingga perubahan dalam satu bagian tidak mempengaruhi bagian lain. Selain itu, prinsip ini juga membuat kode lebih mudah **dibaca, diperluas, dan dipelihara**.

Pada bab ini, akan dibahas tiga prinsip pertama dari SOLID, yaitu **Single Responsibility Principle (SRP)**, **Open-Closed Principle (OCP)**, dan **Liskov Substitution Principle (LSP)**.

5.2 Single Responsibility Principle (SRP)

5.2.1 Definisi dan Konsep

Single Responsibility Principle (SRP) menyatakan bahwa setiap modul atau kelas harus memiliki satu alasan untuk berubah. Artinya, sebuah kelas **tidak boleh menangani lebih dari satu tujuan utama**. Jika suatu kelas memiliki beberapa tanggung jawab, maka perubahan dalam satu tanggung jawab dapat berdampak pada bagian lain yang tidak terkait, sehingga kode menjadi sulit dipelihara.

Kita harus mendesain perangkat lunak sedemikian rupa sehingga setiap elemen dalam kelas atau modul berhubungan dengan satu tanggung jawab tertentu. Ini tidak berarti bahwa setiap kelas hanya boleh memiliki satu metode atau properti, tetapi semua metode dan properti dalam kelas

tersebut harus berkaitan dengan satu tanggung jawab utama. Sebagai hasil dari penerapan SRP, kelas menjadi lebih kecil dan lebih bersih, sehingga lebih mudah untuk dipahami, dipelihara, dan diuji.

5.2.2 Contoh Kode Tanpa SRP (Kode Buruk)

Berikut adalah contoh kode yang **melanggar SRP**, karena satu kelas memiliki lebih dari satu tanggung jawab:

```
public class ReportGenerator {
    public void generateReport() {
        System.out.println("Membuat laporan...");
    }

    public void saveToFile(String filename) {
        System.out.println("Menyimpan laporan ke " + filename);
    }

    public void sendEmail(String email) {
        System.out.println("Mengirim laporan ke email: " + email);
    }
}
```

Pada kode di atas, kelas **ReportGenerator** menangani tiga tanggung jawab berbeda:

1. **Membuat laporan** (`generateReport()`)
2. **Menyimpan laporan ke file** (`saveToFile()`)
3. **Mengirim laporan ke email** (`sendEmail()`)

Jika ada perubahan dalam cara laporan dikirim ke email, maka kita harus mengubah `ReportGenerator`, yang juga memengaruhi cara laporan dibuat dan disimpan. Ini **melanggar SRP**, karena satu kelas memiliki lebih dari satu alasan untuk berubah.

5.2.3 Contoh Kode yang Mematuhi SRP (Kode Baik)

Untuk mematuhi SRP, tanggung jawab harus dipisah ke dalam kelas yang berbeda:

```
public class ReportGenerator {
    public void generateReport() {
        System.out.println("Membuat laporan...");
    }
}

public class FileSaver {
    public void saveToFile(String filename) {
        System.out.println("Menyimpan laporan ke " + filename);
    }
}

public class EmailSender {
    public void sendEmail(String email) {
        System.out.println("Mengirim laporan ke email: " + email);
    }
}
```

Sekarang:

1. **ReportGenerator** hanya membuat laporan
2. **FileSaver** hanya menyimpan laporan ke file

3. **EmailSender** hanya mengirim laporan melalui email

Dengan desain ini, jika metode pengiriman email berubah, hanya **EmailSender** yang perlu diperbarui, **tanpa memengaruhi ReportGenerator atau FileSaver**.

5.3 Open-Closed Principle (OCP)

5.3.1 Definisi dan Konsep

Open-Closed Principle (OCP) menyatakan bahwa kelas harus terbuka untuk ekstensi tetapi tertutup untuk modifikasi. Artinya, ketika ada fitur baru yang perlu ditambahkan, seharusnya **tidak perlu mengubah kode yang sudah ada**, melainkan cukup dengan memperluasnya melalui **inheritance atau composition**. Prinsip ini mengurangi risiko memperkenalkan bug ketika menambahkan fitur baru, karena kode yang sudah stabil tidak perlu diubah.

5.3.2 Contoh Kode Tanpa OCP (Kode Buruk)

Berikut adalah contoh kode yang melanggar **Open-Closed Principle**, karena setiap kali kita ingin menambahkan jenis diskon baru, kita harus **mengubah** kode yang sudah ada:

```
public class DiscountCalculator {
    public double calculateDiscount(String type, double price) {
        if (type.equals("Member")) {
            return price * 0.1;
        } else if (type.equals("VIP")) {
            return price * 0.2;
        }
        return 0;
    }
}
```

Jika terdapat tipe diskon baru (misalnya "Promo"), kita **harus mengubah metode calculateDiscount**. Ini melanggar **OCP**, karena kode tidak **tertutup untuk modifikasi**.

5.3.3 Contoh Kode yang Mematuhi OCP (Kode Baik)

Untuk mematuhi **Open-Closed Principle (OCP)**, suatu kode harus memungkinkan ekstensi tanpa perlu melakukan perubahan pada kode yang sudah ada. Dengan kata lain, ketika ada kebutuhan untuk menambahkan fitur baru, pengembang cukup menambahkan kode baru tanpa mengubah kode yang telah stabil.

Salah satu pendekatan yang sering digunakan dalam penerapan OCP adalah dengan menggunakan **antarmuka (interface)**. Dengan menggunakan antarmuka, perilaku atau fitur baru dapat ditambahkan dalam bentuk kelas baru tanpa harus mengubah kode utama. Hal ini memungkinkan sistem tetap fleksibel dan tidak memerlukan modifikasi terhadap kode yang sudah ada, sehingga mengurangi risiko terjadinya bug akibat perubahan.

Selain antarmuka, pendekatan lain yang dapat digunakan adalah dengan **kelas abstrak (abstract class)**. Dengan menggunakan kelas abstrak, metode umum dapat didefinisikan dalam kelas induk, sementara detail spesifiknya didelegasikan ke kelas turunannya. Hal ini memungkinkan fleksibilitas yang lebih besar dalam pengembangan perangkat lunak, di mana kode dapat diperluas tanpa perlu melakukan perubahan langsung terhadap kode dasar.

Berikut adalah contoh penerapan **OCP menggunakan antarmuka (implements)**:

```
public interface Discount {
```

```

    double applyDiscount(double price);
}

public class MemberDiscount implements Discount {
    public double applyDiscount(double price) {
        return price * 0.1;
    }
}

public class VIPDiscount implements Discount {
    public double applyDiscount(double price) {
        return price * 0.2;
    }
}

public class DiscountCalculator {
    public double calculateDiscount(Discount discount, double price) {
        return discount.applyDiscount(price);
    }
}

```

Dengan pendekatan ini, jika ada jenis diskon baru, cukup dengan menambahkan kelas baru yang mengimplementasikan antarmuka `Discount`, tanpa perlu mengubah kode di dalam `DiscountCalculator`. Hal ini memastikan bahwa kode tetap modular, mudah diperluas, dan tidak melanggar prinsip Open-Closed.

5.4 Liskov Substitution Principle (LSP)

5.4.1 Definisi dan Konsep

Liskov Substitution Principle (LSP) adalah salah satu prinsip dalam Object-Oriented Programming (OOP) yang menyatakan bahwa **objek dari suatu superclass harus dapat digantikan dengan objek dari subclassnya tanpa mengubah keakuratan program**. Dengan kata lain, subclass harus benar-benar menjadi tipe yang valid dari superclassnya, sehingga dapat digunakan di mana saja objek dari superclass digunakan tanpa menimbulkan bug baru atau menyebabkan kerusakan pada fungsi yang sudah ada.

Prinsip ini menjadi salah satu fondasi dalam desain perangkat lunak yang modular dan fleksibel. Jika suatu subclass **tidak mematuhi aturan dari superclass**, maka akan terjadi **inkonsistensi dalam sistem**, yang dapat menyebabkan bug yang sulit ditelusuri dan diperbaiki.

Agar sebuah subclass memenuhi **Liskov Substitution Principle (LSP)**, terdapat beberapa aturan yang harus dipenuhi:

1. Subclass harus memiliki semua properti dan metode yang ada pada superclass.
 - Subclass tidak boleh menghilangkan atau mengubah atribut atau metode yang ada dalam superclass.
2. Subclass tidak boleh menambahkan properti atau metode baru yang tidak ada dalam superclass.
 - Jika subclass memiliki metode baru yang tidak ada di superclass, ini dapat menyebabkan perilaku yang tidak terduga saat subclass digunakan sebagai pengganti superclass.
3. Subclass tidak boleh mengurangi visibilitas properti atau metode yang ada di superclass.

- Jika suatu metode di superclass bersifat public, maka metode yang diwarisi di subclass juga harus tetap public, tidak boleh diubah menjadi private atau protected.
4. Subclass tidak boleh mengubah perilaku dari properti atau metode yang diwarisi dari superclass dengan cara yang dapat merusak logika sistem yang sudah ada.
- Jika suatu metode dalam subclass memiliki perilaku yang berbeda dari superclass sehingga dapat menyebabkan fungsi yang sudah ada rusak, maka prinsip LSP dilanggar.

Jika aturan ini tidak dipatuhi, maka penggunaan subclass sebagai pengganti superclass dapat menyebabkan error atau ketidaksesuaian dalam program, yang melanggar Liskov Substitution Principle.

5.4.2 Contoh Kode yang Melanggar LSP (Kode Buruk)

Untuk memahami pelanggaran **Liskov Substitution Principle**, mari kita lihat contoh **Rectangle-Square Problem**, di mana Square dianggap sebagai turunan dari Rectangle.

```
public class Rectangle {
    protected int width, height;

    public void setWidth(int width) {
        this.width = width;
    }

    public void setHeight(int height) {
        this.height = height;
    }

    public int getArea() {
        return width * height;
    }
}

public class Square extends Rectangle {
    public void setWidth(int width) {
        this.width = this.height = width;
    }

    public void setHeight(int height) {
        this.width = this.height = height;
    }
}
```

Pada kode di atas, Square merupakan subclass dari Rectangle, tetapi mengubah cara kerja metode `setWidth()` dan `setHeight()`. Sehingga, ketika objek Square digunakan sebagai pengganti Rectangle, akan terjadi perilaku yang **tidak sesuai dengan harapan**.

Mari kita lihat contoh kasus berikut:

```
Rectangle rect = new Square();
rect.setWidth(5);
rect.setHeight(10);
System.out.println(rect.getArea()); // Output yang diharapkan: 50, tapi hasilnya: 100
```

Jika Rectangle digunakan seperti biasa, area yang diharapkan adalah **50**. Namun, ketika kita menggantinya dengan Square, area yang dihasilkan menjadi **100**, karena Square secara otomatis mengubah tinggi (height) ketika lebar (width) diubah.

Hal ini melanggar **Liskov Substitution Principle**, karena **Square** tidak benar-benar bisa menggantikan **Rectangle** tanpa menyebabkan error atau perubahan perilaku program.

5.4.3 Contoh Kode yang Mematuhi LSP (Kode Baik)

Untuk mematuhi **Liskov Substitution Principle**, subclass harus benar-benar bisa menggantikan superclass tanpa menyebabkan perubahan perilaku yang tidak diinginkan. Salah satu solusi terbaik adalah dengan **tidak membuat Square sebagai subclass dari Rectangle**, melainkan membuatnya sebagai **kelas yang terpisah** dengan pendekatan yang lebih sesuai:

```
public class Rectangle {
    private int width, height;

    public Rectangle(int width, int height) {
        this.width = width;
        this.height = height;
    }

    public void setWidth(int width) {
        this.width = width;
    }

    public void setHeight(int height) {
        this.height = height;
    }

    public int getArea() {
        return width * height;
    }
}

public class Square {
    private int side;

    public Square(int side) {
        this.side = side;
    }

    public void setSide(int side) {
        this.side = side;
    }

    public int getArea() {
        return side * side;
    }
}
```

Dengan pendekatan ini:

1. **Rectangle** tetap berfungsi sebagaimana mestinya, tanpa perubahan perilaku.
2. **Square** tidak perlu dipaksa untuk mengikuti kontrak dari **Rectangle**, sehingga tetap memiliki aturan sendiri.
3. Keduanya dapat digunakan secara terpisah tanpa melanggar **Liskov Substitution Principle**.

Modul 6 Prinsip SOLID Part 2

Tujuan Praktikum

1. Memahami Interface Segregation Principle dan Dependency Inversion Principle.
2. Mengimplementasikan prinsip SOLID dalam pengembangan aplikasi.

6.1 Pendahuluan

Pada bab sebelumnya, telah dibahas tiga prinsip pertama dalam **SOLID: Single Responsibility Principle (SRP), Open-Closed Principle (OCP), dan Liskov Substitution Principle (LSP)**. Bab ini akan melanjutkan pembahasan mengenai dua prinsip terakhir dalam **SOLID**, yaitu:

1. **Interface Segregation Principle (ISP)**
2. **Dependency Inversion Principle (DIP)**

Kedua prinsip ini berfokus pada desain **antarmuka dan ketergantungan antar komponen** dalam sistem perangkat lunak. **Interface Segregation Principle (ISP)** memastikan bahwa kelas tidak dipaksa untuk mengimplementasikan antarmuka yang tidak mereka butuhkan, sementara **Dependency Inversion Principle (DIP)** bertujuan untuk mengurangi ketergantungan langsung antara modul dengan memperkenalkan abstraksi.

Dengan menerapkan kedua prinsip ini, kode menjadi lebih modular, fleksibel, dan lebih mudah dikelola seiring dengan perkembangan sistem.

6.2 Interface Segregation Principle (ISP)

6.2.1 Definisi dan Konsep

Interface Segregation Principle (ISP) menyatakan bahwa **sebuah kelas tidak boleh dipaksa untuk mengimplementasikan antarmuka yang tidak digunakannya**. Dengan kata lain, lebih baik memiliki beberapa **antarmuka kecil dan spesifik** daripada satu **antarmuka besar dan serbaguna** yang memuat banyak metode yang tidak relevan bagi semua implementasi.

Prinsip ini bertujuan untuk menghindari **kode yang berat, tidak efisien, dan sulit dipelihara** karena adanya metode yang tidak relevan dalam sebuah kelas. Dengan memisahkan antarmuka menjadi bagian yang lebih kecil dan lebih fokus, setiap kelas hanya perlu mengimplementasikan metode yang benar-benar sesuai dengan perannya.

Agar prinsip ini tetap dipatuhi, sebuah antarmuka harus:

1. Hanya berisi metode yang relevan bagi kelas yang mengimplementasikannya.
2. Dapat dipisah menjadi beberapa antarmuka kecil jika ada terlalu banyak metode yang berbeda dalam satu antarmuka besar.
3. Tidak memaksa kelas untuk mengimplementasikan metode yang tidak digunakan.

Jika prinsip ini dilanggar, kelas akan mengandung metode yang tidak dibutuhkan, sehingga dapat menyebabkan **kode menjadi lebih kompleks dan sulit dipahami**.

6.2.2 Contoh Kode yang Melanggar ISP (Kode Buruk)

Berikut adalah contoh pelanggaran **Interface Segregation Principle**, di mana satu antarmuka besar (**IPrinter**) memuat metode yang tidak relevan untuk semua kelas yang mengimplementasikannya:

```
public interface IPrinter {  
    void print(String document);  
}
```

```

    void scan(String document);
    void fax(String document);
}

public class BasicPrinter implements IPrinter {
    public void print(String document) {
        System.out.println("Mencetak dokumen: " + document);
    }

    public void scan(String document) {
        throw new UnsupportedOperationException("Basic printer tidak bisa melakukan scan!");
    }

    public void fax(String document) {
        throw new UnsupportedOperationException("Basic printer tidak bisa melakukan fax!");
    }
}

```

Pada kode di atas, BasicPrinter dipaksa untuk mengimplementasikan scan() dan fax() meskipun printer dasar tidak mendukung fitur tersebut. Akibatnya, metode scan() dan fax() harus dibiarkan kosong atau bahkan memunculkan error, yang melanggar Interface Segregation Principle (ISP).

6.2.3 Contoh Kode yang Mematuhi ISP (Kode Baik)

Untuk mematuhi **Interface Segregation Principle**, antarmuka harus dipecah menjadi bagian yang lebih spesifik sesuai dengan fungsionalitasnya:

```

public interface IPrinter {
    void print(String document);
}

public interface IScanner {
    void scan(String document);
}

public interface IFax {
    void fax(String document);
}

public class BasicPrinter implements IPrinter {
    public void print(String document) {
        System.out.println("Mencetak dokumen: " + document);
    }
}

public class MultiFunctionPrinter implements IPrinter, IScanner, IFax {
    public void print(String document) {
        System.out.println("Mencetak dokumen: " + document);
    }

    public void scan(String document) {
        System.out.println("Memindai dokumen: " + document);
    }

    public void fax(String document) {
        System.out.println("Mengirim fax: " + document);
    }
}

```

Sekarang, BasicPrinter hanya mengimplementasikan **antarmuka IPrinter**, sehingga tidak memiliki metode yang tidak digunakannya. Sementara itu, MultiFunctionPrinter mengimplementasikan semua antarmuka karena memiliki semua fitur yang dibutuhkan.

Pendekatan ini memastikan bahwa kelas hanya memiliki metode yang relevan dan tidak dipaksa mengimplementasikan fungsionalitas yang tidak mereka butuhkan, sesuai dengan Interface Segregation Principle.

6.3 Dependency Inversion Principle (DIP)

6.3.1 Definisi dan Konsep

Dependency Inversion Principle (DIP) merupakan salah satu prinsip dalam **SOLID** yang bertujuan untuk mengurangi ketergantungan langsung antara modul tingkat tinggi dan modul tingkat rendah. **Modul tingkat tinggi tidak boleh bergantung langsung pada modul tingkat rendah. Sebaliknya, keduanya harus bergantung pada abstraksi.**

Dengan menerapkan DIP, desain sistem menjadi lebih **fleksibel, modular, dan mudah diperluas**. Jika suatu modul terlalu bergantung pada modul lain, maka setiap perubahan dalam satu bagian kode dapat memengaruhi bagian lain, membuat kode menjadi sulit untuk diperbaiki dan dipelihara. Dengan adanya **abstraksi**, perubahan dalam implementasi dapat dilakukan tanpa mengganggu bagian lain dalam sistem.

Aturan utama **Dependency Inversion Principle** adalah:

1. Modul tingkat tinggi tidak boleh bergantung langsung pada modul tingkat rendah.
2. Baik modul tingkat tinggi maupun rendah harus bergantung pada abstraksi.
3. Abstraksi tidak boleh bergantung pada detail; sebaliknya, detail harus bergantung pada abstraksi.

Jika prinsip ini dilanggar, sistem akan memiliki **tight coupling (ketergantungan yang erat)**, yang menyebabkan sulitnya memperbarui atau mengganti bagian kode tertentu tanpa memengaruhi seluruh sistem.

6.3.2 Contoh Kode yang Melanggar DIP (Kode Buruk)

Berikut adalah contoh kode yang **melanggar Dependency Inversion Principle**, di mana kelas Car langsung bergantung pada kelas konkret FuelInjector:

```
public class Car {
    private FuelInjector fuelInjector;

    public Car() {
        this.fuelInjector = new FuelInjector();
    }

    public void start() {
        fuelInjector.injectFuel();
        System.out.println("Mobil dinyalakan.");
    }
}

public class FuelInjector {
    public void injectFuel() {
        System.out.println("Menyuntikkan bahan bakar.");
    }
}
```

Pada kode di atas:

1. Car langsung membuat objek FuelInjector di dalam konstruktornya, sehingga memiliki ketergantungan kuat (tight coupling) terhadap implementasi spesifik FuelInjector.

2. Jika kita ingin mengganti sistem injeksi bahan bakar, misalnya dengan **ElectricFuelInjector**, kita harus **mengubah kode dalam Car**, yang berisiko menyebabkan bug di seluruh sistem.
3. Kode menjadi sulit diuji karena **Car selalu menggunakan FuelInjector tanpa fleksibilitas untuk menggunakan implementasi lain**.

Dengan kata lain, **Car "terjebak" dengan FuelInjector**, dan tidak dapat dengan mudah diperluas untuk mendukung implementasi lain tanpa melakukan modifikasi langsung dalam kelas Car.

6.3.3 Contoh Kode yang Mematuhi DIP (Kode Baik)

Untuk mematuhi **Dependency Inversion Principle**, kita perlu **memisahkan Car dari FuelInjector dengan menggunakan abstraksi**. Salah satu cara terbaik untuk melakukan ini adalah dengan menggunakan **interface** yang menjadi perantara antara keduanya.

Pada kode di atas:

1. Membuat antarmuka IFuelInjector sebagai abstraksi.
2. FuelInjector akan mengimplementasikan IFuelInjector.
3. **Car akan menerima IFuelInjector melalui Dependency Injection**, sehingga bisa menggunakan berbagai jenis injeksi bahan bakar tanpa mengubah kode Car.

Berikut adalah kode yang telah diperbaiki:

```
// Abstraksi: Interface sebagai perantara antara Car dan Fuel Injector
public interface IFuelInjector {
    void injectFuel();
}

// Implementasi konkret dari IFuelInjector
public class FuelInjector implements IFuelInjector {
    public void injectFuel() {
        System.out.println("Menyuntikkan bahan bakar.");
    }
}

// Implementasi lain dari IFuelInjector (untuk menunjukkan fleksibilitas)
public class ElectricFuelInjector implements IFuelInjector {
    public void injectFuel() {
        System.out.println("Menggunakan listrik untuk menyuplai bahan bakar.");
    }
}

// Kelas Car hanya bergantung pada abstraksi (IFuelInjector), bukan implementasi langsung
public class Car {
    private IFuelInjector fuelInjector;

    // Dependency Injection melalui konstruktor
    public Car(IFuelInjector fuelInjector) {
        this.fuelInjector = fuelInjector;
    }

    public void start() {
        fuelInjector.injectFuel();
        System.out.println("Mobil dinyalakan.");
    }
}

// Main Program untuk menjalankan kode
public class Main {
    public static void main(String[] args) {
        // Menggunakan FuelInjector sebagai dependency
        IFuelInjector fuelInjector = new FuelInjector();
        Car myCar = new Car(fuelInjector);
    }
}
```

```
myCar.start();

// Menggunakan ElectricFuelInjector sebagai dependency
IFuelInjector electricFuelInjector = new ElectricFuelInjector();
Car electricCar = new Car(electricFuelInjector);
electricCar.start();
}
```

Dengan dilakukan-nya cara diatas, maka:

1. Kelas Car tidak lagi bergantung langsung pada FuelInjector.
2. Kelas Car bergantung pada antarmuka IFuelInjector, yang merupakan abstraksi.
3. Jika ada jenis sistem injeksi bahan bakar baru, cukup buat kelas baru yang mengimplementasikan IFuelInjector, tanpa mengubah Car.
4. Kode lebih fleksibel dan dapat diuji dengan lebih mudah.

Dengan desain ini, kita dapat dengan mudah menambahkan atau mengganti sistem injeksi bahan bakar tanpa perlu memodifikasi Car, **memenuhi prinsip Dependency Inversion** dengan sempurna.



Modul 7 Penerapan SOLID Dalam Studi Kasus Pemrograman

Tujuan Praktikum

1. Menggunakan prinsip SOLID dalam studi kasus pemrograman.
2. Mengembangkan kode yang lebih modular dan terstruktur.

7.1 Pendahuluan

Pada bab sebelumnya, kita telah membahas **lima prinsip dalam SOLID** secara terpisah. Namun, dalam pengembangan perangkat lunak nyata, prinsip-prinsip ini sering digunakan **bersama-sama** untuk menciptakan kode yang lebih **modular, fleksibel, dan mudah dipelihara**.

Pada bab ini, kita akan menerapkan semua prinsip **SOLID** dalam sebuah **studi kasus nyata**, yang kemudian akan kita analisis per prinsipnya. Dengan cara ini, kita dapat melihat **bagaimana prinsip-prinsip tersebut saling mendukung** dalam menciptakan kode yang berkualitas tinggi.

7.2 Studi Kasus dan Full Code

Kita akan membangun sistem sederhana untuk **manajemen pembayaran** yang memungkinkan pengguna melakukan pembayaran menggunakan berbagai metode seperti **kartu kredit, e-wallet, dan transfer bank**.

Sistem ini akan memiliki fitur:

1. Melakukan pembayaran dengan berbagai metode.
2. Menyimpan riwayat transaksi.
3. Menggunakan pola desain yang mematuhi SOLID agar mudah diperluas.

Sebelum melihat kode lengkapnya, berikut adalah **struktur kelas dan package yang digunakan**:

```
- model/  
  - Payment.java           // Interface untuk abstraksi metode pembayaran  
  - CreditCardPayment.java // Implementasi pembayaran dengan kartu kredit  
  - EWalletPayment.java    // Implementasi pembayaran dengan e-wallet  
  - BankTransferPayment.java // Implementasi pembayaran dengan transfer bank  
  - Transaction.java       // Model untuk menyimpan transaksi  
  
- service/  
  - PaymentProcessor.java  // Mengelola pembayaran, menerapkan DIP  
  - TransactionHistory.java // Mengelola riwayat transaksi  
  
- main/  
  - Main.java              // Program utama yang menjalankan sistem
```

Kode Lengkap Implementasi

Abstraksi Metode Pembayaran (Payment)

```
package model;  
  
public interface Payment {  
    void pay(double amount);  
}
```

Implementasi Pembayaran dengan berbagai Metode

```
package model;

// Pembayaran menggunakan kartu kredit
public class CreditCardPayment implements Payment {
    public void pay(double amount) {
        System.out.println("Membayar " + amount + " menggunakan Kartu Kredit.");
    }
}
```

```
package model;

// Pembayaran menggunakan e-wallet
public class EWalletPayment implements Payment {
    public void pay(double amount) {
        System.out.println("Membayar " + amount + " menggunakan E-Wallet.");
    }
}
```

```
package model;

// Pembayaran menggunakan transfer bank
public class BankTransferPayment implements Payment {
    public void pay(double amount) {
        System.out.println("Membayar " + amount + " menggunakan Transfer Bank.");
    }
}
```

Model Transaksi

```
package model;

public class Transaction {
    private String id;
    private double amount;
    private Payment paymentMethod;

    public Transaction(String id, double amount, Payment paymentMethod) {
        this.id = id;
        this.amount = amount;
        this.paymentMethod = paymentMethod;
    }

    public void processPayment() {
        paymentMethod.pay(amount);
        System.out.println("Transaksi " + id + " berhasil.");
    }

    @Override
    public String toString() {
        return "ID: " + id + ", Jumlah: " + amount + ", Metode: " +
            paymentMethod.getClass().getSimpleName();
    }
}
```

Pengelola Pembayaran (PaymentProcessor)

```
package service;

import model.Payment;
import model.Transaction;

public class PaymentProcessor {
    private TransactionHistory transactionHistory;

    public PaymentProcessor(TransactionHistory transactionHistory) {
        this.transactionHistory = transactionHistory;
    }

    public void processTransaction(String id, double amount, Payment paymentMethod)
    {
        Transaction transaction = new Transaction(id, amount, paymentMethod);
        transaction.processPayment();
        transactionHistory.addTransaction(transaction);
    }
}
```

Penyimpanan Riwayat Transaksi (Transaction History)

```
package service;

import model.Transaction;
import java.util.ArrayList;
import java.util.List;

public class TransactionHistory {
    private List<Transaction> transactions = new ArrayList<>();

    public void addTransaction(Transaction transaction) {
        transactions.add(transaction);
    }

    public void showTransactions() {
        System.out.println("\n=== Riwayat Transaksi ===");
        if (transactions.isEmpty()) {
            System.out.println("Belum ada transaksi.");
        } else {
            for (Transaction transaction : transactions) {
                System.out.println(transaction);
            }
        }
    }
}
```

Program Utama (Main)

```
package main;

import model.*;
import service.PaymentProcessor;
import service.TransactionHistory;

import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
    }
}
```



```

TransactionHistory history = new TransactionHistory();
PaymentProcessor processor = new PaymentProcessor(history);

Payment creditCard = new CreditCardPayment();
Payment eWallet = new EWalletPayment();
Payment bankTransfer = new BankTransferPayment();

// Simulasi transaksi
processor.processTransaction("TXN001", 100.0, creditCard);
processor.processTransaction("TXN002", 200.0, eWallet);
processor.processTransaction("TXN003", 300.0, bankTransfer);

// Menampilkan riwayat transaksi setelah pembayaran dilakukan
history.showTransactions();

// Menambahkan fitur interaktif untuk melihat riwayat transaksi
System.out.println("\nIngin melihat riwayat transaksi lagi? (y/n)");
String input = scanner.nextLine();

if (input.equalsIgnoreCase("y")) {
    history.showTransactions();
}

System.out.println("Terima kasih telah menggunakan sistem pembayaran!");
scanner.close();
}
}

```

7.3 Single Responsibility Principle (SRP)

Single Responsibility Principle (SRP) menyatakan bahwa setiap kelas harus memiliki satu alasan untuk berubah, artinya setiap kelas harus memiliki satu tanggung jawab utama. Prinsip ini diterapkan dalam sistem pembayaran dengan memisahkan tugas ke dalam beberapa kelas sesuai dengan fungsinya masing-masing. Dengan pemisahan ini, perubahan dalam satu bagian kode tidak akan mempengaruhi bagian lainnya, sehingga meningkatkan keterbacaan dan kemudahan pemeliharaan kode. Dalam contoh diatas, SRP diterapkan dengan cara berikut:

1. Kelas **Transaction**: Kelas ini hanya bertanggung jawab untuk menyimpan informasi transaksi. Kelas ini tidak menangani logika pembayaran atau penyimpanan riwayat transaksi.
2. Kelas **PaymentProcessor**: Kelas ini hanya bertanggung jawab untuk menangani proses pembayaran. Kelas ini tidak menyimpan data transaksi dalam dirinya sendiri, tetapi menggunakan class **TransactionHistory**.
3. Kelas **TransactionHistory**: Kelas ini bertanggung jawab hanya untuk menyimpan dan menampilkan riwayat transaksi. Kelas ini tidak menangani logika pembayaran atau pembuatan transaksi.

Dengan menerapkan SRP, setiap kelas memiliki fungsi yang jelas dan spesifik, sehingga jika ada perubahan pada sistem, hanya satu kelas yang perlu diubah tanpa mempengaruhi bagian lain.

7.4 Open-Closed Principle (OCP)

Open-Closed Principle (OCP) menyatakan bahwa suatu kelas harus terbuka untuk ekstensi tetapi tertutup untuk modifikasi. Artinya, kita harus bisa menambahkan fitur baru tanpa harus mengubah kode yang sudah ada. Dalam sistem ini, prinsip OCP diterapkan dalam desain metode pembayaran. Untuk menangani berbagai jenis pembayaran, kita menggunakan antarmuka **Payment** sebagai

dasar, dan setiap metode pembayaran diimplementasikan dalam kelasnya sendiri. Dengan pendekatan ini, kita dapat menambahkan metode pembayaran baru tanpa harus mengubah kode yang sudah ada di **PaymentProcessor**.

Sebagai contoh, jika kita ingin menambahkan metode pembayaran baru seperti **CryptocurrencyPayment**, kita cukup membuat kelas baru yang mengimplementasikan antarmuka **Payment**. Kode yang sudah ada tidak perlu diubah, sehingga sistem tetap stabil dan mudah diperluas.

Penerapan prinsip ini memastikan bahwa **penambahan fitur baru tidak mengganggu fungsi sistem yang sudah berjalan**, menghindari perubahan yang tidak perlu pada kode lama, serta mempermudah pengujian dan pemeliharaan sistem.

7.5 Liskov Substitution Principle (LSP)

Liskov Substitution Principle (LSP) menyatakan bahwa subclass harus bisa menggantikan superclass tanpa mengubah perilaku program. Dalam sistem ini, prinsip ini diterapkan melalui antarmuka **Payment** yang diimplementasikan oleh berbagai kelas metode pembayaran seperti **CreditCardPayment**, **EWalletPayment**, dan **BankTransferPayment**.

Dalam kode ini, kelas **PaymentProcessor** menggunakan referensi **Payment** tanpa mengetahui implementasi spesifiknya. Hal ini memungkinkan kita untuk mengganti metode pembayaran tanpa mengubah logika **PaymentProcessor**. Semua objek yang mengimplementasikan **Payment** dapat digunakan tanpa mengubah cara **PaymentProcessor** bekerja.

Sebagai contoh, jika kita ingin menggunakan **EWalletPayment** alih-alih **CreditCardPayment**, kita cukup mengubah objek yang dikirimkan ke **PaymentProcessor**, tanpa perlu mengubah kode lainnya. Dengan mengikuti prinsip LSP, kita memastikan bahwa semua subclass dapat digunakan secara **interchangeable** tanpa menyebabkan bug atau mengubah perilaku yang tidak diinginkan dalam program.

7.6 Interface Segregation Principle (ISP)

Interface Segregation Principle (ISP) menyatakan bahwa sebuah antarmuka tidak boleh memaksa kelas yang mengimplementasikannya untuk menggunakan metode yang tidak mereka butuhkan. Artinya, lebih baik memiliki beberapa antarmuka kecil dengan metode yang spesifik daripada satu antarmuka besar dengan banyak metode yang tidak selalu relevan untuk semua implementasi.

Dalam sistem pembayaran ini, ISP diterapkan dengan mendefinisikan antarmuka **Payment**, yang hanya memiliki satu metode **pay()**. Dengan pendekatan ini, setiap metode pembayaran hanya perlu mengimplementasikan fungsi pembayaran tanpa harus menangani fitur tambahan yang tidak diperlukan.

Jika antarmuka **Payment** juga memiliki metode lain seperti **refund()** atau **printReceipt()**, maka kelas seperti **EWalletPayment** atau **BankTransferPayment** mungkin harus mengimplementasikan metode yang tidak mereka butuhkan. Dengan memisahkan tanggung jawab ke dalam antarmuka yang lebih kecil, setiap metode pembayaran hanya perlu menangani operasi yang relevan dengan fungsinya.

Penerapan ISP ini memastikan bahwa sistem lebih fleksibel, mudah dikembangkan, dan tidak memaksa kelas untuk memiliki metode yang tidak relevan dengan tugasnya.

7.7 Dependency Inversion Principle (DIP)

Dependency Inversion Principle (DIP) menyatakan bahwa modul tingkat tinggi tidak boleh bergantung pada modul tingkat rendah, tetapi keduanya harus bergantung pada abstraksi. Artinya, kita harus memisahkan ketergantungan antara kelas dengan menggunakan antarmuka atau abstraksi.

Dalam sistem ini, `PaymentProcessor` tidak langsung bergantung pada implementasi spesifik dari `CreditCardPayment`, `EWalletPayment`, atau `BankTransferPayment`. Sebaliknya, `PaymentProcessor` hanya mengetahui bahwa ia berinteraksi dengan objek bertipe `Payment`.

Dengan menggunakan antarmuka `Payment`, kita bisa mengganti atau menambah metode pembayaran baru tanpa harus mengubah kode dalam `PaymentProcessor`. Jika `PaymentProcessor` langsung menggunakan kelas `CreditCardPayment`, maka setiap kali kita ingin menambahkan metode pembayaran baru, kita harus mengubah kode `PaymentProcessor`, yang bertentangan dengan prinsip OCP.

Dengan mengikuti prinsip DIP, sistem ini menjadi **modular, fleksibel, dan lebih mudah diuji**, karena kita dapat dengan mudah mengganti implementasi tanpa harus mengubah kelas yang bergantung padanya.



Modul 8 COTS Penerapan Tubes 1

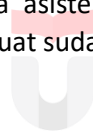
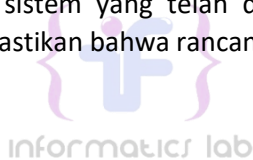
Tujuan Praktikum
<ol style="list-style-type: none">1. Menerapkan desain arsitektur atau sistem yang telah dibuat dalam bentuk aplikasi GUI.2. Mengembangkan aplikasi dengan menerapkan prinsip SOLID agar kode lebih modular dan mudah dikembangkan.

8.1 COTS Penerapan Tugas Besar 1

Pada Modul 8 ini, mahasiswa diharapkan sudah memiliki pemahaman yang jelas mengenai aplikasi yang akan dikembangkan, termasuk tujuan dan fungsionalitas utamanya. Selain itu, mahasiswa juga diharapkan telah menyusun desain awal arsitektur atau sistem sebagai dasar dalam perancangan aplikasi.

Selanjutnya, desain tersebut akan diimplementasikan dalam bentuk aplikasi dengan antarmuka grafis pengguna (GUI) yang interaktif dan sesuai dengan kebutuhan pengguna. Dalam proses pengembangannya, mahasiswa harus menerapkan prinsip SOLID untuk memastikan bahwa kode yang ditulis bersifat modular, mudah dikembangkan, serta mengikuti praktik terbaik dalam rekayasa perangkat lunak.

Mahasiswa juga diharapkan dapat menjelaskan dan mempresentasikan perancangan arsitektur atau sistem yang telah dibuat kepada asisten praktikum, guna mendapatkan masukan serta memastikan bahwa rancangan yang dibuat sudah sesuai dengan konsep yang diharapkan.



School of Computing
Telkom University

Modul 9 SonarQube

Tujuan Praktikum
1. Mengetahui konsep analisis kualitas kode dalam pengembangan perangkat lunak.
2. Menggunakan SonarQube untuk menganalisis dan memperbaiki kode program.

9.1 Pendahuluan

SonarQube, adalah alat tinjauan kode otomatis yang dikelola sendiri yang secara sistematis membantu Anda memberikan kode yang bersih. Sebagai elemen inti dari Sonar solution, SonarQube terintegrasi ke dalam alur kerja Anda yang ada dan mendeteksi masalah dalam kode Anda untuk membantu Anda melakukan pemeriksaan kode berkelanjutan pada proyek Anda. Alat ini menganalisis 30+ bahasa pemrograman yang berbeda dan terintegrasi ke dalam CI pipeline dan DevOps platform Anda untuk memastikan bahwa kode Anda memenuhi standar kualitas tinggi.

Solusi Sonar melakukan pemeriksaan pada setiap tahap proses pengembangan:

- SonarLint memberikan umpan balik langsung di IDE Anda saat Anda menulis kode sehingga Anda dapat menemukan dan memperbaiki masalah sebelum komit.
- PR analysis SonarQube cocok dengan alur kerja CI/CD Anda dengan analisis PR SonarQube & penggunaan Gerbang Kualitas.
- Quality Gates menjaga kode dengan masalah agar tidak dirilis ke produksi, alat utama dalam membantu Anda memasukkan metodologi Clean as You Code.
- Pendekatan Clean as You Code membantu Anda fokus dalam mengirimkan kode yang bersih baru untuk rilis, dengan mengetahui bahwa kode yang ada akan ditingkatkan seiring berjalannya waktu.

Quality Gate adalah indikator kualitas kode yang dapat dikonfigurasi untuk memberikan sinyal go/no-go pada kelayakan rilis kode saat ini. Ini menunjukkan apakah kode Anda bersih dan dapat bergerak maju.

- Gerbang Kualitas yang lewat (hijau) berarti kode tersebut memenuhi standar Anda dan siap untuk digabungkan.
- Gerbang Kualitas (merah) yang gagal berarti ada masalah yang harus ditangani

9.2 Prerequisite

Anda harus dapat menginstal Java (Oracle JRE atau OpenJDK) pada mesin tempat Anda berencana menjalankan SonarQube.

9.3 Hardware Requirement

1. Instance skala kecil (individu atau tim kecil) dari server SonarQube membutuhkan setidaknya 2GB RAM untuk berjalan secara efisien dan 1GB RAM gratis untuk OS. Jika Anda memasang instans untuk tim besar atau Perusahaan, harap pertimbangkan rekomendasi tambahan di bawah ini.

2. Jumlah ruang disk yang Anda butuhkan akan bergantung pada seberapa banyak kode yang Anda analisis dengan SonarQube.
3. SonarQube harus diinstal pada hard drive yang memiliki kinerja baca & tulis yang sangat baik. Yang paling penting, folder "data" menyimpan indeks Elasticsearch dimana sejumlah besar I/O akan dilakukan saat server aktif dan berjalan. Oleh karena itu, kinerja hard drive baca & tulis yang hebat akan berdampak besar pada kinerja server SonarQube secara keseluruhan.
4. SonarQube tidak mendukung sistem 32-bit di sisi server. Namun, SonarQube mendukung sistem 32-bit di sisi pemindai.

9.4 Instalasi SonarQube

1. Cek versi Java dengan menjalankan 'java --version' pada terminal anda. Jika versi java bukan versi 11 atau 17, maka unduh dan download pada JDK versi 11 atau 17 di perangkat anda pada link <https://www.oracle.com/id/java/technologies/javase/jdk11-archive-downloads.html>

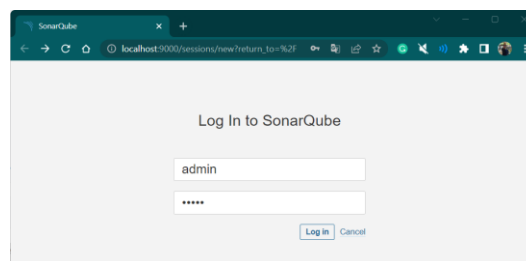
```
C:\Users\>java --version
java 11 2018-09-25
Java(TM) SE Runtime Environment 18.9 (build 11+28)
Java HotSpot(TM) 64-Bit Server VM 18.9 (build 11+28, mixed mode)
```

Gambar 9-1 Cek Versi Java

2. Unduh file zip instalasi SonarQube via <https://www.sonarqube.org/downloads/>
3. Ekstrak file zip dan masuk ke dalam folder. Contoh pada terminal windows:

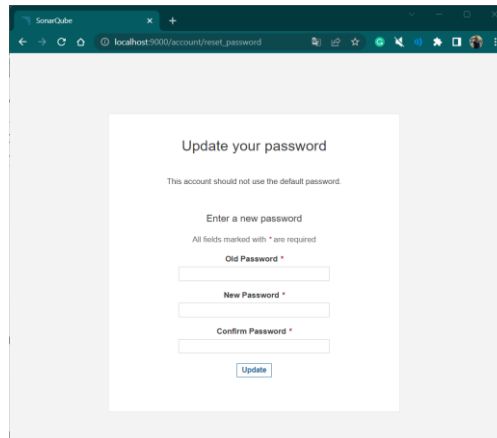
```
C:\sonarqube\bin\windows-x86-64\StartSonar.bat
```

4. Setelah instansi Anda aktif dan berjalan, Masuk ke <http://localhost:9000> menggunakan kredensial Administrator Sistem:
 - login: admin
 - kata sandi: admin



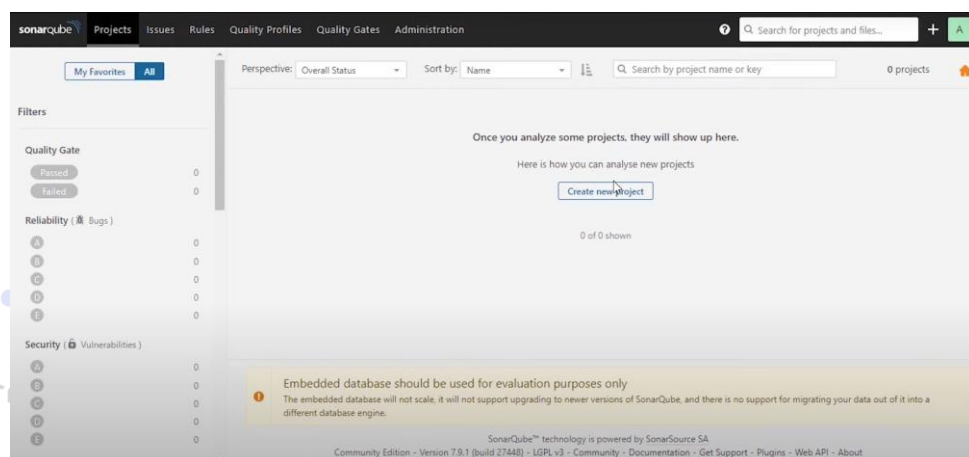
Gambar 9-2 Landing Page SonarQube

5. Setelah login anda diminta untuk mengubah sandi lama anda. Isi sandi lama dan sandi baru anda, lalu klik Update



Gambar 9-3 Halaman Update Password

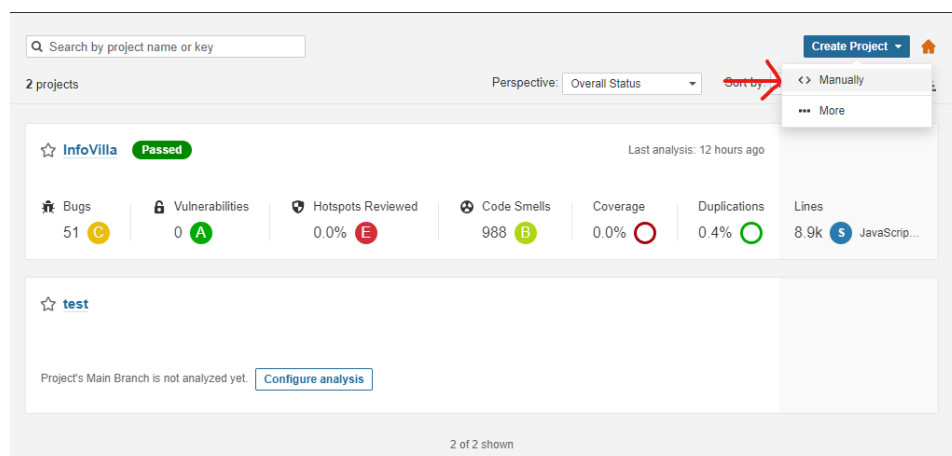
6. Selanjutnya akan tampil halaman utama dari sonarqube



Gambar 9-4 Halaman Utama SonarQube

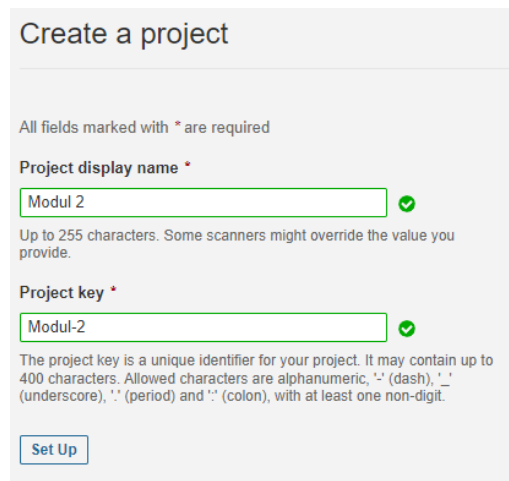
9.5 Membuat dan Menganalisis Project

1. Klik tombol **Create new project** dan klik **Manually**.



Gambar 9-5 Inisialisasi Proyek Baru 1

2. Berikan proyek Anda **Display name** dan **Project key** dan klik tombol **Set Up**.



Create a project

All fields marked with * are required

Project display name *

Modul 2 ✓

Up to 255 characters. Some scanners might override the value you provide.

Project key *

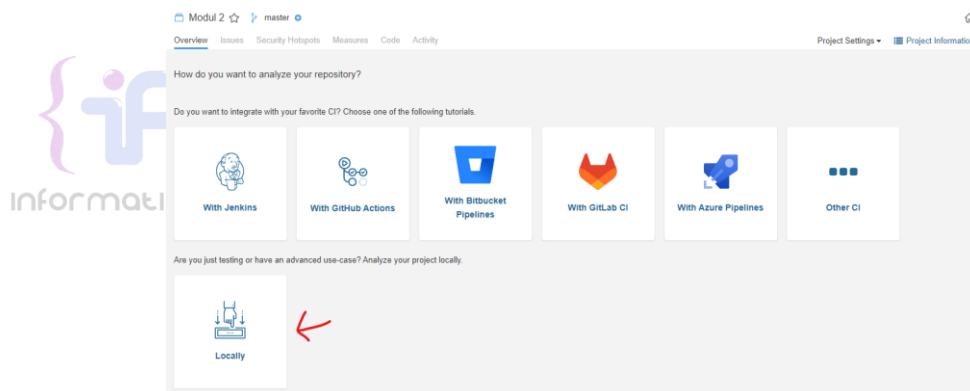
Modul-2 ✓

The project key is a unique identifier for your project. It may contain up to 400 characters. Allowed characters are alphanumeric, '-' (dash), '_' (underscore), '.' (period) and ':' (colon), with at least one non-digit.

Set Up

Gambar 9-6 Inisialisasi Proyek Baru 2

3. Setelah Project berhasil di buat akan tampil beberapa opsi untuk integrasi dengan CI dan analyze pada local. Untuk melakukan analisis project local dapat memilih opsi **Locally**.



Gambar 9-7 Menggunakan proyek lokal

4. Di bawah **Provide a token**, pilih **Generate a token**. Beri nama token Anda, klik tombol **Generate**, dan klik **Continue**.

1 Provide a token

☒ Generate a project token

Token name ? Expires in

Analyze "Modul 2" 30 days Generate

☐ Use existing token

The token is used to identify you when an analysis is performed. If it has been compromised, you can revoke it at any point in time in your [user account](#).

1 Provide a token

Analyze "Modul 2": sqp_178e5171f3189c9c8a6dcc5c040fae93a9ff7e55 🗑

The token is used to identify you when an analysis is performed. If it has been compromised, you can revoke it at any point in time in your [user account](#).

Continue

Gambar 9-8 Generate Token

5. Pilih bahasa utama proyek Anda di bawah **Run analysis on your project**, dan ikuti petunjuk untuk menganalisis proyek Anda. Di sini Anda akan mengunduh dan menjalankan Pemindai pada kode Anda (jika Anda menggunakan Maven atau Gradle, Pemindai diunduh secara otomatis).
6. Untuk bahasa pemrograman selain menggunakan Maven, Gradle (Contoh : JS, Python, Go, dll), anda diharuskan mendownload **SonarScanner** terlebih dahulu melalui <https://docs.sonarqube.org/latest/analysis/scan/sonarscanner/>.
7. Download scanner sesuai dengan OS yang digunakan.

infor **SonarScanner**

By [SonarSource](#) | GNU LGPL 3 | [Issue Tracker](#)

4.7 Show more versions

2022-02-22

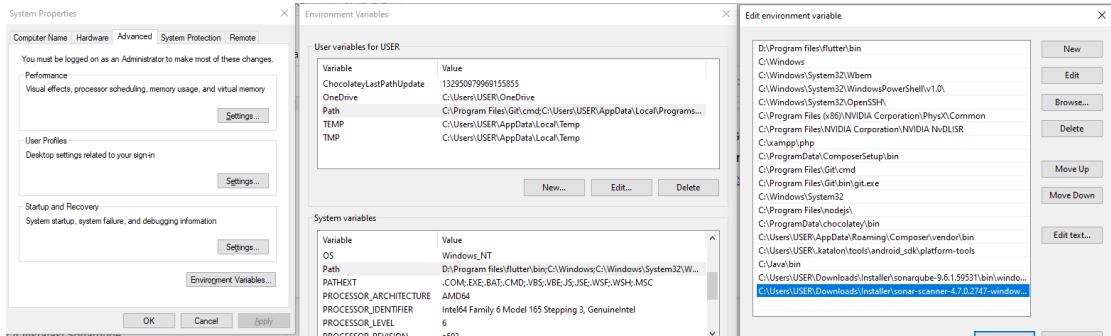
Ease import of custom certificates with the Docker image, update embedded JRE 11

[Linux 64-bit](#) [Windows 64-bit](#) [Mac OS X 64-bit](#) [Docker](#)

[Any \(Requires a pre-installed JVM\)](#) [Release notes](#)

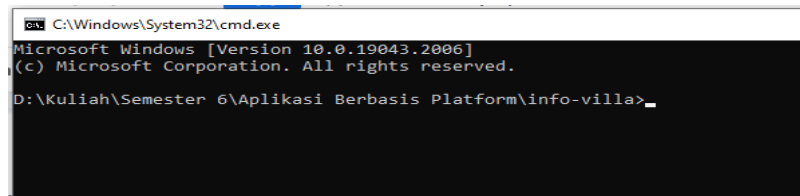
Gambar 9-9 Download SonarScanner

8. Extract file dan taruh alamat scanner pada %PATH%.



Gambar 9-10 Edit Environment Variables

9. Buka **Command Prompt** dengan alamat mengarah ke proyek yang akan dianalisa.

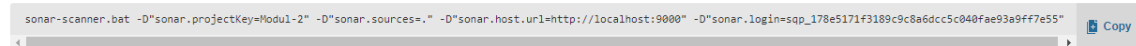


Gambar 9-11 Buka CMD

10. Copy script scanner (sesuaikan dengan pilihan OS) yang telah diberikan oleh sonarqube dan Jalankan pada command prompt

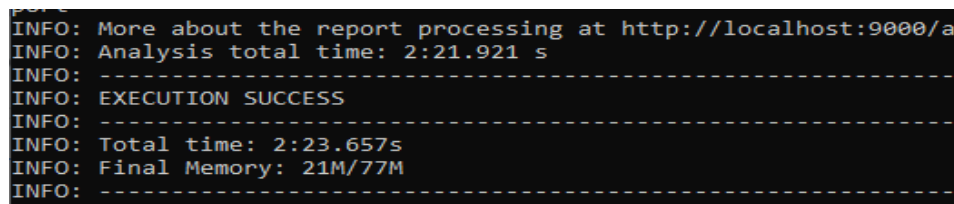
Execute the Scanner

Running a SonarQube analysis is straightforward. You just need to execute the following commands in your project's folder.



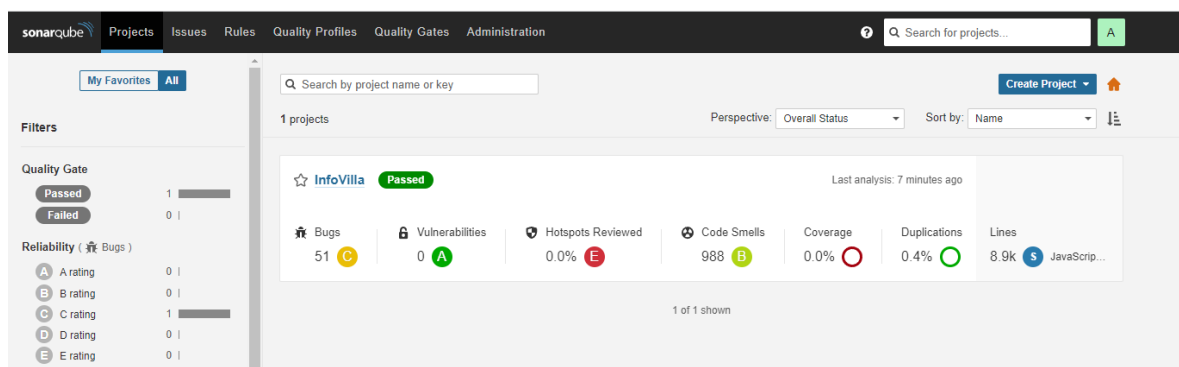
Gambar 9-12 Copy Script Scanner

11. Apabila proses analisis sudah selesai, anda dapat melihat hasilnya pada sonarqube.



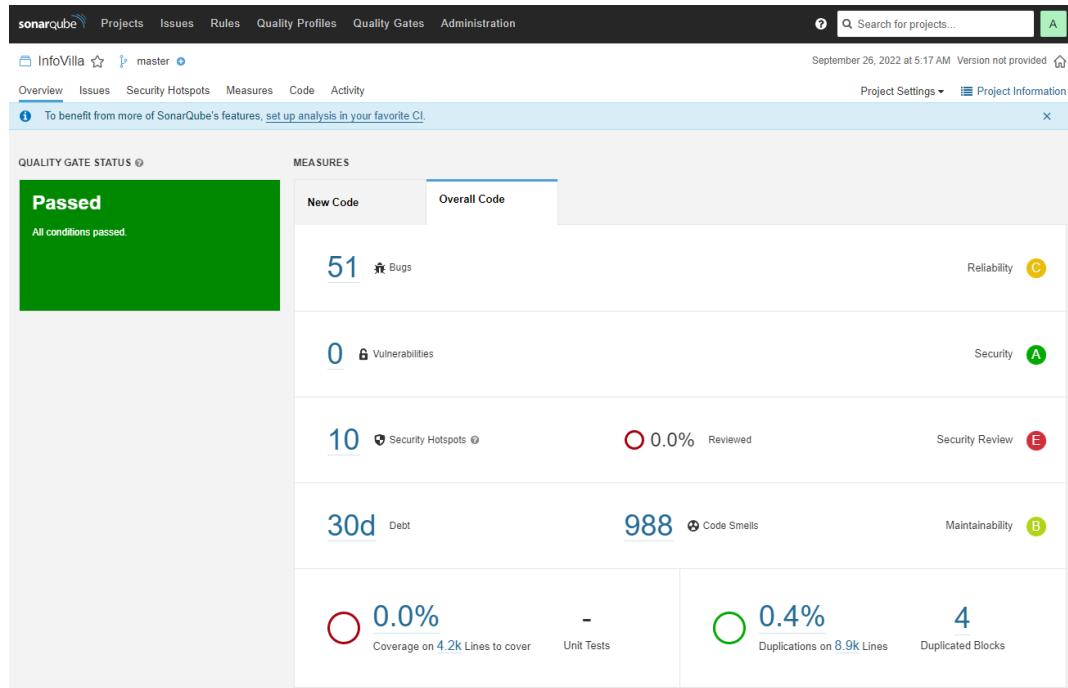
Gambar 9-13 Proses Analisis Berhasil

Setelah melakukan analyze project, maka akan tampil informasi seperti gambar berikut.



Gambar 9-14 Halaman Utama dengan Proyek Berhasil di Analisis

Untuk melihat detail dari hasil analisis, tekan nama project yang telah dianalisis. Maka akan tampil halaman seperti gambar dibawah ini.



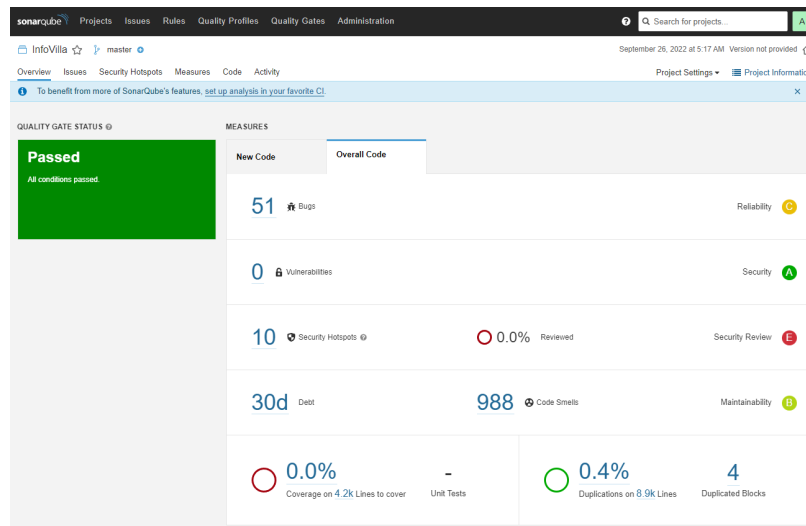
Gambar 9-15 Ringkasan Hasil Analisis 1

Halaman ini menampilkan hasil analisis project secara detail. Ada dua tab utama pada halaman ini yaitu **New Code** dan **Overall Code**. **New Code** akan menampilkan hasil hanya untuk perubahan kode terbaru. Apabila kode dirubah dan sonarqube dijalankan lagi maka **New Code** hanya menampilkan hasil analisis pada kode yang telah diubah sedangkan **Overall Code** menampilkan hasil analisis secara keseluruhan. Berikut adalah penjelasan informasi-informasi yang tertera pada gambar diatas terdiri dari:

Informasi	Keterangan
Bugs	Masalah yang mewakili sesuatu yang salah dalam kode yang dapat menyebabkan error.
Vulnerabilities	Masalah terkait keamanan yang mewakili pintu belakang bagi penyerang (hacker).
Security Hotspot	blok kode di mana keamanan menjadi perhatian.
Debt	Perkiraan waktu yang diperlukan untuk memperbaiki semua Masalah Pemeliharaan / Code Smells.
Code Smells	Masalah pemeliharaan yang membuat kode Anda membingungkan dan sulit untuk di maintenance.
Coverage	Persentase kode yang dicakup oleh pengujian otomatis.
Duplications	Duplikasi mengacu pada pengulangan kode dalam aplikasi.

9.6 Project Tabs

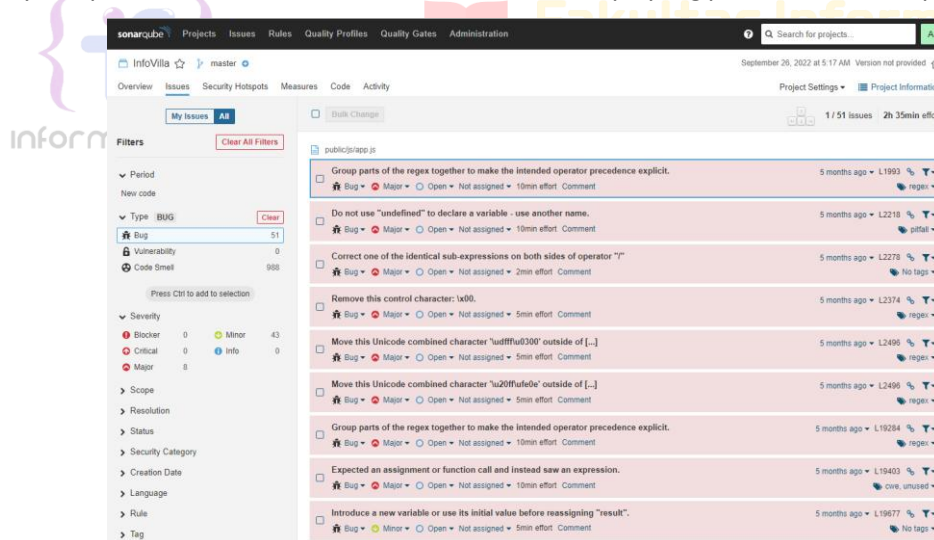
9.6.1 Overview Tab



Gambar 9-16 Ringkasan Hasil Analisis 2

9.6.2 Issues Tab

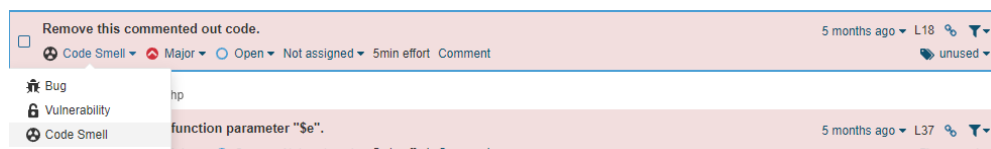
Issues tab memberi kekuatan penuh untuk menganalisis secara rinci apa masalah utama, di mana lokasinya, kapan ditambahkan ke basis kode Anda dan siapa yang pertama kali memperkenalkannya.



Gambar 9-17 Issues Tab 1

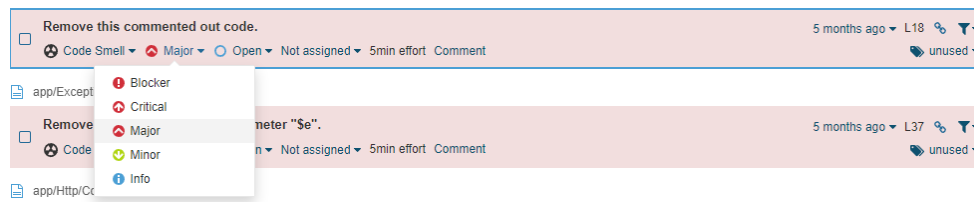
Setiap Issue memiliki beberapa option. Anda dapat melakukan beberapa action pada setiap option tersebut seperti:

1. Anda dapat merubah tags pada issue.



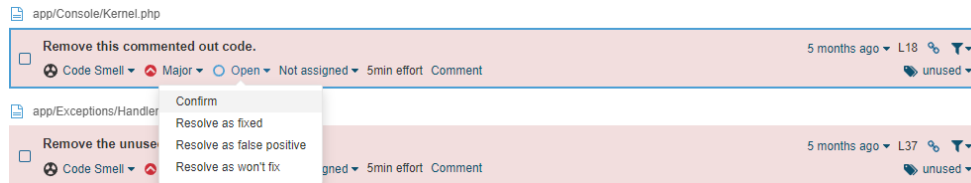
Gambar 9-18 Issues Tab 2

2. Anda dapat merubah tingkatan severity pada issue



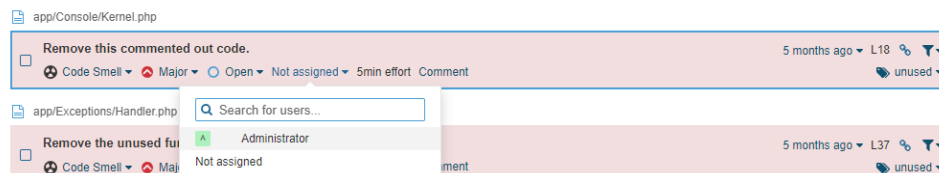
Gambar 9-19 Issues Tab 3

3. Anda dapat merubah status pada issue.



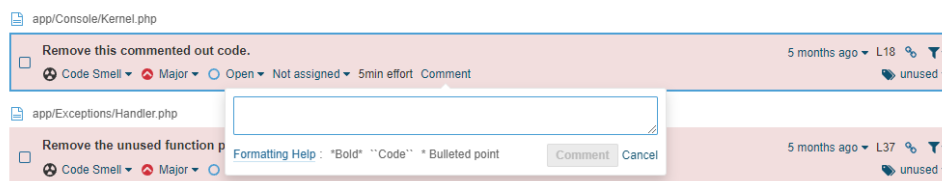
Gambar 9-20 Issues Tab 4

4. Anda dapat menetapkan atau assign issue ini kepada user lainnya.



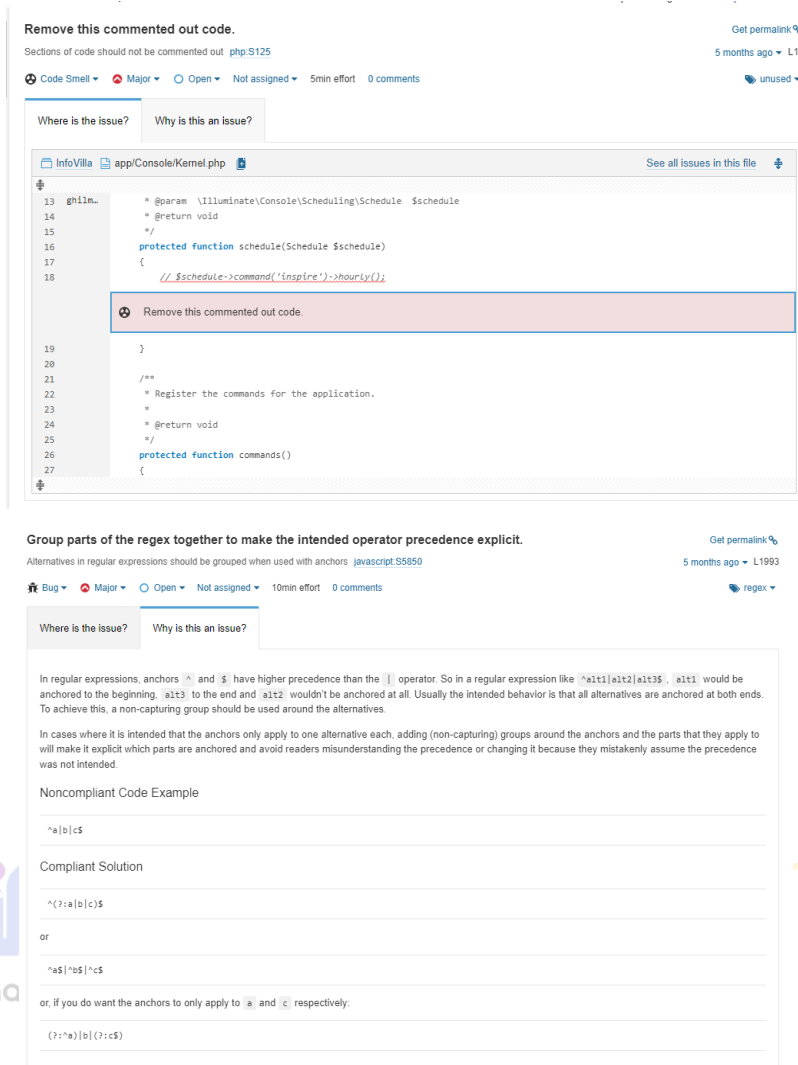
Gambar 9-21 Issues Tab 5

5. Anda dapat memberikan komentar pada issue.



Gambar 9-22 Issues Tab 6

Untuk melihat detail dari masing masing issue, anda dapat menekan issue tersebut untuk melihat baris kode yang ditetapkan memiliki permasalahan oleh sonarqube seperti gambar di bawah ini.

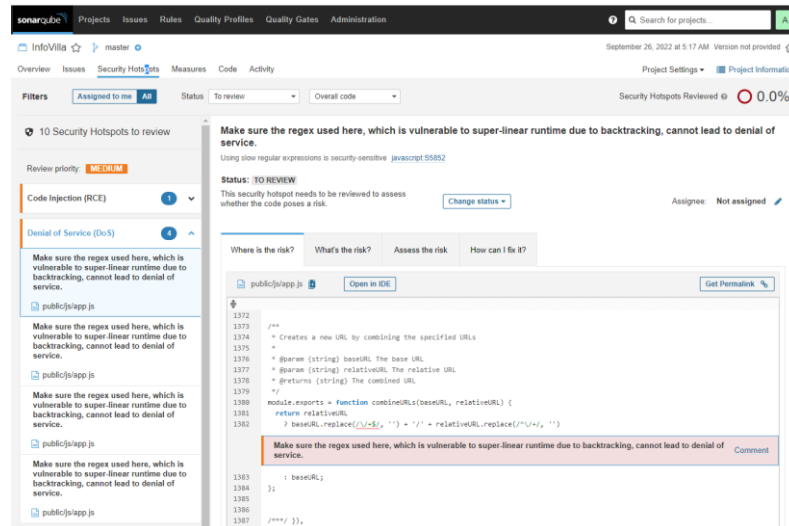


Gambar 9-23 Issues Tab 7

Pada halaman ini, Tab **Where is the issue?** menampilkan baris kode beserta issue-nya Lalu pada Tab **Why is this an issue?** menampilkan penjelasan serta alasan mengapa baris kode tersebut ditetapkan sebagai issue.

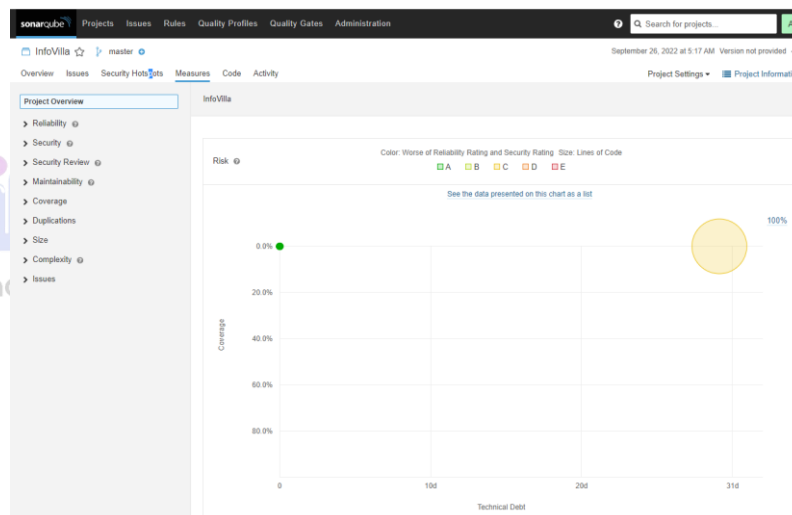
9.6.3 Security Hotspot Tab

Security Hotspot menyoroti bagian kode yang sensitif terhadap keamanan yang perlu ditinjau oleh pengembang. Setelah peninjauan, Anda akan menemukan tidak ada ancaman pada kode atau Anda perlu menerapkan perbaikan untuk mengamankan kode. Security Hotspot membantu memfokuskan upaya pengembang yang secara manual memeriksa kode sensitif keamanan.



9.6.4 Measures Tab

Measures tab memungkinkan untuk mendalami Quality Matrix SonarQube.

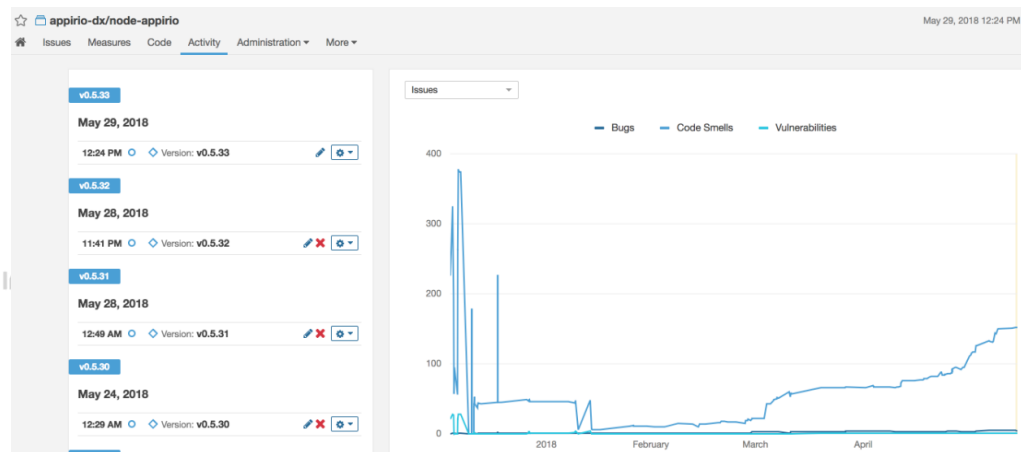


	Lines of Code	Bugs	Vulnerabilities	Code Smells	Security Hotspots	Coverage	Duplications
InfoVilla							
└─ app	493	0	0	8	0	0.0%	0.0%
└─ bootstrap	16	0	0	0	0	0.0%	0.0%
└─ config	514	0	0	2	1	0.0%	0.0%
└─ database	159	0	0	1	0	0.0%	0.0%
└─ lang/en	140	0	0	1	0	0.0%	45.2%
└─ public	7,478	9	0	971	5	0.0%	0.0%
└─ resources	4	42	0	4	4	0.0%	0.0%
└─ routes	33	0	0	0	0	0.0%	0.0%
└─ tests	37	0	0	0	0	0.0%	0.0%
└─ phpunit.xml	29	0	0	1	0	—	0.0%
└─ webpack.mix.js	4	0	0	0	0	0.0%	0.0%

Gambar 9-26 Code Tab

9.6.6 Activity Tab

Activity tab memberikan detail tentang aktivitas pemindaian proyek



Gambar 9-27 Activity Tab

Modul 10 REST API Spring Boot

Tujuan Praktikum

1. Mengetahui konsep REST API dan cara kerjanya.
2. Menggunakan Spring Boot untuk membangun REST API.
3. Menggunakan HTTP Methods (GET, POST, PUT, DELETE) dalam REST API.

10.1 Pengenalan API

10.1.1 Apa itu API?

Application Programming Interface (API) adalah sebuah antarmuka komputasi yang mendefinisikan bagaimana sistem atau komponen perangkat lunak berinteraksi satu sama lain. API mendefinisikan jenis **permintaan (request)** yang dapat dilakukan, bagaimana membuatnya, format data yang harus digunakan, serta konvensi yang harus diikuti dalam penggunaannya.

API memungkinkan sistem untuk berkomunikasi secara modular tanpa perlu mengetahui implementasi internalnya. Dengan API, pengembang dapat fokus pada pemanfaatan fitur yang disediakan tanpa harus memahami cara kerja detail dari sistem yang mereka akses.

10.1.2 Fungsi dan Manfaat API?

API memiliki berbagai manfaat dalam pengembangan perangkat lunak, di antaranya:

1. **Abstraksi**
API menyembunyikan detail implementasi dan hanya mengekspos fungsionalitas yang diperlukan pengguna.
2. **Interoperabilitas**
Memungkinkan komunikasi antara berbagai sistem atau aplikasi yang menggunakan teknologi yang berbeda.
3. **Efisiensi Pengembangan**
Mengurangi waktu dan usaha pengembang dengan menyediakan fitur yang dapat digunakan kembali, sehingga mereka tidak perlu membangun fitur dari awal.
4. **Keamanan**
API bertindak sebagai lapisan kontrol akses, memungkinkan sistem untuk membatasi hak akses ke sumber daya tertentu.
5. **Ekstensibilitas**
API dapat diperluas dengan mudah tanpa perlu mengubah seluruh sistem, sehingga mendukung skalabilitas.

10.1.3 Prinsip Desain API yang Baik

Dalam merancang API, ada beberapa pertanyaan penting yang harus dijawab untuk memastikan API yang dibuat dapat digunakan dengan baik:

1. **Mengapa API ini dibuat?**
Apakah digunakan untuk integrasi internal atau dibuka untuk pengembang eksternal?
2. **Apa tujuan utama dari API ini?**

Apakah API dibuat untuk memungkinkan akses ke data, menyediakan layanan, atau memungkinkan pengguna untuk melakukan tindakan tertentu?

3. **Bagaimana cara terbaik untuk mendesain API ini?**

Teknologi apa yang digunakan? Apakah API akan menggunakan REST, GraphQL, atau teknologi lain? Bagaimana struktur endpoint-nya?

Dengan menjawab pertanyaan di atas, pengembang dapat merancang API yang lebih **fleksibel, aman, dan mudah digunakan**.

10.2 REST API

10.2.1 Apa itu REST API?

REST API (Representational State Transfer API) adalah sebuah antarmuka pemrograman aplikasi (API) yang mematuhi prinsip desain REST (Representational State Transfer). REST API digunakan



untuk menghubungkan dan mengintegrasikan berbagai sistem dengan cara yang ringan, fleksibel, dan efisien.

REST API menjadi standar utama dalam pengembangan **web service** modern karena memungkinkan komunikasi antar sistem melalui **protokol HTTP** dengan format data yang sederhana seperti **JSON** dan **XML**.

REST pertama kali diperkenalkan oleh **Dr. Roy Fielding** dalam disertasinya pada tahun 2000. REST API menjadi arsitektur utama dalam pengembangan **sistem berbasis microservices**, karena mendukung pemisahan antar layanan secara modular dan independen.

10.2.2 Prinsip Desain REST API

REST API memiliki enam prinsip utama yang dikenal sebagai **RESTful architectural constraints**:

1. **Uniform Interface**

Semua permintaan API terhadap sumber daya harus memiliki format yang seragam dan menggunakan Uniform Resource Identifier (URI) yang unik.

2. **Client-server Decoupling**

REST API dirancang agar klien dan server bersifat independen. Klien hanya perlu mengetahui URI sumber daya, sementara server hanya bertanggung jawab mengelola data dan tidak mengontrol antarmuka pengguna.

3. **Statelessness**

Setiap permintaan API bersifat stateless, artinya server tidak menyimpan status sesi klien. Semua Informasi yang diperlukan untuk memproses permintaan dikirim dalam setiap request.

4. **Cacheability**

Data dalam respons API dapat di-cache oleh klien atau server untuk meningkatkan efisiensi dan performa. Server perlu menentukan apakah respons API boleh disimpan atau tidak.

5. **Layered System Architecture**

REST API memungkinkan adanya beberapa lapisan antara klien dan server, seperti gateway API, load balancer, atau sistem keamanan, tanpa memengaruhi komunikasi antara klien dan server utama.

6. **Code on Demand**

Dalam beberapa kasus, REST API dapat mengirimkan kode eksekusi (misalnya JavaScript) ke klien, namun prinsip ini jarang digunakan.

10.2.3 Bagaimana REST API bekerja?

REST API bekerja dengan mengirimkan **permintaan (request)** dari klien ke server menggunakan protokol **HTTP**. Setiap permintaan berisi:

- **Metode HTTP**: Menentukan operasi yang dilakukan (GET, POST, PUT, DELETE).
- **URI**: Lokasi sumber daya yang diminta (/products/1).
- **Header**: Berisi informasi tambahan seperti autentikasi atau format data (Content-Type: application/json).

- **Body** (Opsional): Digunakan dalam metode seperti POST atau PUT untuk mengirim data ke server.

Contoh permintaan **GET** untuk mengambil data produk:

```
GET /products/1 HTTP/1.1
Host: api.example.com
Accept: application/json
Authorization: Bearer token123
```

Server akan mengembalikan respons dalam format JSON seperti berikut:

```
{
  "id": 1,
  "name": "Laptop",
  "price": 15000000
}
```

10.2.4 Metode HTTP dalam REST API

Metode	Deskripsi
GET	Mengambil data dari server.
POST	Menambahkan data baru ke server.
PUT	Memperbarui data yang sudah ada.
DELETE	Menghapus data dari server.

10.2.5 Keuntungan dan Tantangan REST API

Keuntungan REST API

- **Scalability** → REST API mendukung arsitektur terdistribusi dan mudah diskalakan.
- **Portability** → Dapat diakses dari berbagai platform seperti web, mobile, IoT.
- **Independence** → Klien dan server bisa dikembangkan secara independen.
- **Lightweight** → Menggunakan format data ringan seperti **JSON** untuk komunikasi yang cepat.

Tantangan REST API

- **Keamanan** → API harus dilengkapi dengan mekanisme autentikasi dan enkripsi.
- **Versi API** → Jika API diperbarui, versi lama harus tetap didukung agar tidak merusak sistem yang telah terintegrasi.
- **Konsistensi Endpoint** → URL endpoint harus konsisten agar mudah dipahami oleh pengembang lain.

10.3 REST API Spring Boot di Eclipse

Spring Framework adalah kerangka kerja untuk pengembangan aplikasi berbasis Java yang menyediakan model pemrograman dan konfigurasi yang fleksibel. Framework ini berfokus pada "plumbing" atau infrastruktur aplikasi, memungkinkan pengembang untuk lebih fokus pada logika bisnis tanpa bergantung pada lingkungan deployment tertentu.

Spring Boot adalah ekstensi dari Spring Framework yang menyederhanakan pengembangan aplikasi Java dengan konfigurasi minimal. Spring Boot memungkinkan pembuatan aplikasi mandiri (stand-alone) dan siap produksi (production-grade) tanpa perlu konfigurasi manual yang rumit. Dengan fitur seperti auto-configuration dan embedded server, Spring Boot memungkinkan aplikasi untuk langsung dijalankan tanpa banyak pengaturan tambahan.

Untuk membuat REST API dengan Spring Boot di Eclipse, langkah pertama yang harus dilakukan adalah membuat proyek menggunakan Spring Initializr. Spring Initializr adalah alat yang digunakan untuk menghasilkan proyek Spring Boot dengan konfigurasi awal.

Berikut adalah panduan langkah demi langkah untuk membuat proyek Spring Boot di Eclipse menggunakan Spring Initializr:

1. Buka Spring Initializr

Terdapat dua cara untuk mengakses **Spring Initializr**:

- **Opsi 1: Menggunakan Browser**
 - Buka [Spring Initializr](#) di browser.
 - Konfigurasi proyek seperti yang dijelaskan pada langkah berikutnya.
- **Opsi 2: Menggunakan Eclipse**
 - Buka Eclipse.
 - Pilih **File > New > Spring Starter Project**.
 - Eclipse akan menampilkan antarmuka **Spring Initializr** yang sudah terintegrasi.

2. Konfigurasi Proyek Spring Boot

Setelah membuka Spring Initializr, atur konfigurasi proyek sebagai berikut:

- **Project:** Pilih Maven (atau Gradle jika menggunakan Gradle).
- **Language:** Pilih Java.
- **Spring Boot Version:** Pilih versi terbaru yang **stabil** (misalnya 3.5.0).
- **Group ID:** com.example (atau sesuai kebutuhan proyek).
- **Artifact ID:** rest-api (atau nama proyek yang diinginkan).
- **Name:** rest-api
- **Description:** Spring Boot REST API Example
- **Package Name:** com.example.restapi
- **Packaging:** Pilih Jar.
- **Java Version:** Pilih 23 (versi sesuai JDK yang diinstal).

3. Menambahkan Dependencies

Tambahkan **dependencies** yang diperlukan untuk proyek **REST API**:

- **Spring Web** → Untuk membuat REST API berbasis Spring.
- **Spring Boot DevTools** → Untuk mempercepat pengembangan dengan live reload.

- **Spring Data JPA** → Untuk menghubungkan ke database menggunakan JPA.
- **MySQL Driver** → Jika menggunakan MySQL sebagai database utama.

Jika menggunakan Spring Initializr di **browser**, klik **Generate** untuk mengunduh proyek dalam format .zip.

4. Mengimpor Proyek ke Eclipse

Jika menggunakan Spring Initializr dari browser, impor proyek ke Eclipse dengan langkah berikut:

1. **Ekstrak file .zip** yang sudah diunduh dari Spring Initializr.
2. Buka Eclipse dan pilih **File > Import > Existing Maven Projects**.
3. Klik **Next**, lalu cari folder proyek yang telah diekstrak.
4. Pilih proyek dan klik **Finish**.
5. Eclipse akan otomatis mendeteksi dan mengimpor proyek sebagai proyek **Spring Boot berbasis Maven**.

5. Struktur Proyek Spring Boot

Setelah proyek berhasil diimpor, berikut adalah **struktur direktori** dari proyek **Spring Boot REST API** yang telah dibuat:

```
rest-api
|
|— src
|   |
|   |— main
|       |
|       |— java/com/example/restapi
|           |
|           |— RestApiApplication.java (Main Class)
|           |— controller/ (Controller Layer)
|           |— service/ (Service Layer)
|           |— repository/ (Repository Layer)
|           |— model/ (Entity Model)
|       |— resources
|           |— application.properties (Konfigurasi Spring Boot)
|— pom.xml (File konfigurasi Maven)
|— mvnw
|— mvnw.cmd
|— .gitignore
|— README.md
```

Penjelasan Struktur Proyek:

- RestApiApplication.java → **Main class** untuk menjalankan aplikasi.
- controller/ → Berisi kelas yang menangani permintaan HTTP.
- service/ → Berisi logika bisnis aplikasi.
- repository/ → Berisi kode untuk mengakses database.

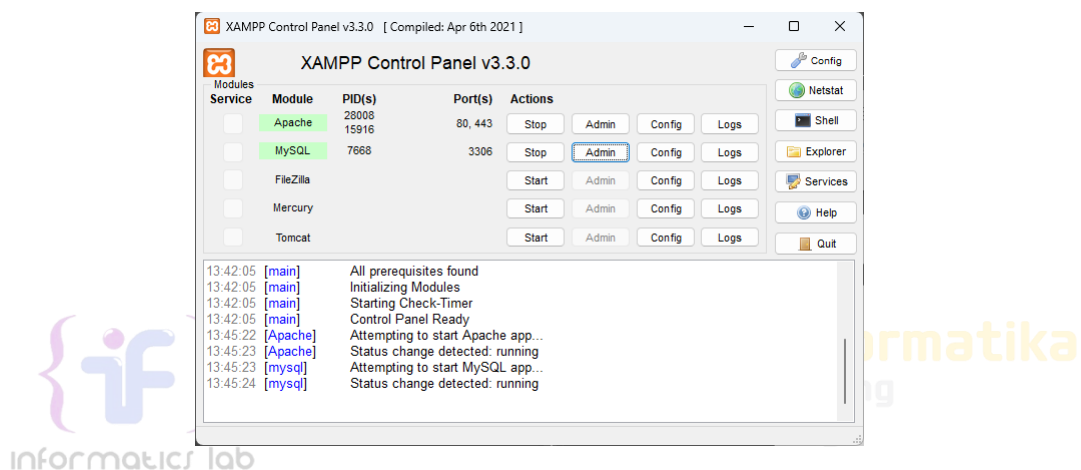
- model/ → Berisi definisi entitas JPA.
- application.properties → Berisi konfigurasi aplikasi, termasuk koneksi database.

6. Konfigurasi Koneksi Database

Tambahkan konfigurasi database di file `src/main/resources/application.properties`. Jika menggunakan MySQL, tambahkan konfigurasi sebagai berikut:

```
spring.datasource.url=jdbc:mysql://localhost:3306/penjualan
spring.datasource.username=root
spring.datasource.password=
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect
```

Selain itu, Anda juga harus menyiapkan XAMPP.

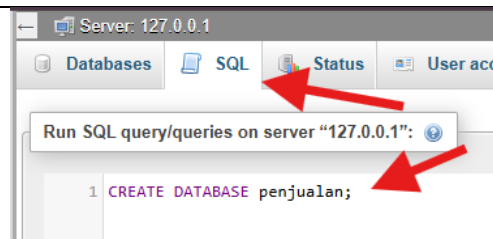


Gambar 10-1 Menu XAMPP

Silahkan buat Database baru dengan nama Penjualan (menyesuaikan konfigurasi) dan membuat tabel bernama product.

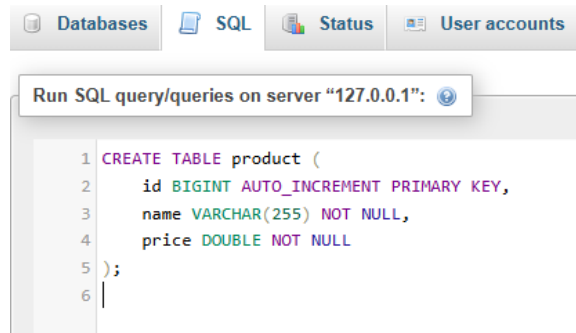
Gunakan Query berikut:

```
CREATE DATABASE penjualan;
```

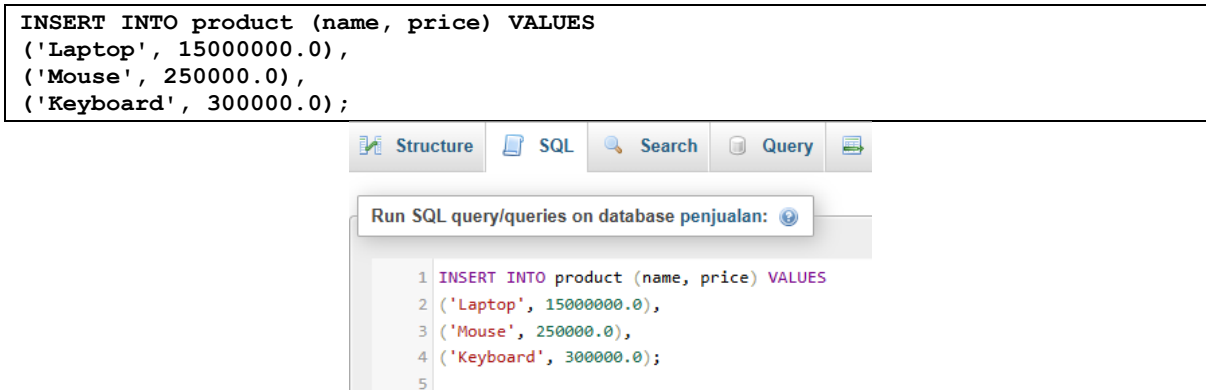


Gambar 10-2 Query Membuat Database

```
CREATE TABLE product (
  id BIGINT AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(255) NOT NULL,
  price DOUBLE NOT NULL
);
```



Gambar 10-3 Query Membuat Table Product



Gambar 10-4 Query Memasukkan Data ke dalam Table Product

7. Implementasi REST API Sederhana

Di langkah ini, kita akan membuat **CRUD sederhana** untuk entitas **Product** menggunakan **Spring Boot**.

1. Model Entity (Product.java)

```

package com.example.restapi.model;

import jakarta.persistence.*;

@Entity
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private double price;

    public Product() {}

    public Product(String name, double price) {
        this.name = name;
        this.price = price;
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }
}

```



```

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public double getPrice() {
        return price;
    }

    public void setPrice(double price) {
        this.price = price;
    }
}

```

2. Repository (ProductRepository.java)

```

package com.example.restapi.repository;

import com.example.restapi.model.Product;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface ProductRepository extends JpaRepository<Product, Long> {
}

```

3. Service Layer (ProductService.java)

```

package com.example.restapi.service;

import com.example.restapi.model.Product;
import com.example.restapi.repository.ProductRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.util.List;
import java.util.Optional;

@Service
public class ProductService {

    @Autowired
    private ProductRepository repository;

    public List<Product> getAllProducts() {
        return repository.findAll();
    }

    public Optional<Product> getProductById(Long id) {
        return repository.findById(id);
    }

    public Product saveProduct(Product product) {
        return repository.save(product);
    }

    public Product updateProduct(Long id, Product productDetails) {
        return repository.findById(id).map(product -> {
            product.setName(productDetails.getName());
            product.setPrice(productDetails.getPrice());
            return repository.save(product);
        }).orElse(null);
    }

    public void deleteProduct(Long id) {

```

```

        repository.deleteById(id);
    }
}

```

4. Controller Layer (ProductController.java)

```

package com.example.restapi.controller;

import com.example.restapi.model.Product;
import com.example.restapi.service.ProductService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.Optional;

@RestController
@RequestMapping("/products")
public class ProductController {

    @Autowired
    private ProductService service;

    @GetMapping
    public List<Product> getAll() {
        return service.getAllProducts();
    }

    @GetMapping("/{id}")
    public ResponseEntity<Product> getById(@PathVariable Long id) {
        Optional<Product> product = service.getProductById(id);
        return product.map(ResponseEntity::ok).orElseGet(() ->
        ResponseEntity.notFound().build());
    }

    @PostMapping
    public Product create(@RequestBody Product product) {
        return service.saveProduct(product);
    }

    @PutMapping("/{id}")
    public ResponseEntity<Product> update(@PathVariable Long id, @RequestBody
    Product productDetails) {
        Product updatedProduct = service.updateProduct(id, productDetails);
        return updatedProduct != null ? ResponseEntity.ok(updatedProduct) :
        ResponseEntity.notFound().build();
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<Void> delete(@PathVariable Long id) {
        service.deleteProduct(id);
        return ResponseEntity.noContent().build();
    }
}

```

8. Menjalankan Aplikasi Spring Boot

Setelah implementasi kode selesai, sekarang kita dapat menjalankan aplikasi Spring Boot.

Langkah-langkah menjalankan aplikasi:

1. Buka RestApiApplication

```

package com.example.restapi;

```

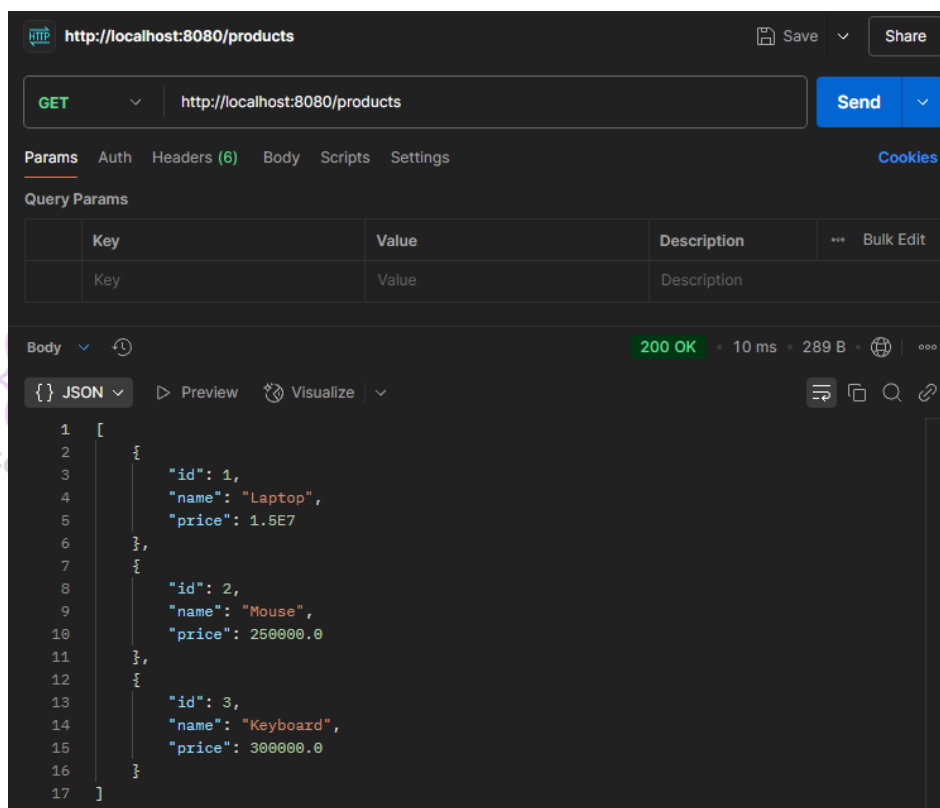
```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class RestApiApplication {
    public static void main(String[] args) {
        SpringApplication.run(RestApiApplication.class, args);
    }
}
```

2. Jalankan sebagai Spring Boot Application.
3. Jika berhasil, maka akan muncul output di Console:

```
Tomcat started on port(s): 8080 (http)
Started RestApiApplication in X.XX seconds
```

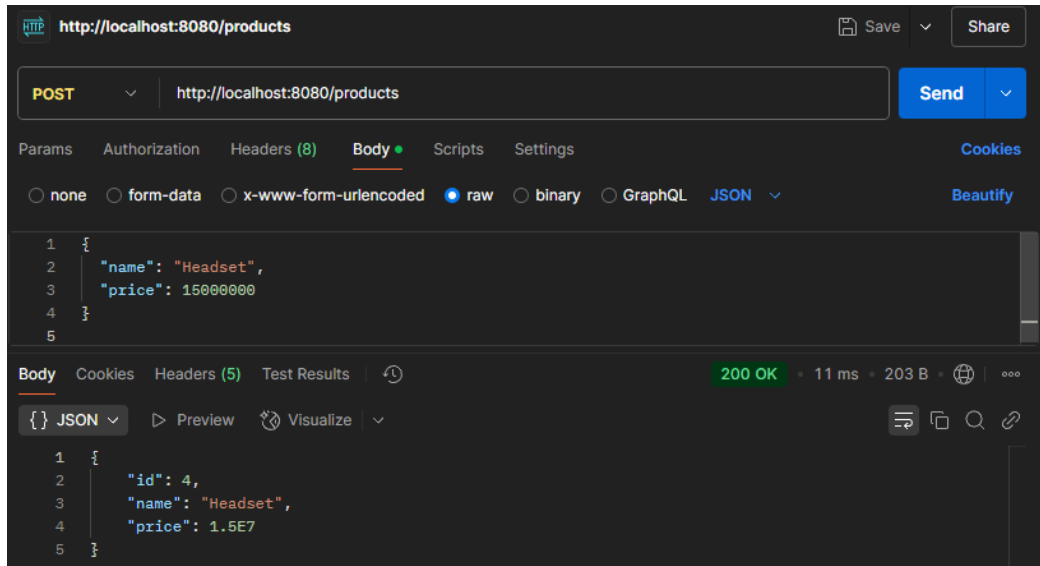
4. Uji API dengan Postman atau Browser:
 - GET <http://localhost:8080/products> → Menampilkan semua produk.



Gambar 10-5 Uji API GET

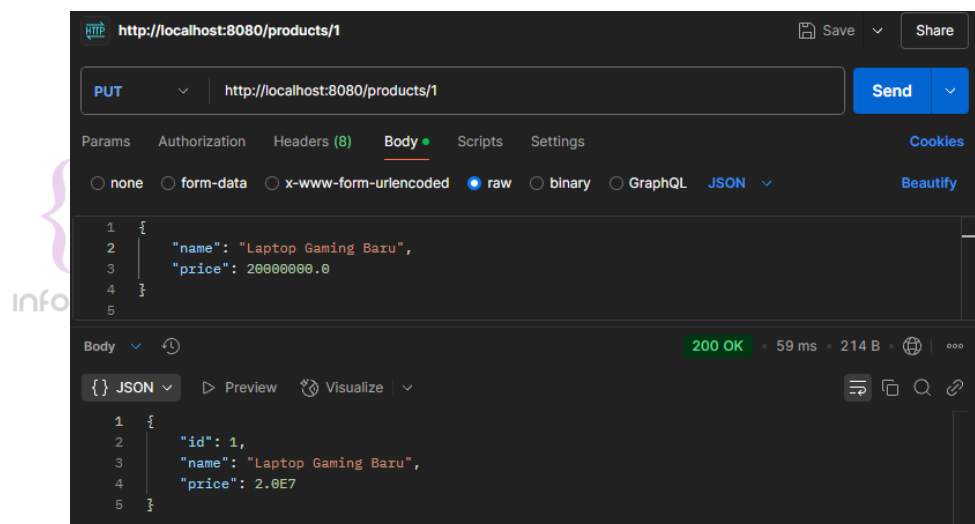
- POST <http://localhost:8080/products> dengan JSON:

```
{
  "name": "Headset",
  "price": 15000000
}
```



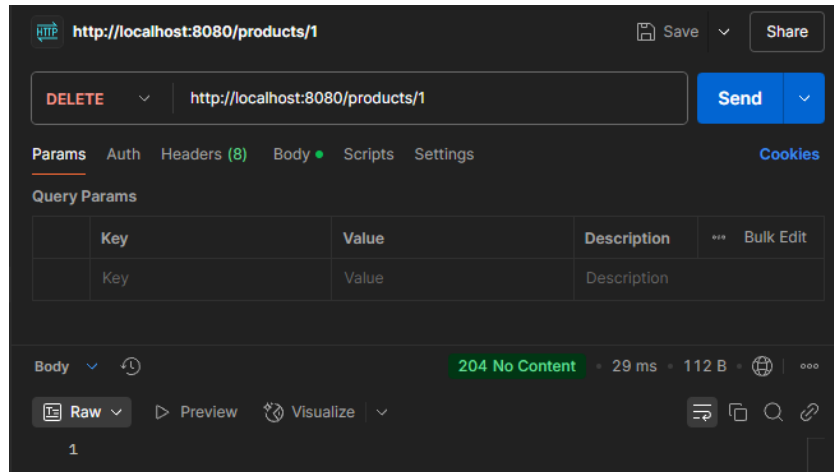
Gambar 10-6 Uji API POST

- `PUT http://localhost:8080/products/1` → Memperbarui produk dengan id 1.



Gambar 10-7 Uji API PUT

- `DELETE http://localhost:8080/products/1` → Menghapus produk dengan id 1.



Gambar 10-8 Uji API DELETE

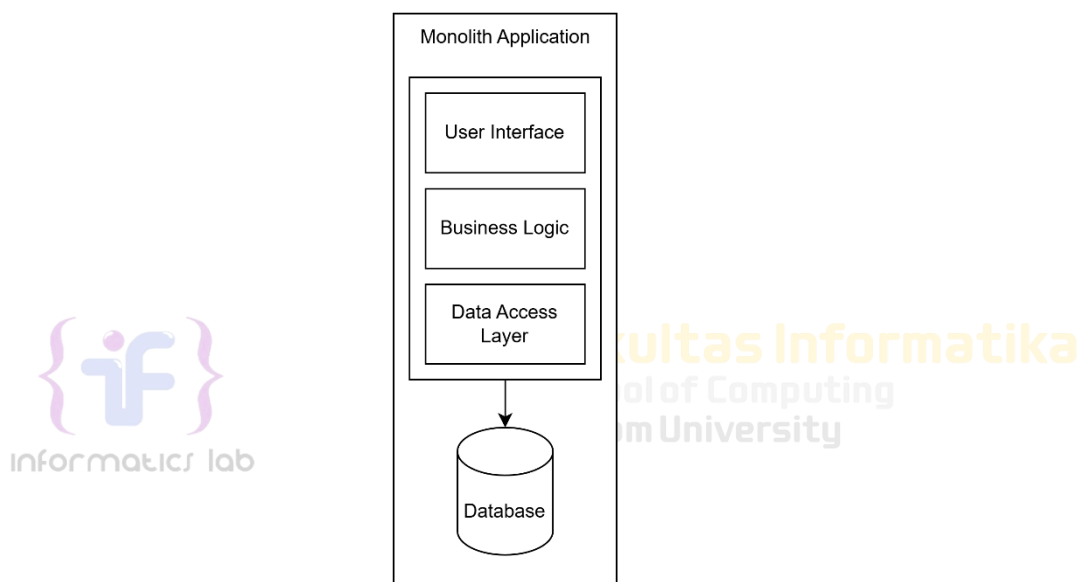
Modul 11 Architectural Pattern: Monolithic

Tujuan Praktikum

1. Memahami konsep Monolithic Architecture dalam pengembangan aplikasi.
2. Menganalisis kelebihan dan kekurangan arsitektur monolitik.
3. Mengembangkan aplikasi berbasis monolitik menggunakan Spring Boot.

11.1 Monolithic Architecture

Monolith Architecture adalah salah satu pendekatan tradisional dalam membangun aplikasi di mana seluruh komponen aplikasi (user interface, logika bisnis, data access, dan database dijalankan dalam satu kesatuan atau satu sourcecode/codebase. Semua fitur dari aplikasi tersebut berjalan dalam satu proses yang besar.



Gambar 11-1. Monolith Architecture

Gambar di atas menunjukkan arsitektur Monolith Application, di mana semua komponen yaitu User Interface, Business Logic, dan Data Access Layer terintegrasi dalam satu sistem dan menggunakan satu database terpusat. Arsitektur ini sederhana untuk dikembangkan dan diterapkan, tetapi seiring pertumbuhan aplikasi, skalabilitas dan pemeliharaannya bisa menjadi lebih sulit karena setiap perubahan mempengaruhi seluruh sistem.

11.1.1 Karakteristik Monolithic Architecture

Terdapat beberapa karakteristik utama dari monolithic architecture, yaitu:

6. Semua komponen dalam satu aplikasi, komponen-komponen tersebut umumnya merupakan backend, frontend, dan database. Komponen-komponen tersebut berada dalam satu kesatuan.
7. Komunikasi antar komponen langsung berada dalam satu proses.
8. Pengelolaan sistem dengan monolithic architecture lebih sederhana karena semuanya berada dalam 1 sourcecode.

11.1.2 Kelebihan dan Kekurangan Monolithic Architecture

Meskipun monolithic architecture ini terlihat sederhana, terdapat kelebihan dan kekurangannya. Untuk kelebihan dari monolithic architecture yaitu:

1. Mudah dikembangkan untuk aplikasi kecil dan menengah.
2. Proses deploy sederhana, seperti melalui satu container.
3. Efisien untuk sistem kecil karena komunikasi antar modul langsung.
4. Satu source code lebih mudah dikelola dalam tahap awal.

Meskipun begitu, terdapat beberapa kekurangannya, antara lain:

1. Sulit diperbesar karena ukuran total source code membengkak.
2. Setiap perubahan kecil mengharuskan untuk deploy ulang seluruh aplikasi.
3. Kurang efisien untuk skala besar karena sulit dibagi menjadi bagian kecil.
4. Sulit dipelihara jika aplikasi berkembang terlalu besar.
5. Jika terdapat kesalahan pada suatu modul, maka seluruh aplikasi dapat berhenti.

11.2 Implementasi Monolithic dengan Spring Boot

Berikut merupakan contoh implementasi monolithic dengan menggunakan Java dan Spring Boot. Pada contoh kali ini, akan dibuat 2 service, yaitu user service dan product service.

1. Konfigurasi Project dan Database

Seperti yang dilakukan pada Modul 10 bagian “10.3 REST API Spring Boot di Eclipse”, buatlah project baru dengan nama monolithic-app sebagai contoh. Dependencies yang akan digunakan adalah:

- Spring Web
- Spring Boot DevTools (untuk automatic reload)
- Spring Data JPA
- MySQL Driver

Buat database untuk digunakan dalam project ini. Konfigurasi database di file application.properties yang ada di folder src/main/resources. Buatlah table user dan table product pada database. Table users terdiri dari:

- id
- name
- email.

Table products terdiri dari:

- id
- name
- price

2. Model

Sebelum membuat user model dan product model, buatlah package untuk mengatur file-file model ini. Sebagai contoh buatlah package model atau “com.example.monolithic.model”. Selanjutnya dalam package tersebut, buatlah file baru dengan nama “User.java” dan “Product.java”. Berikut merupakan isi dari model user dalam user.java:

```
package com.example.monolithic.model;
```

```

import jakarta.persistence.*;

@Entity
@Table(name = "users")
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    private String name;

    @Column(nullable = false, unique = true)
    private String email;

    public User() {}

    public User(String name, String email) {
        this.name = name;
        this.email = email;
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getEmail() {
        return email;
    }
}

```



```
public void setEmail(String email) {  
    this.email = email;  
}  
}
```

Selanjutnya, ini merupakan isi dari model product pada file product.java:

```
package com.example.monolithic.model;  
  
import jakarta.persistence.*;  
  
@Entity  
@Table(name = "products")  
public class Product {  
  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long id;  
  
    @Column(nullable = false)  
    private String name;  
  
    @Column(nullable = false)  
    private double price;  
  
    public Product() {}  
  
    public Product(String name, double price) {  
        this.name = name;  
        this.price = price;  
    }  
    public Long getId() {  
        return id;  
    }  
    public void setId(Long id) {  
        this.id = id;  
    }  
    public String getName() {  
        return name;  
    }  
}
```

```

    }
    public void setName(String name) {
        this.name = name;
    }
    public double getPrice() {
        return price;
    }
    public void setPrice(double price) {
        this.price = price;
    }
}

```

3. Repository

Repository digunakan untuk berkomunikasi dengan database. Pada contoh kali ini karena terdapat dua model, maka akan dibuat dua repository, yaitu User Repository dan Product Repository. Buatlah package yang berisi kumpulan repository tersebut, sebagai contoh package itu bernama “com.example.monolithic.repository”. Buatlah file java baru untuk kedua repository, yang bernama UserRepository.java dan ProductRepository.java.

UserRepository.java

```

package com.example.monolithic.repository;

import com.example.monolithic.model.User;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface UserRepository extends JpaRepository<User, Long> {
}

```

ProductRepository.java

```

package com.example.monolithic.repository;

import com.example.monolithic.model.Product;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface ProductRepository extends JpaRepository<Product, Long> {
}

```

4. Service

Service akan menggunakan repository yang telah dibuat sebelumnya. Buatlah package untuk mengelompokkan service dengan nama "com.example.monolithic.service". Pada package tersebut, buat 2 file java untuk service nya dengan nama UserService.java dan ProductService.java.

UserService.java

```
package com.example.monolithic.service;

import com.example.monolithic.model.User;
import com.example.monolithic.repository.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;

@Service
public class UserService {

    @Autowired
    private UserRepository userRepository;

    public List<User> getAllUsers() {
        return userRepository.findAll();
    }

    public Optional<User> getUserById(Long id) {
        return userRepository.findById(id);
    }

    public User createUser(User user) {
        return userRepository.save(user);
    }

    public void deleteUser(Long id) {
        userRepository.deleteById(id);
    }
}
```

```
}  
}
```

ProductService.java

```
package com.example.monolithic.service;  
  
import com.example.monolithic.model.Product;  
import com.example.monolithic.repository.ProductRepository;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
import java.util.List;  
import java.util.Optional;  
  
@Service  
public class ProductService {  
  
    @Autowired  
    private ProductRepository productRepository;  
  
    public List<Product> getAllProducts() {  
        return productRepository.findAll();  
    }  
  
    public Optional<Product> getProductById(Long id) {  
        return productRepository.findById(id);  
    }  
  
    public Product createProduct(Product product) {  
        return productRepository.save(product);  
    }  
  
    public void deleteProduct(Long id) {  
        productRepository.deleteById(id);  
    }  
}
```

5. Controller

Buat juga package untuk mengelompokkan controller yang akan digunakan untuk membuat API dengan nama "com.example.monolithic.controller". Controller yang akan dibuat adalah UserController.java dan ProductController.

UserController.java

```
package com.example.monolithic.controller;

import com.example.monolithic.model.User;
import com.example.monolithic.service.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.Optional;

@RestController
@RequestMapping("/users")
public class UserController {

    @Autowired
    private UserService userService;

    @GetMapping
    public List<User> getAllUsers() {
        return userService.getAllUsers();
    }

    @GetMapping("/{id}")
    public Optional<User> getUserById(@PathVariable Long id) {
        return userService.getUserById(id);
    }

    @PostMapping
    public User createUser(@RequestBody User user) {
        return userService.createUser(user);
    }

    @DeleteMapping("/{id}")
    public String deleteUser(@PathVariable Long id) {
```

```

        userService.deleteUser(id);
        return "User deleted successfully";
    }
}

```

ProductController.java

```

package com.example.monolithic.controller;

import com.example.monolithic.model.Product;
import com.example.monolithic.service.ProductService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.Optional;

@RestController
@RequestMapping("/products")
public class ProductController {

    @Autowired
    private ProductService productService;

    @GetMapping
    public List<Product> getAllProducts() {
        return productService.getAllProducts();
    }

    @PostMapping
    public Product createProduct(@RequestBody Product product) {
        return productService.createProduct(product);
    }
}

```

6. Test API

Jalankan project yang telah dibuat, dan uji API yang telah dibuat dengan Postman.

a. Create New User

```
POST http://localhost:8080/users
```

Body (JSON):

```
{
  "name": "John Doe",
  "email": "johndoe@example.com"
}
```

b. Get All Users

```
GET http://localhost:8080/users
```

c. Get a User by ID

```
GET http://localhost:8080/users/1
```

d. Delete a User

```
DELETE http://localhost:8080/users
```



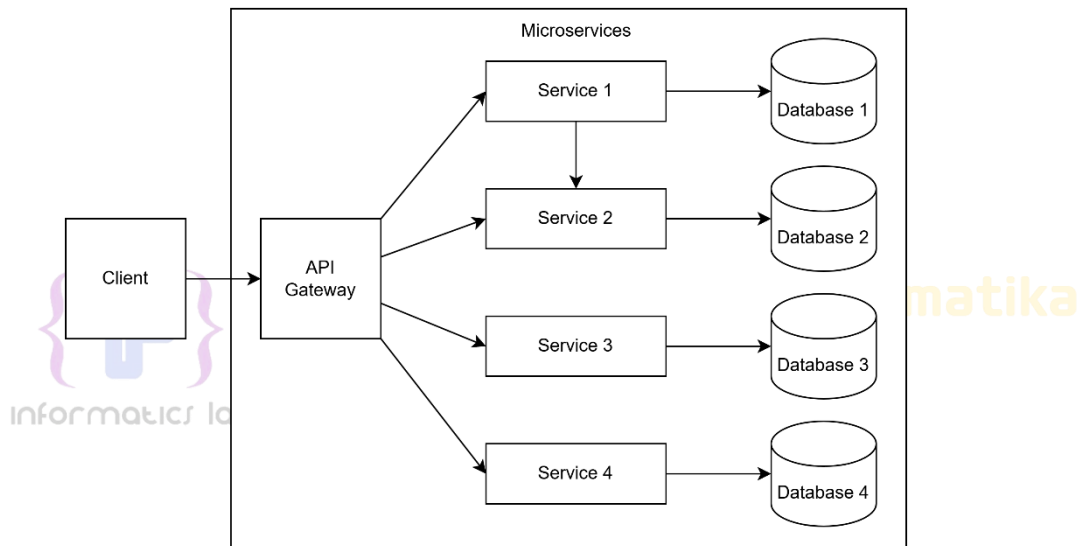
Modul 12 Architectural Pattern: Microservice

Tujuan Praktikum

1. Mengetahui konsep Microservices Architecture dalam pengembangan aplikasi.
2. Menganalisis perbedaan antara Monolithic Architecture dan Microservices Architecture.
3. Mengembangkan Microservices dengan Spring Boot.

12.1 Microservices Architecture

Microservices adalah salah satu architectural pattern di mana aplikasi dibagi menjadi beberapa layanan (services) kecil, setiap layanan dapat berjalan secara independen dan berkomunikasi satu sama lain menggunakan protokol seperti Application Programming Interface (API), gRPC, message brokers (Kafka, RabbitMQ, dll), dan lain sebagainya.



Gambar 12-1. Microservices Architecture

Gambar tersebut menunjukkan arsitektur berbasis microservices, di mana client (seperti aplikasi web atau mobile) berkomunikasi melalui API Gateway untuk mengakses berbagai layanan. API Gateway bertindak sebagai perantara yang mengarahkan permintaan client ke microservices yang sesuai. Setiap microservice beroperasi secara independen dan memiliki database terpisah, yang memungkinkan skalabilitas, isolasi data, dan fleksibilitas dalam pengelolaan layanan. Selain itu, beberapa microservices dapat berkomunikasi langsung satu sama lain jika diperlukan, seperti yang ditunjukkan antara Service 1 dan Service 2.

12.1.1 Karakteristik Microservices Architecture

Terdapat beberapa karakteristik utama dari microservices architecture, yaitu:

1. Setiap layanan yang independen membuatnya mudah untuk dikembangkan, diuji, dan dideploy secara terpisah.
2. Setiap layanan yang independen juga dapat diskalakan secara individual, sehingga setiap layanan bisa diperbesar tanpa mempengaruhi layanan lain.
3. Komunikasi antar layanan dengan menggunakan protokol.
4. Jika satu layanan gagal, tidak akan langsung menyebabkan sistem mati total.

12.1.2 Kelebihan dan Kekurangan Microservices Architecture

Meskipun microservices architecture ini terlihat lebih rumit daripada monolithic, terdapat kelebihan dan kekurangannya. Untuk kelebihan dari microservices architecture yaitu:

1. Tim dapat bekerja pada layanan yang berbeda secara paralel.
2. Setiap layanan bisa dideploy terpisah tanpa harus merilis seluruh aplikasi.
3. Bisa diskalakan berdasarkan kebutuhan setiap layanan.
4. Mudah untuk melakukan update pada bagian tertentu tanpa mengganggu sistem lain.
5. Jika ada suatu layanan gagal, tidak akan langsung menyebabkan sistem mati total.

Meskipun begitu, terdapat beberapa kekurangannya, antara lain:

1. Kompleksitas yang tinggi karena banyak layanan yang harus dikelola.
2. Membutuhkan infrastruktur tambahan untuk mengelola setiap layanan.
3. Komunikasi antar layanan menambah latensi.
4. Debugging yang lebih sulit karena banyak service yang harus dipantau.

12.2 Implementasi Microservices dengan Spring Boot

Berikut merupakan contoh implementasi microservice dengan menggunakan framework Spring Boot dalam Eclipse IDE. Dalam contoh kali ini akan dibuat dua layanan/service, yaitu User Service dan Product Service. Kedua service tersebut akan berkomunikasi dengan REST APIs melalui API Gateway dan akan diatur melalui Eureka Discovery Server.

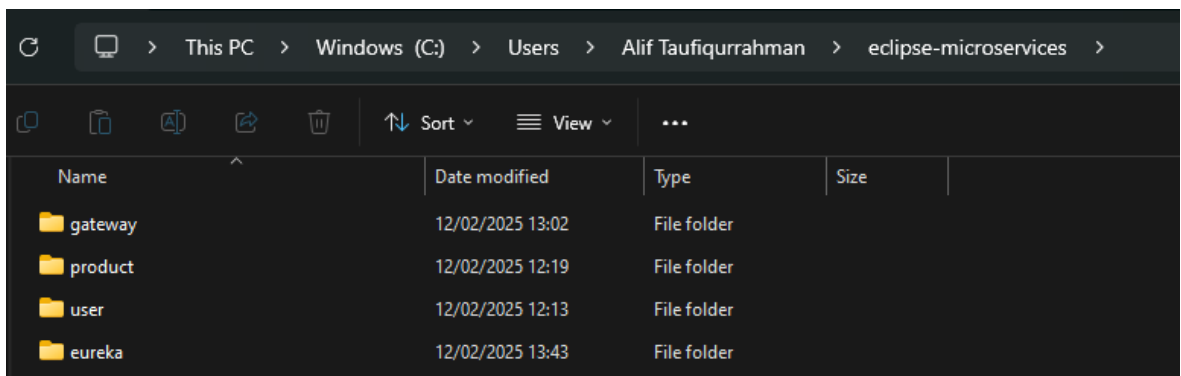
Eureka Discovery Server dalam Spring Boot berfungsi sebagai service registry dalam arsitektur microservices. Server ini memungkinkan setiap layanan untuk didaftarkan dan menemukan layanan lain secara dinamis.

12.2.1 Membuat Spring Boot Project untuk Setiap Layanan di Eclipse

Dalam mengimplementasikan microservices dalam Eclipse IDE, setiap layanan harus dibuka dalam satu (1) tab atau aplikasi Eclipse. Dalam contoh ini membutuhkan empat (4) project yang dibuka dengan aplikasi Eclipse yang terpisah. Keempat project tersebut adalah *api-gateway*, *eureka-server*, *user-service*, dan *product service*.

1. Workspace/Folder

Sebelum membuat project, buatlah 1 folder untuk menampung project-project yang akan dibuat. Pada contoh ini, buatlah 4 folder untuk project yang akan dibuat.

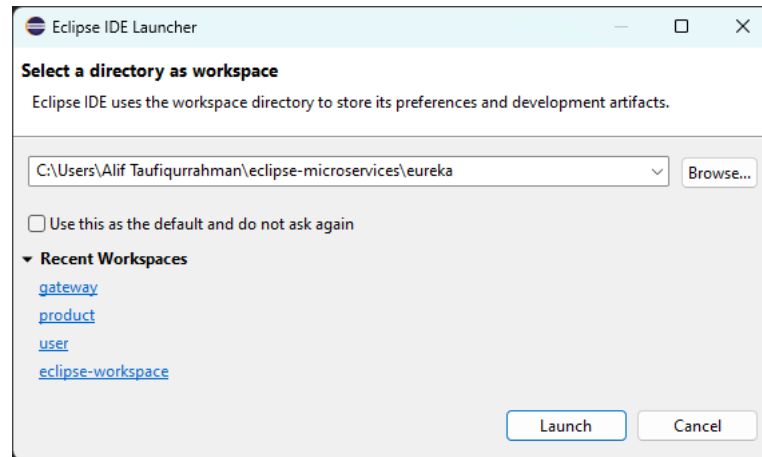


Gambar 12-2. Folder-folder project

Sebagai contoh, pada gambar 20 di atas, terdapat 4 folder untuk masing-masing project yang akan dibuat.

2. Eureka Server
 - a. Workspace

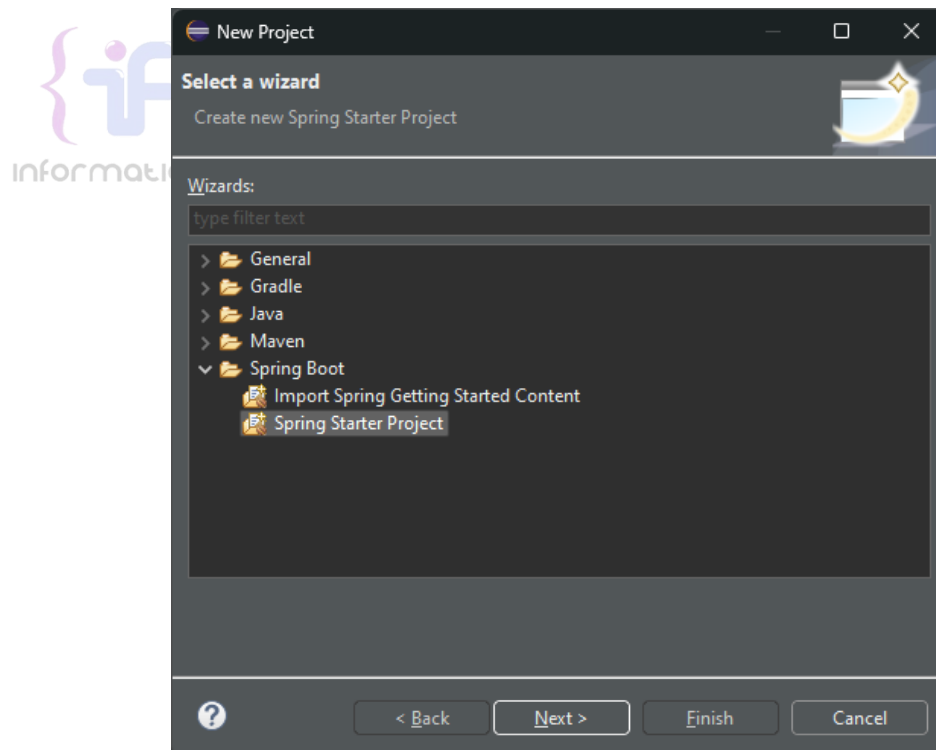
Project pertama yang akan dibuat adalah project *eureka-server*. Sebelum membuat project, buka Eclipse IDE. Ketika memilih workspace, pilih folder eureka yang telah dibuat.



Gambar 12-3 Eureka Server Workspace

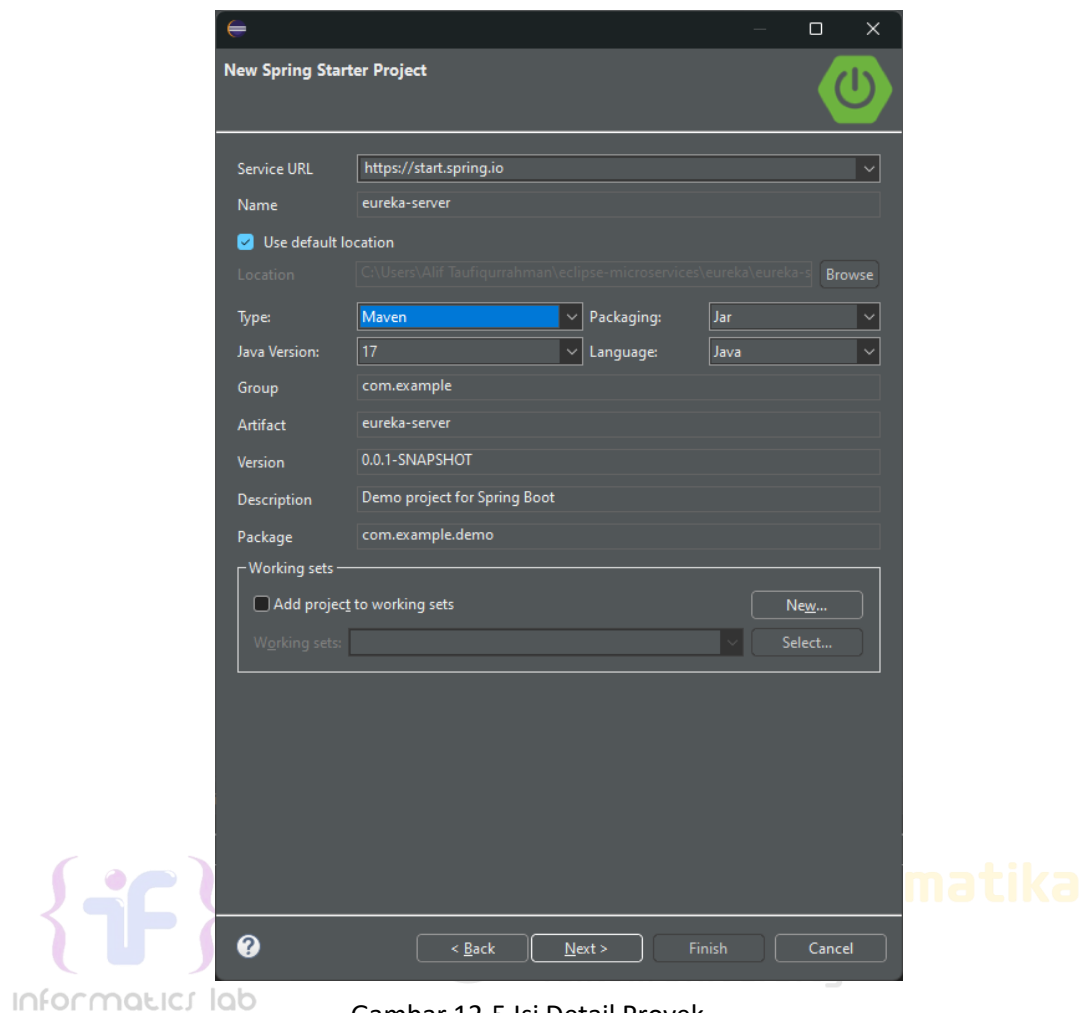
- b. New Project

Click File → New → Project → Spring Starter Project



Gambar 12-4 Membuat Proyek Spring 1

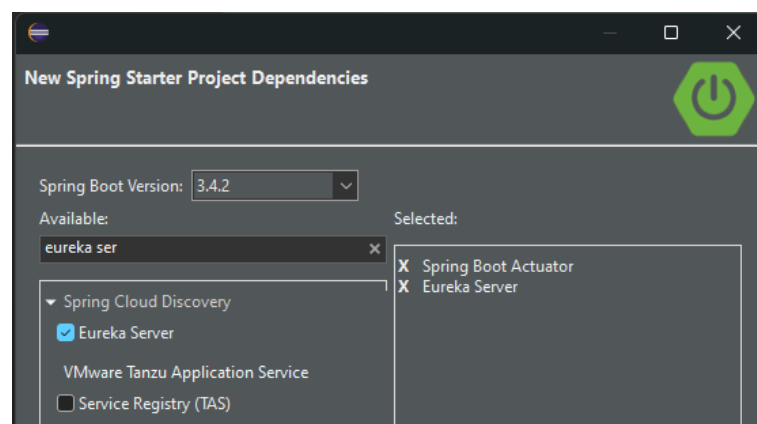
Klik Next. Masukkan Name (nama project) dengan “eureka-server”, pilih Type menjadi “Maven”, dan pilih Java Version menjadi 17.



Gambar 12-5 Isi Detail Proyek

Klik Next dan pilih dependencies. Dependencies yang akan digunakan pada project eureka server ini adalah:

- Eureka Server
- Spring Boot Actuator



Gambar 12-6 Menambah Dependencies Proyek 1

Selanjutnya klik finish dan project eureka akan terbuat.

c. Konfigurasi Eureka Server

Buka File `EurekaServerApplication.java` pada package `com.example.demo` yang telah dibuat. Lalu tambahkan `@EnableEurekaServer` di atas definisi class.

```
package com.example.demo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import
org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

@SpringBootApplication
@EnableEurekaServer
public class EurekaServerApplication {

    public static void main(String[] args) {
        SpringApplication.run(EurekaServerApplication.class, args);
    }
}
```

Selanjutnya konfigurasi file `application.properties` di dalam folder `"src/main/resources"` dengan menambahkan hal berikut: (`spring.application.name` dapat disesuaikan dengan nama project masing-masing).

```
spring.application.name=eureka-server
server.port=8761

eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false
```

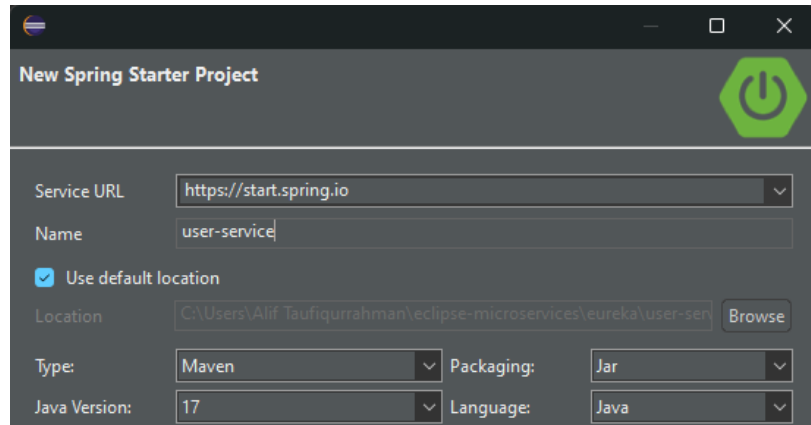
3. User Service

a. Workspace

Seperti langkah yang terdapat pada nomor "2. Eureka Server" bagian "a. Workspace". Buka Eclipse baru, tidak disatukan/disamakan dengan project lainnya. Sebagai contoh, masukkan "C:\Users\<name> \eclipse-microservices\user" sebagai workspace yang akan digunakan.

b. New Project

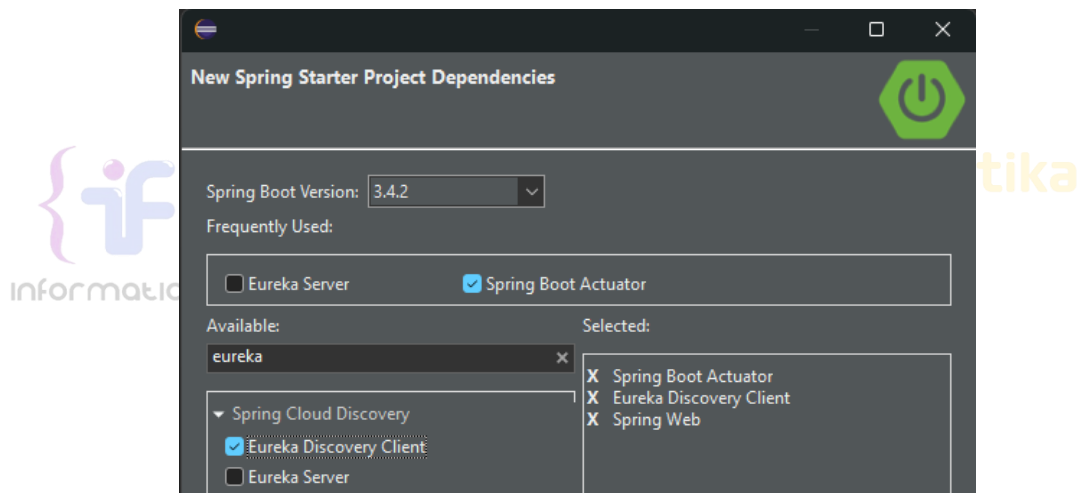
Seperti langkah yang terdapat pada nomor "2. Eureka Server" bagian "b. New Project". Masukkan nama project yang sesuai. Sebagai contoh, masukkan "user-service" untuk project name.



Gambar 12-7 Membuat Proyek Spring 2

Pada service ini menggunakan dependencies yang berbeda dengan project eureka server sebelumnya. Service ini menggunakan 3 dependencies, yaitu:

- Spring Web
- Eureka Discovery Client
- Spring Boot Actuator



Gambar 12-8 Menambah Dependencies Proyek 1

c. Konfigurasi User Service

Pada project user-service, konfigurasi file application.properties yang ada pada folder src/main/resources.

```
server.port=8081
spring.application.name=user-service
eureka.client.service-url.defaultZone=http://localhost:8761/eureka/
```

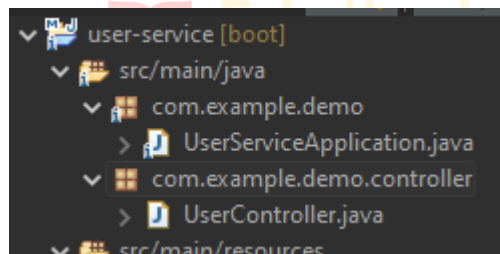
Selanjutnya, konfigurasi file utama UserServiceApplication.java dengan menambahkan "@EnableDiscoveryClient" sebelum deklarasi class.

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
```

```
import
org.springframework.cloud.client.discovery.EnableDiscoveryClient;

@SpringBootApplication
@EnableDiscoveryClient
public class UserServiceApplication {
    public static void main(String[] args) {
        SpringApplication.run(UserServiceApplication.class, args);
    }
}
```

Untuk menjalankan fungsionalitas dari microservices, tambahkan controller untuk menggunakan REST API. Bagian ini hanyalah contoh sederhana yang Anda bisa tambah sendiri. Sebelum membuat class controller, buat package untuk menaruh controllernya agar project tidak berantakan. Sebagai contoh, package yang digunakan pada project ini adalah “com.example.demo”. Klik kanan > New > Package pada “com.example.demo” pada tab Package Explorer di bagian kiri layar aplikasi. Masukkan “com.example.demo.controller” pada input new package. Klik kanan > New > Class untuk membuat class UserController.java.



Gambar 12-9 Struktur Folder

Konfigurasi file tersebut sehingga menjadi:

```
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/users")
public class UserController {
    @GetMapping("/")
    public String getUser() {
        return "Test User";
    }
}
```

4. Product Service

a. Workspace

Seperti langkah yang terdapat pada nomor “3. User Service” bagian “a. Workspace”. Buka Eclipse baru, tidak disatukan/disamakan dengan project lainnya. Sebagai contoh, masukkan “ C:\Users\<name> \eclipse-microservices\product” sebagai workspace yang akan digunakan.

b. New project

Masukkan project name menjadi “product-service”. Kemudian pilih dependencies yang sama dengan project User Service sebelumnya, yaitu:

- Spring Web
- Eureka Discovery Client
- Spring Boot Actuator

c. Konfigurasi Product Server

Pada project productr-service, konfigurasi file application.properties yang ada pada folder src/main/resources.

```
server.port=8082
spring.application.name=product-service
eureka.client.service-url.defaultZone=http://localhost:8761/eureka/
```

Selanjutnya, konfigurasi file utama ProductServiceApplication.java dengan menambahkan “@EnableDiscoveryClient” sebelum deklarasi class.

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import
org.springframework.cloud.client.discovery.EnableDiscoveryClient;

@SpringBootApplication
@EnableDiscoveryClient
public class ProductServiceApplication {
    public static void main(String[] args) {
        SpringApplication.run(ProductServiceApplication.class, args);
    }
}
```

Buat package baru untuk mengatur file controller seperti sebelumnya, sebagai contoh “com.example.demo.controller”. Buat class ProductController.java pada package tersebut. Isi class tersebut menjadi seperti:

```
import org.springframework.web.bind.annotation.GetMapping;
```

```
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/products")
public class ProductController {

    @GetMapping("/")
    public String getProduct() {
        return "Test Product";
    }
}
```

5. API Gateway

a. Workspace

Seperti langkah sebelumnya, buka Eclipse baru dan masukkan “C:\Users\<name>\eclipse-microservices\gateway” pada input workspace.

b. New Project

Seperti langkah pembuatan project baru pada langkah sebelumnya, masukkan “api-gateway” sebagai nama project. Dependencies yang akan digunakan adalah:

- Eureka Discovery Client
- Spring Boot Actuator

c. Konfigurasi API Gateway

Dibutuhkan dependencies tambahan untuk mengkonfigurasi API Gateway, yaitu Spring Cloud Gateway. Pada root folder project, buka file “pom.xml” dan tambahkan dependency berikut pada bagian <dependencies>:

```
<dependencies>
  <!-- ... Other dependency -->
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-webflux</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-gateway</artifactId>
  </dependency>
  <!-- ... Other dependency -->
</dependencies>
```


Jika belum ada versi pada pom.xml, tambahkan spring cloud version di file pom.xml pada bagian dependencyManagement.

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-dependencies</artifactId>
      <version>2022.0.4</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

Konfigurasi file application.properties yang ada pada folder src/main/resources menjadi seperti berikut.

```
server.port=8083
spring.application.name=api-gateway
eureka.client.service-url.defaultZone=http://localhost:8761/eureka/

spring.cloud.gateway.routes[0].id=user-service
spring.cloud.gateway.routes[0].uri=lb://user-service
spring.cloud.gateway.routes[0].predicates[0]=Path=/users/**

spring.cloud.gateway.routes[1].id=product-service
spring.cloud.gateway.routes[1].uri=lb://product-service
spring.cloud.gateway.routes[1].predicates[0]=Path=/products/**
```

Tambahkan “@EnableDiscoveryClient” pada class utama ApiGatewayAppliacion.java menjadi seperti berikut:

```
package com.example.apigateway;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
```

```
import
org.springframework.cloud.client.discovery.EnableDiscoveryClient;

@SpringBootApplication
@EnableDiscoveryClient
public class ApiGatewayApplication {
    public static void main(String[] args) {
        SpringApplication.run(ApiGatewayApplication.class, args);
    }
}
```

6. Run Microservices.

Jalankan keempat project tersebut dalam workspace yang berbeda. Jalankan Eureka Server, User Service, Product Service, dan Api Gateway project. Buka <http://localhost:8761/> untuk mengecek semua service yang teregistrasi dalam eureka server.

7. Test API Gateway

Dalam menguji API yang ada pada microservices, anda dapat menggunakan tools seperti Postman, dengan cURL, atau yang lainnya.

a. Test User Service

GET <http://localhost:8083/users/>

Response:

"Test User"

b. Test Product Service

GET <http://localhost:8083/products/>

Response:

"Test Product"

12.2.2 Komunikasi Antar Service

Terdapat berbagai cara untuk membuat komunikasi antar service yang ada pada ekosistem microservices. Salah satunya adalah dengan menggunakan RestTemplate. Sebagai contoh, akan dibuat komunikasi user service dengan product service. User service akan memanggil API pada product service. Berikut konfigurasi yang ada pada User Service

1. Bean Rest Template

Pada class utama UserServiceApplication.java tambahkan "Bean" untuk rest templatennya.

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
```

```

import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
import org.springframework.context.annotation.Bean;
import org.springframework.web.client.RestTemplate;

@SpringBootApplication
@EnableDiscoveryClient
public class UserServiceApplication {
    public static void main(String[] args) {
        SpringApplication.run(UserServiceApplication.class, args);
    }

    @Bean
    public RestTemplate restTemplate() {
        return new RestTemplate();
    }
}

```

2. User Controller

Tambahkan konfigurasi komunikasi API dengan rest template pada user controller.

```

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.client.RestTemplate;

@RestController
@RequestMapping("/users")
public class UserController {
    @Autowired
    private RestTemplate restTemplate;

    @GetMapping("/")
    public String getUser() {
        return "Test User";
    }

    @GetMapping("/product")
    public String getProductMessage() {

```

```
        String productMessage =  
restTemplate.getForObject("http://localhost:8083/products/ ",  
String.class);  
        return "User Service calling Product Service: " +  
productMessage;  
    }  
}
```

3. Test API

Uji API dari user service yang berkomunikasi dengan product service melalui:

GET http://localhost:8083/users/product

Response:

User Service calling Product Service: Test Product



Modul 13 JMeter Part 1

Performance Testing, Load Testing, & Stress Testing

Tujuan Praktikum
1. Memahami penggunaan dasar JMeter.
2. Menggunakan JMeter untuk melakukan performance testing, load testing, dan stress testing.

13.1 Pendahuluan

JMeter adalah sebuah *open-source software* yang digunakan untuk melakukan *performance testing* dan *load testing* pada berbagai *platform*, seperti aplikasi web, API, database, hingga server FTP. JMeter memungkinkan pengguna untuk mensimulasikan sejumlah besar pengguna virtual untuk mengukur respons sistem terhadap berbagai skenario penggunaan.



13.2 Prerequisite

Sebelum menjalankan JMeter, terdapat beberapa persyaratan yang harus dipenuhi. JMeter merupakan *software* berbasis Java, sehingga diperlukan Java Development Kit (JDK) atau Java Runtime Environment (JRE). JMeter kompatibel dengan Java 8 atau lebih tinggi, namun disarankan menggunakan dengan versi yang terbaru untuk meningkatkan keamanan dan kinerja aplikasi. Meskipun JRE dapat digunakan, JDK lebih direkomendasikan untuk fungsionalitas yang lebih baik.

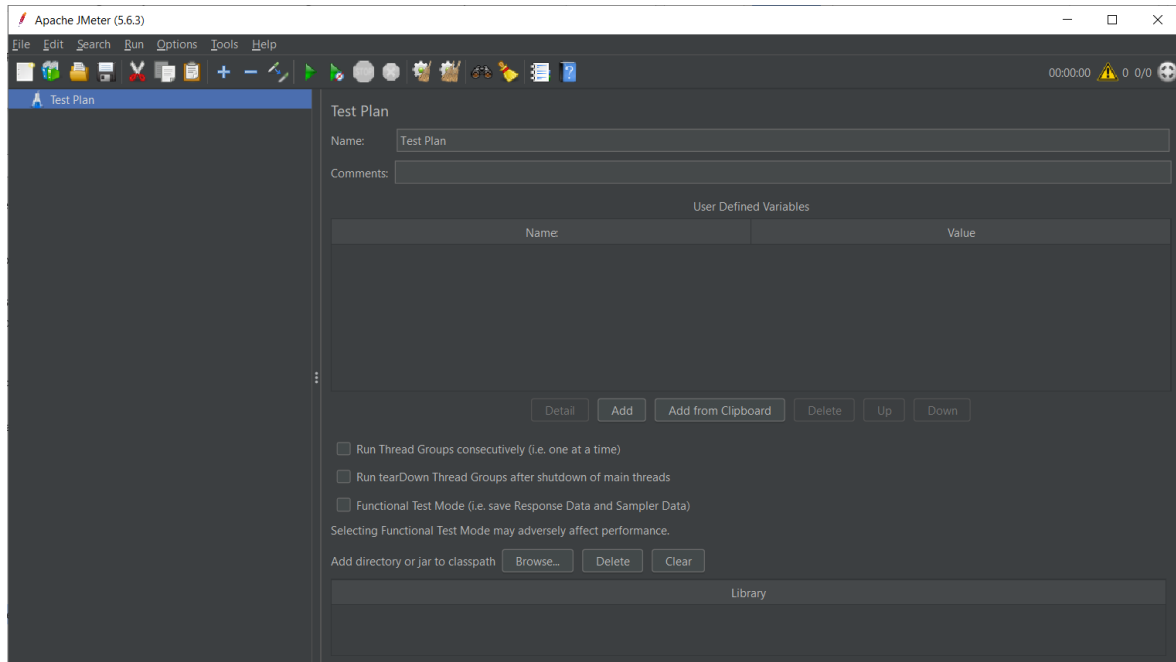
13.3 Instalasi JMeter

Untuk menggunakan JMeter, diperlukan proses instalasi yang mencakup mengunduh, mengekstrak, dan menjalankannya. Berikut adalah langkah-langkah instalasi JMeter:

1. Buka halaman *download* pada situs resmi Apache JMeter yaitu https://jmeter.apache.org/download_jmeter.cgi
2. Pilih versi terbaru yang tersedia.
3. Unduh file Binary (zip/tgz), bukan source code. Pilih format JMeter sesuai dengan OS yang digunakan, umumnya pilih file berformat .zip dengan contoh apache-jmeter-5.6.3.zip.
4. Ekstrak file yang telah didownload.
5. Masuk ke folder JMeter yang telah diekstrak dan navigasikan ke folder 'bin'.
6. Buka aplikasi JMeter melalui file jmeter.bat.

13.4 Antarmuka JMeter

Setelah instalasi berhasil, JMeter dapat dijalankan dalam mode GUI. Salah satu antarmuka utama JMeter yaitu Test Plan. Panel test plan merupakan struktur yang menampilkan elemen-elemen dalam skenario pengujian.



Gambar 13-1 Tampilan Utama JMETER

Test Plan adalah area tempat Anda menambahkan berbagai elemen yang diperlukan dalam pengujian JMeter, seperti ThreadGroup, Listener, Sampler, Assertion, Timer, Processor, serta pengaturan lain yang mendukung pelaksanaan tes. Di dalam Test Plan, Anda juga dapat mengonfigurasi variabel global dan mengimpor pustaka, misalnya file .jar, ke dalam direktori.

13.4.1 Thread Group

Thread group akan menentukan jumlah pengguna virtual yang akan diuji, termasuk durasi eksekusi dan bagaimana request akan dikirimkan. Panel thread group bisa dibuka dengan klik kanan pada **Test Plan > Add > Threads (Users) > Thread Group**. Setiap thread merepresentasikan satu pengguna virtual yang akan mengirim permintaan server. Konfigurasi utama pada Thread Group meliputi:

6. Action to be taken after a Sample error: Menentukan apa yang terjadi jika kesalahan sampler terjadi, baik karena sample yang gagal atau pernyataan yang gagal. Berikut merupakan pilihan yang ada pada Action:
 - a. Continue: Abaikan kesalahan dan lanjut ke pengujian berikutnya.
 - b. Start Next Thread Loop: Abaikan kesalahan, mulai loop berikutnya dan lanjutkan dengan pengujian.
 - c. Stop Thread: Hentikan thread.
 - d. Stop Test: Seluruh pengujian akan dihentikan pada akhir sample.
 - e. Stop Test Now: Seluruh pengujian dihentikan secara tiba-tiba. Setiap sampler akan berhenti.
2. Number of Threads (Users): Jumlah pengguna virtual yang akan dikirim.
3. Ramp-Up Period: Waktu yang dibutuhkan hingga mencapai total yang ditentukan.
4. Loop Count: Jumlah iterasi atau pengulangan pengujian.

13.4.2 Listener

Listener adalah komponen untuk menampilkan dan menyimpan hasil pengujian. Listener dapat menampilkan data dalam bentuk tabel, grafik, atau log. Setiap listener akan membaca dan menganalisis hasil request yang dikirimkan oleh Sampler.

13.4.3 Sampler

Sampler bertujuan untuk mengirimkan request ke server. Setiap sampler memiliki jenis permintaan yang berbeda, seperti HTTP Request, FTP Request, JDBC Request, dan lainnya. Berikut merupakan contoh sampler yang umum digunakan:

1. HTTP Request: Mengirimkan request HTTP atau HTTPS ke server.
2. JDBC Request: Mengirimkan query ke database.
3. FTP Request: Mengunduh atau mengunggah file melalui FTP.
4. dan lain sebagainya.

13.4.4 Assertion

Assertion digunakan untuk memvalidasi hasil pengujian dengan memeriksa apakah respons yang diterima itu sesuai dengan ekspektasi yang didefinisikan. Assertion membantu memastikan bahwa hasil pengujian memiliki status yang benar, waktu respons yang wajar, atau konten yang sesuai.

13.4.5 Timer

Timer digunakan untuk mengatur waktu tunda antara request yang dikirimkan oleh pengguna virtual. Tanpa Timer, JMeter akan mengirimkan request secara terus-menerus tanpa jeda, hal ini tidak mencerminkan skenario penggunaan yang nyata.

13.4.6 Processor

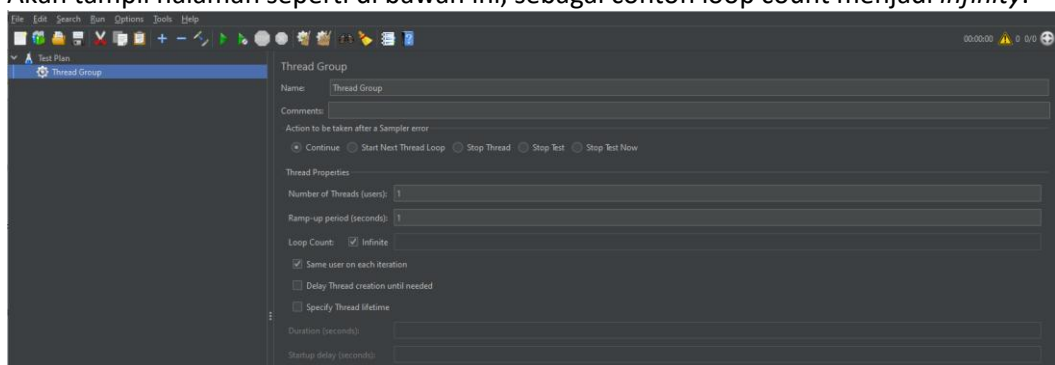
Processor dalam JMeter berfungsi untuk memodifikasi request atau respons sebelum atau setelah dikirimkan oleh sampler. Processor dibagi menjadi dua jenis utama:

1. Pre-Processor digunakan untuk memanipulasi request sebelum dikirim ke server.
2. Post-Processor digunakan untuk memanipulasi data yang diterima dari server.

13.5 Performance Test

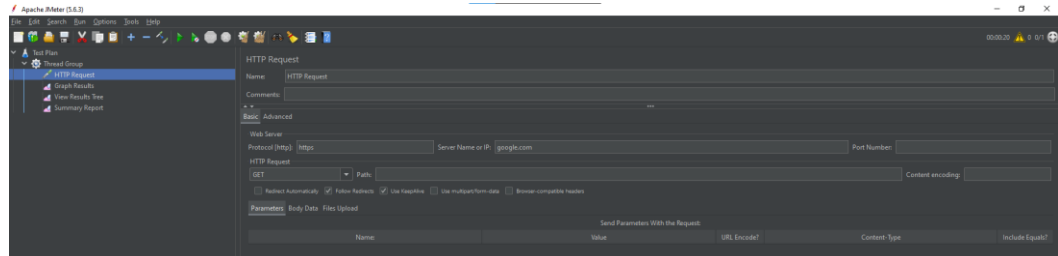
Performance Testing adalah pengujian yang bertujuan untuk mengukur dan mengevaluasi kinerja sistem dalam berbagai kondisi. Performance testing terdapat beberapa tipe, yaitu load testing, stress testing, dan jenis pengujian lainnya. Berikut merupakan langkah-langkah dalam membuat performance test menggunakan JMeter:

1. Buka JMeter dalam mode GUI.
2. Pilih File → New jika ingin membuat test plan baru.
3. Menambahkan Thread Group, setiap elemen pada thread group dijelaskan pada bagian 13.4.1.
 - a. Klik kanan pada Test Plan → Add → Threads (Users) → Thread Group.
 - b. Akan tampil halaman seperti di bawah ini, sebagai contoh loop count menjadi *infinity*.



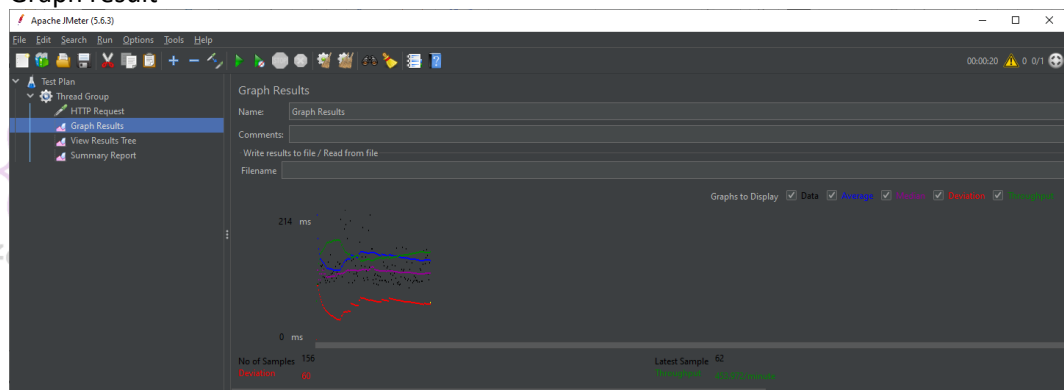
Gambar 13-2 Menambahkan Thread Group

4. Menambahkan Sampler, Anda dapat menambahkan berbagai elemen sampler sesuai dengan kebutuhan pengujian. Sebagai contoh sampler yang akan digunakan adalah HTTP Request.
 - d. Klik kanan pada Thread Group → Add → Sampler → HTTP Request.
 - e. Atur parameter:
 - Server Name or IP: Masukkan URL website target, contoh: google.com
 - Protocol dapat diisi dengan http atau https.
 - Path merupakan endpoint atau path pada URL target, contoh: google.com/maps



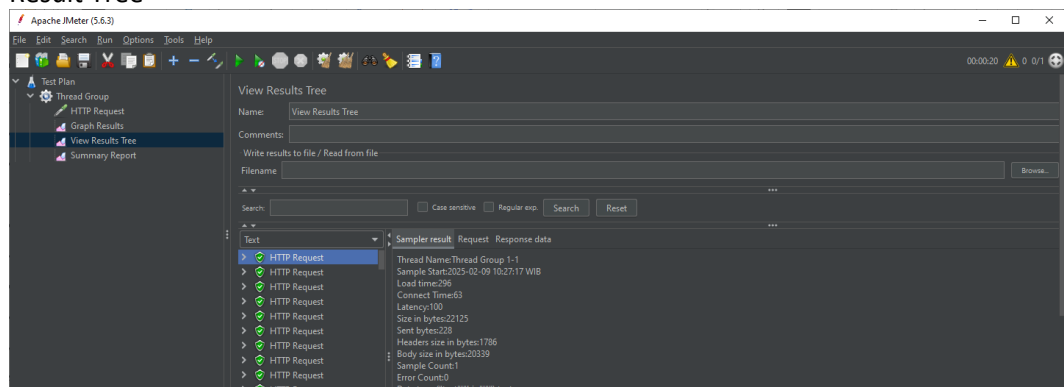
Gambar 13-3 Pengaturan Parameter

5. Menambahkan listener, untuk melihat hasil pengujian, Anda dapat menambahkan berbagai jenis listener.
 - d. Klik kanan pada Thread Group → Add → Listener → ... pilih sesuai kebutuhan.
 - e. Graph result



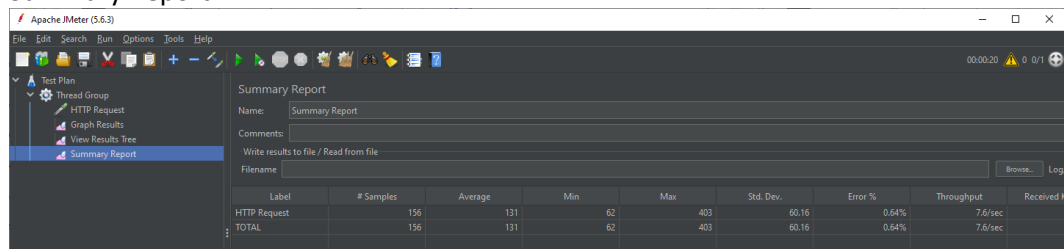
Gambar 13-4 Graph Hasil

f. Result Tree



Gambar 13-5 Result Tree

g. Summary Report



Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received K
HTTP Request	136	131	62	403	60.16	0.64%	7.6/sec	
TOTAL	136	131	62	403	60.16	0.64%	7.6/sec	

Gambar 13-6 Ringkasan Hasil Pengujian

6. Jalankan pengujian.

Untuk menjalankan pengujian dan mendapatkan hasilnya, Anda dapat menekan button play pada toolbar atau menekan CTRL + R pada keyboard. JMeter akan menjalankan skenario pengujian sesuai konfigurasi yang dibuat. Hasil dapat dianalisis melalui Listener yang telah ditambahkan.

13.5.1 Load Test

Load testing merupakan salah satu tipe performance testing yang digunakan untuk mengevaluasi kinerja aplikasi dengan memberikan beban tertentu yang diharapkan dapat berjalan normal. Tujuan dari load testing adalah untuk mengetahui bagaimana sistem berperforma saat digunakan oleh sejumlah pengguna dalam kondisi normal. Sebagai contoh, akan diuji situs situs web e-commerce untuk melihat apakah dapat menangani 500 pengguna secara bersamaan yang melakukan transaksi dalam waktu 10 menit. Berikut merupakan tahapan dan konfigurasi yang akan dilakukan:

1. Thread Group
 - a. Konfigurasi Number of Threads (users) sebanyak 500 (hanya contoh, setiap website memiliki load yang berbeda) untuk mendukung tujuan ini.
 - b. Konfigurasi Ramp-up period ke 100 detik agar pengguna meningkat secara bertahap.
2. Sampler
 - d. Tambahkan HTTP Request Sampler untuk menguji situs web.
 - e. Tambahkan Server Name or IP yang sesuai, contoh: tokopedia.com
 - f. Tambahkan Path jika ingin mensimulasikan aktivitas tertentu seperti pencarian produk.
3. Listener
Tambahkan listener sesuai dengan kebutuhan yang ingin dianalisis.

13.5.2 Stress Test

Stress testing merupakan salah satu tipe performance testing yang dilakukan untuk mengetahui batas maksimum kinerja sistem dengan memberikan beban yang jauh lebih tinggi dari kondisi normal. Pengujian ini bertujuan untuk melihat bagaimana sistem bereaksi saat mencapai atau melebihi kapasitasnya. Sebagai contoh dalam mengetahui batas maksimum sistem e-commerce, sehingga akan diuji dengan 5000 pengguna secara bersamaan yang melakukan aktivitas terus-menerus dalam waktu 30 menit. Berikut merupakan tahapan dan konfigurasi yang akan dilakukan:

1. Thread Group
 - c. Konfigurasi Number of Threads (users) sebanyak 5000 (hanya contoh, setiap website memiliki load yang berbeda) untuk mendukung tujuan ini.
 - d. Konfigurasi Ramp-up period yang lebih pendek untuk memberikan lonjakan pengguna yang cepat.
2. Sampler
 - a. Tambahkan HTTP Request Sampler untuk menguji situs web.

- b. Tambahkan Server Name or IP yang sesuai, contoh: tokopedia.com
 - c. Tambahkan Path jika ingin mensimulasikan aktivitas tertentu seperti pencarian produk.
 - d. Menggunakan Loop untuk menambah intensitas request
3. Listener
- Tambahkan listener sesuai dengan kebutuhan yang ingin dianalisis.



Modul 14 JMeter Part 2

Functional Testing & API Testing

Tujuan Praktikum
1. Memahami konsep dasar pengujian fungsional pada API.
2. Mengimplementasikan pengujian dengan menggunakan JMeter.

14.1 Functional Testing dengan JMeter

Functional Testing adalah jenis pengujian perangkat lunak yang bertujuan untuk memvalidasi apakah sistem bekerja sesuai dengan persyaratan dan spesifikasi yang telah ditentukan. Pengujian ini memverifikasi apakah fitur dalam aplikasi bekerja sebagaimana mestinya berdasarkan spesifikasi.

JMeter umumnya digunakan untuk performance testing, namun juga dapat digunakan untuk functional testing. Melalui JMeter dapat mengotomatiskan pengujian ini dengan cara mengirimkan permintaan (request) ke aplikasi dan memverifikasi respons yang diterima. Terdapat berbagai scenario yang dapat dilakukan pada JMeter, seperti:

1. Pengujian API/Web Service untuk memastikan respons yang diberikan sesuai dengan yang diharapkan.
2. Pengujian login dan autentikasi untuk memvalidasi mekanisme keamanan aplikasi.
3. Validasi data pada database, dan lain sebagainya.

14.2 API Testing dengan JMeter

Pada bagian ini, akan dilakukan functional testing terhadap API. API Testing adalah proses menguji API untuk memastikan bahwa permintaan dan respons yang dikirim melalui API sesuai dengan spesifikasi. API memungkinkan komunikasi antara berbagai sistem atau layanan dalam aplikasi modern, sehingga pengujian API menjadi sangat penting.

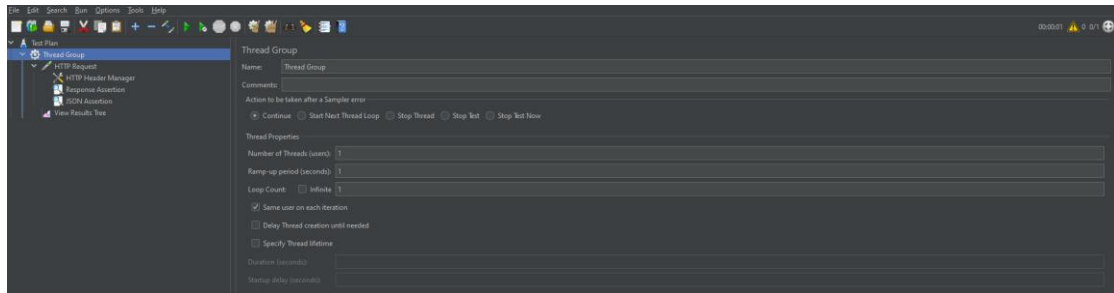
Terdapat beberapa jenis API utama yang dapat diuji, seperti:

1. REST API yang menggunakan format JSON atau XML.
2. SOAP API yang menggunakan format XML berbasis SOAP protocol.
3. GraphQL API yang merupakan API modern yang memungkinkan query spesifik.

14.2.1 Membuat Functional Test pada API menggunakan JMeter

Berikut ini akan dibuat pengujian fungsional untuk REST API menggunakan Jmeter. Pada REST API, terdapat empat operasi utama, yaitu create (post), read (get), update (put/patch), dan delete (delete). Dalam bagian ini akan dilakukan functional test terhadap fungsi create.

1. Test Plan
Buat test plan baru melalui File → New
2. Thread Group
Buat thread group melalui klik kanan pada "Test Plan" → Add → Threads (Users) → Thread Group. Atur parameter thread group sebagai berikut:
 - Number of Threads (Users): 1
 - Ramp-up Period: 1
 - Loop Count: 1



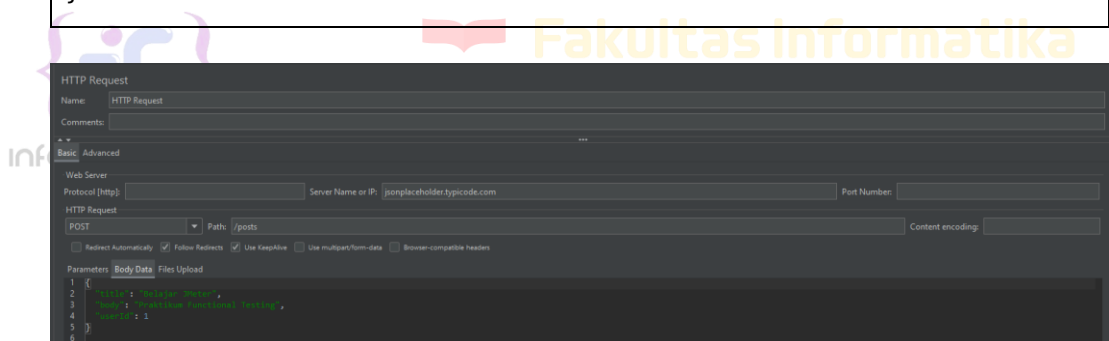
Gambar 14-1 Menambahkan Tread Group

3. Sampler

Karena akan melakukan functional test terhadap API, maka sampler yang akan digunakan adalah HTTP Request. Tambahkan HTTP Request sampler dengan klik kanan pada Thread Group → Add → Sampler → HTTP Request. Endpoint API yang akan diuji adalah jsonplaceholder.typicode.com/posts. Atur konfigurasi sebagai berikut:

- Server Name or IP: jsonplaceholder.typicode.com
- Path: /posts
- Method: POST (digunakan post karena akan menguji fungsi create)
- Body Data:

```
{
  "title": "Belajar JMeter",
  "body": "Praktikum Functional Testing",
  "userId": 1
}
```

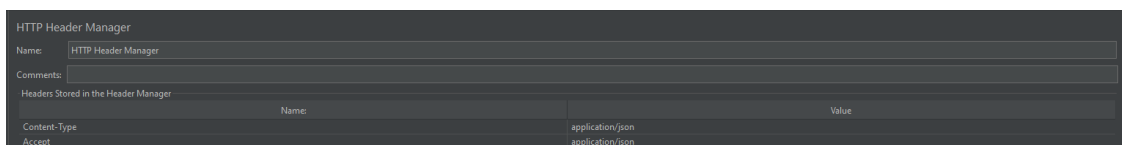


Gambar 14-2 Menambahkan Sampler

4. HTTP Header Manager

Klik kanan pada HTTP Request → Add → Config Element → HTTP Header Manager. Selanjutnya tambahkan header berikut:

- Content-Type: application/json
- Accept: application/json



Gambar 14-3 Menambahkan Header Manager

5. Assertion

Assertion dalam JMeter adalah mekanisme untuk memverifikasi bahwa respons yang diterima sesuai dengan harapan. Melalui assertion dapat dipastikan pengujian yang dilakukan valid. Jika

assertion gagal, maka hasil pengujian dianggap tidak berhasil. Terdapat beberapa jenis assertion yang umum digunakan, seperti:

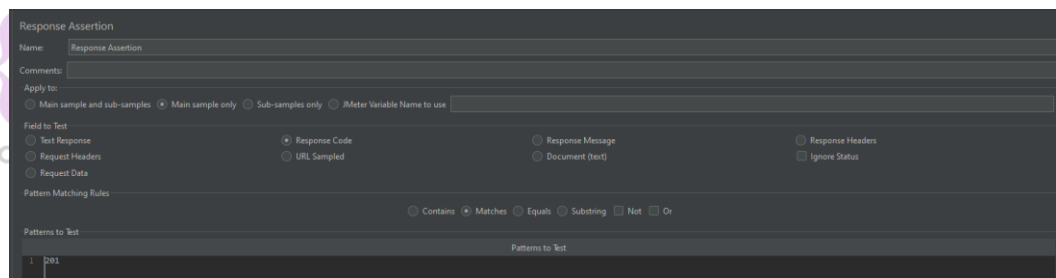
- Response Assertion
Memeriksa respons API mengandung teks tertentu atau memiliki kode status yang benar. Contohnya adalah memastikan kode status 200 untuk permintaan GET.
- Duration Assertion
Memeriksa waktu respons lebih cepat dari batas tertentu. Contoh memastikan waktu respons tidak lebih dari 500ms.
- Size Assertion
Memeriksa ukuran respons yang diterima sesuai dengan ekspektasi.
- JSON Assertion
Memverifikasi respons dalam format JSON memiliki elemen tertentu, seperti respons API POST yang mengandung id.
- XPath Assertion
Memverifikasi respons dalam format XML.

Pada contoh ini akan digunakan response assertion dan JSON assertion.

a. Response Assertion

Klik kanan pada HTTP Request (GET) → Add → Assertions → Response Assertion. Konfigurasi sebagai berikut:

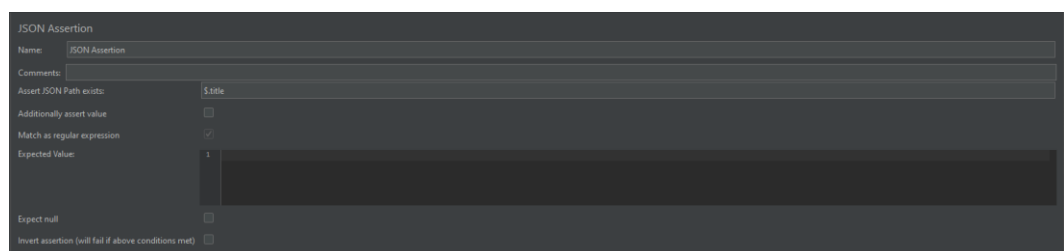
- Pilih "Response Code" pada Field to Test.
- Pada Pattern to Test, masukkan 201 untuk memastikan permintaan berhasil diproses.



Gambar 14-4 Response Assertion

b. JSON Assertion

Klik kanan pada HTTP Request (GET) → Add → Assertions → JSON Assertion. Masukkan \$.title untuk memastikan bahwa judul dari postingan yang diterima sesuai dengan ekspektasi.



Gambar 14-5 JSON Assertion

6. Listener

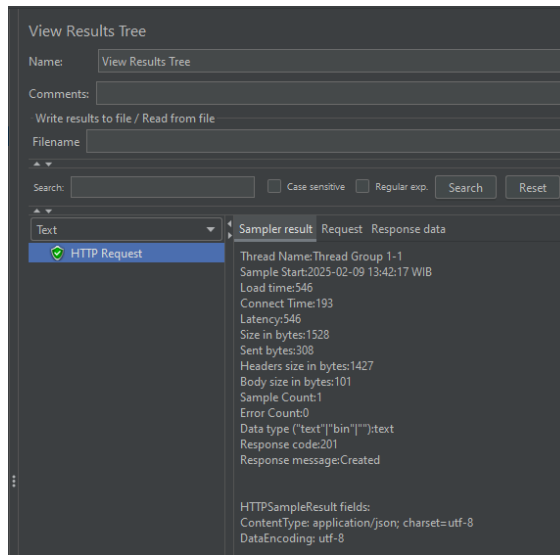
Listener yang akan dipakai pada contoh kali ini adalah Result Tree. Listener ini dapat diakses melalui Klik kanan pada Thread Group → Add → Listener → View Result Tree. Result Tree dipakai untuk melihat hasil dari assertion.

7. Hasil Pengujian

Setelah menjalankan pengujian, perhatikan hasil yang ditampilkan di results tree. Jika assertion berhasil, maka pengujian dianggap sukses. Sebaliknya, jika assertion gagal, respons API tidak sesuai dengan harapan karena terdapat kesalahan. Berikut merupakan beberapa kemungkinan kesalahan yang dapat terjadi:

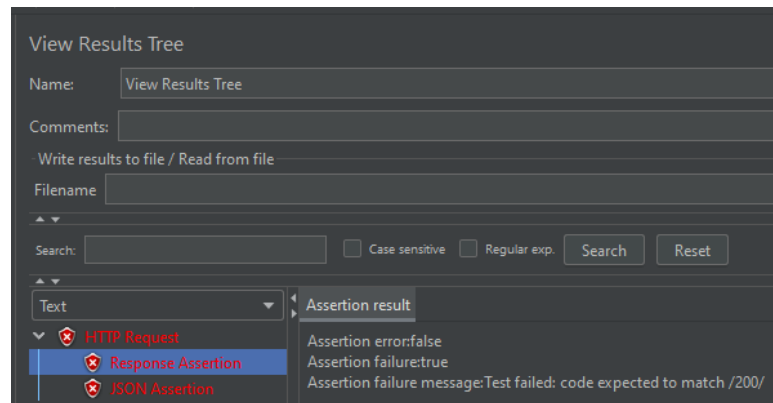
- Request dan Respons tidak sesuai dengan format.
- API mengalami error (500 internal server error).
- API tidak mengembalikan data yang diharapkan (404 not found), dan lain sebagainya.

Hasil jika pengujian berhasil:



Gambar 14-6 Pengujian Berhasil

Hasil jika pengujian gagal:



Gambar 14-7 Pengujian Gagal.

Terdapat assertion yang gagal beserta pesan errornya.

Modul 15 COTS Penerapan Tubes 2

Tujuan Praktikum

1. Menerapkan arsitektur perangkat lunak yang telah dipilih pada tugas besar.
2. Mengimplementasikan aplikasi menggunakan Spring Boot sesuai dengan desain yang telah dirancang.

15.1 COTS Penerapan Tugas Besar 2

Pada Modul 15 ini, kalian telah mempelajari berbagai jenis arsitektur perangkat lunak yang umum digunakan dalam pengembangan aplikasi. Mulai dari Model-View-Controller (MVC) yang membantu memisahkan logika bisnis, tampilan, dan kontrol alur data, hingga arsitektur Monolitik, yang mengembangkan aplikasi sebagai satu kesatuan yang terintegrasi.

Selain itu, kalian juga telah mempelajari konsep Microservices, yang memungkinkan sistem dibangun sebagai kumpulan layanan kecil yang dapat dikembangkan, diterapkan, dan dioperasikan secara independen. Dengan pemahaman ini, diharapkan kalian mampu menentukan pendekatan arsitektur yang paling sesuai untuk aplikasi yang sedang dikembangkan serta memahami kelebihan dan tantangan dari masing-masing pendekatan tersebut.

Sebagai langkah selanjutnya, kalian akan menerapkan desain arsitektur yang telah dibuat menggunakan framework Spring Boot. Penerapan ini harus sesuai dengan arsitektur yang telah dirancang sebelumnya, baik itu berbasis MVC, monolitik, maupun microservices. Implementasi akan mencakup penyusunan struktur proyek yang sesuai, pemisahan lapisan-lapisan dalam aplikasi berdasarkan arsitektur yang dipilih, serta penggunaan best practices dalam pengembangan Spring Boot.

Dengan menerapkan desain arsitektur yang telah dirancang, diharapkan aplikasi yang dikembangkan lebih terstruktur, modular, scalable, dan mudah dikelola sesuai dengan prinsip pengembangan perangkat lunak yang baik.

Daftar Pustaka

[1]	Anonim. 2023. <i>Modul Praktikum Pemrograman Berorientasi Objek</i> . Fakultas Informatika. Institut Teknologi Telkom, Bandung.
[2]	Anonim. 2023. <i>Modul Praktikum Konstruksi Perangkat Lunak</i> . Fakultas Informatika. Institut Teknologi Telkom, Bandung.
[3]	Anonim. 2024. <i>Modul Praktikum Pengujian Perangkat Lunak</i> . Fakultas Informatika. Institut Teknologi Telkom, Bandung.
[4]	Deitel, H.M & Deitel,P.J. 2004. <i>Java How to Program Sixth Edition</i> . New Jersey: Prentice Hall.
[5]	Downey, Tim. 2012. <i>Guide to Web Development with Java</i> . Springer
[6]	Sierra, Kathy & Bates, Bert. 2005. <i>Head First Java, 2nd Edition</i> . Sebastopol: O'Reilly Media.
[7]	Sierra, Kathy & Bates, Bert. 2008. <i>SCJP Sun Certified Programmer for Java™ 6 Study Guide</i> . New York: McGraw-Hill
[8]	Edward Hill, Jr. 2005. <i>Learning to Program Java</i> . New York: iUniverse.
[9]	Java Tutorial. https://beginnersbook.com/java-tutorial-for-beginners-with-examples/ . 28 Desember 2018.
[10]	Java Tutorial. https://tutorialspoint.com/java/ . 28 Desember 2018.
[11]	Java Tutorial. https://w3schools.com/java . 28 Desember 2018.
[12]	Java SE Documentations. http://docs.oracle.com/javase . 28 Desember 2018.
[13]	SonarQube Doucmentations. https://docs.sonarsource.com/sonarqube-server/10.8/ . 3 Februari 2025
[14]	SOLID Principle Simplified. https://medium.com/@shubhadeepchat/solid-principle-simplified-b18b73b3e440 . 6 Februari 2025



Kontak Kami :



@fiflab



Praktikum IF LAB



informaticslab@telkomuniversity.ac.id



informatics.labs.telkomuniversity.ac.id