



Unit Testing dan Load Testing Web App Sederhana

Versi: 1.2 (20 Maret 2018)

Update 1.1:

(Menambahkan *constructor* pada `StudentServiceDatabase`)

Update 1.2

(`import static org.assertj.core.internal.bytebuddy.matcher.ElementMatchers.is;` dihapus)

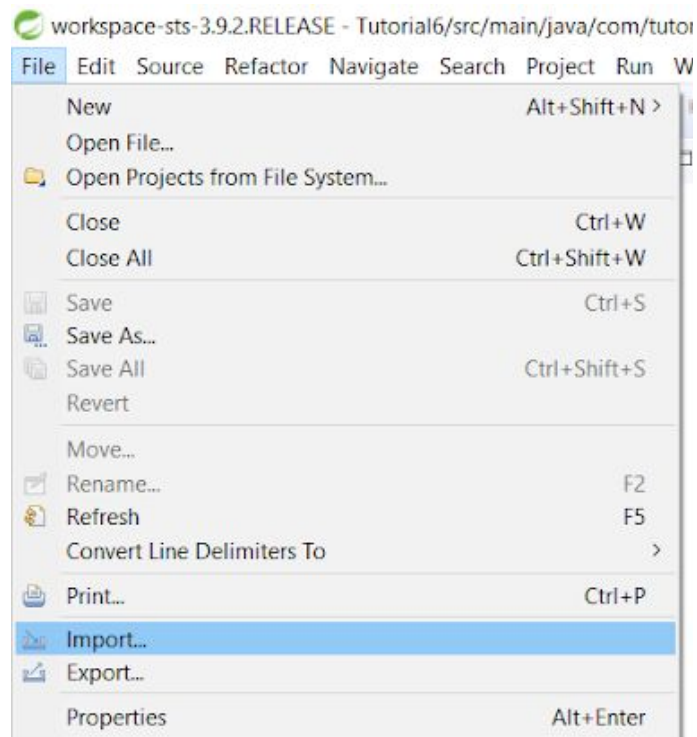
Outline:

- Melakukan testing sederhana terhadap service dengan plugin JUnit dan Mockito
- Melakukan load testing sederhana dengan aplikasi Apache JMeter

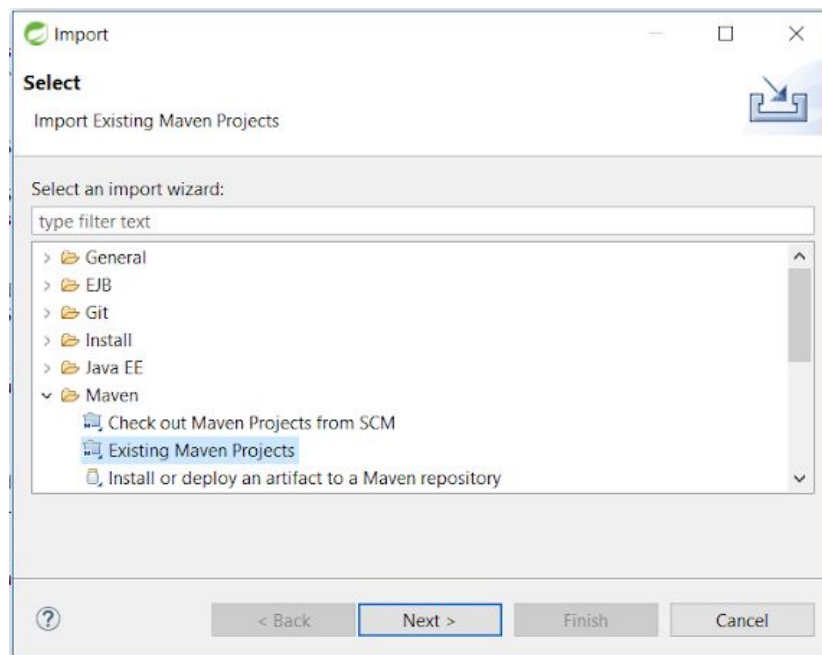
Requirements:

Sebelum memulai tutorial ini pastikan komputer Anda sudah terinstall JDK, Maven, Git, dan Eclipse dengan Spring Tools Suite atau Standalone Spring Tools Suite.

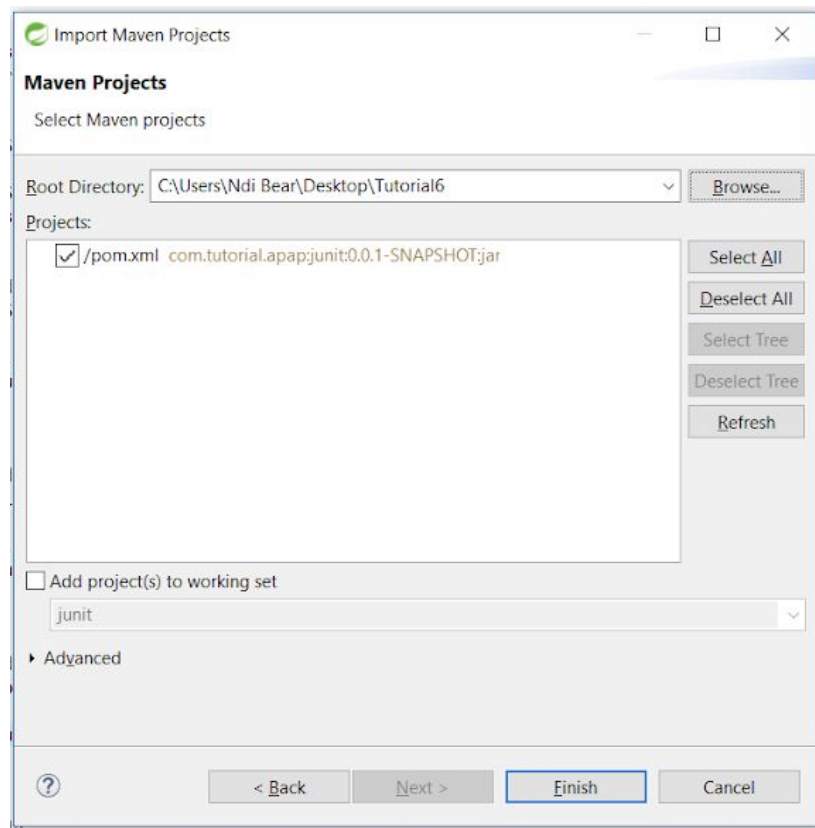
Dalam tutorial ini kita akan menggunakan kembali **project** dari tutorial sebelumnya. Anda perlu melakukan **copy project tutorial 05** yang sudah dikerjakan minggu lalu (tanpa mengcopy folder `.git` dari project tutorial 05). Lalu pada STS dapat dilakukan *import* dengan klik **File > Import**.



Setelah itu pilih **Maven > Existing Maven Project** dan klik **Next**



Maka akan muncul *window* dan masukkan direktori file Tutorial sebelumnya yang ingin diimport menuju STS. Lalu klik **Finish**.



Tunggu sejenak dan folder telah berhasil diimpor.

Perhatian: Jangan copy-paste kode pada pdf ini ke eclipse/IDE Anda. Kode Anda tidak akan berjalan atau dikenali oleh compiler karena karakter tidak memiliki encoding yang sama.

Unit Testing

Penjelasan

Dalam pemrograman komputer, *unit testing* merupakan sebuah metode untuk melakukan *testing* pada program yang telah kita buat berdasarkan masing-masing fungsi ataupun *method*. *Testing* ini dilakukan agar setiap fungsi dan *method* yang kita implementasi apakah sudah sesuai dengan *output* yang diharapkan.

Hal ini sangat dibutuhkan karena sebagai *programmer*, kita tidak hanya akan membuat program dengan skala yang kecil. Melainkan kita juga akan membuat program dengan skala yang besar namun tetap dengan kualitas kode yang tetap baik. Maka dari itu, dalam pemrograman disediakan *unit testing* sebagai media untuk *testing* apakah program yang kita buat sudah baik.

Spring Boot telah memiliki 2 *dependencies* utama dalam melaksanakan *unit testing*, yaitu:

- JUnit
- Mockito.

JUnit merupakan sebuah *testing framework* yang terfokus pada pengecekan *class* dan *method*. JUnit dapat membantu *programmer* dalam melakukan pengecekan secara otomatis untuk setiap *class* dan *method*. Lalu Mockito merupakan sebuah *dependencies* untuk menciptakan *dummy* dalam pengecekan sehingga *programmer* dapat memanipulasi objek untuk dijadikan *test case*.

Tugas anda pada tutorial kali ini yaitu menciptakan *testing* untuk setiap *method* pada *service* yang telah dibuat pada tutorial sebelumnya (Tutorial 5).

Namun sebelum instalasi dimulai harap ubah *StudentMapper*, *StudentService*, dan *StudentService* seperti dibawah:

StudentMapper (Mengubah *return value* yang awalnya *void* menjadi *boolean*)

```
@Insert("INSERT INTO student (npm, name, gpa) VALUES ({npm}, {name}, {gpa})")
boolean addStudent (StudentModel student);
```

StudentService (Mengubah *return value* yang awalnya *void* menjadi *boolean*)

```
boolean addStudent (StudentModel student);
```

StudentServiceDatabase (Mengubah *return value* yang awalnya *void* menjadi *boolean*)

```
@Override
```

```
public boolean addStudent (StudentModel student)
{
    return studentMapper.addStudent (student);
}
```

StudentServiceDatabase (Menambahkan *constructor* pada StudentServiceDatabase)

```
public StudentServiceDatabase(){}

public StudentServiceDatabase(StudentMapper studentMapper){
    this.studentMapper = studentMapper;
}
```

Tutorial

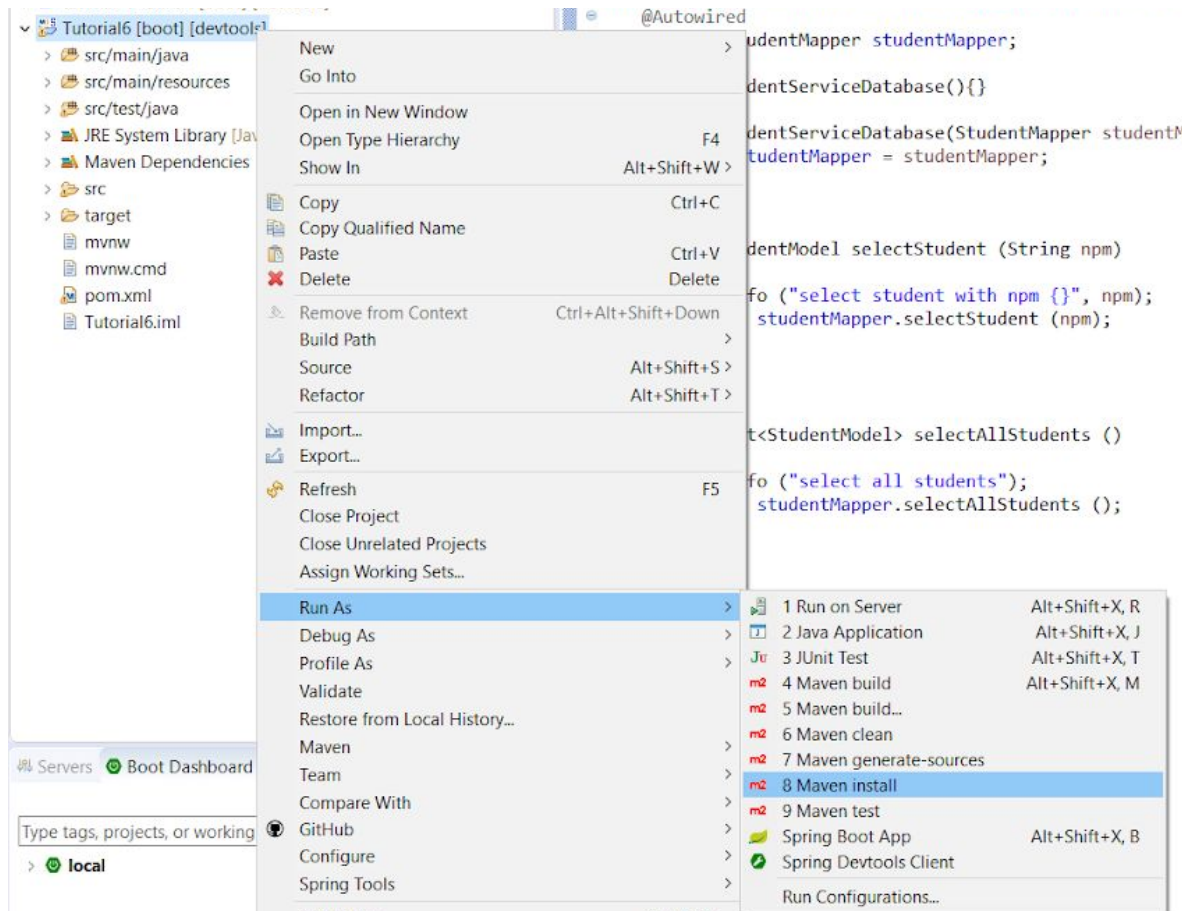
Instalasi JUnit dan Mockito

Pertama, tambahkan dependensi JUnit dan Mockito pada pom.xml yang terdapat pada *root* dari proyek ini.

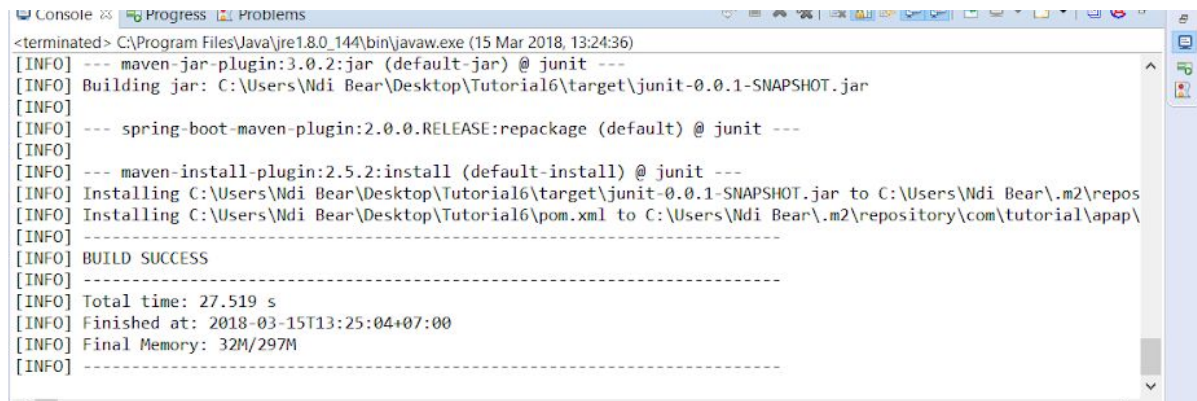
```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.12</version>
  <scope>test</scope>
  <exclusions>
    <exclusion>
      <groupId>org.hamcrest</groupId>
      <artifactId>hamcrest-core</artifactId>
    </exclusion>
  </exclusions>
</dependency>

<dependency>
  <groupId>org.hamcrest</groupId>
  <artifactId>hamcrest-library</artifactId>
  <version>1.3</version>
  <scope>test</scope>
</dependency>
```

Kedua, lakukan Maven Clean Install pada proyek dengan mengklik kanan proyek ini lalu menuju Maven Install.



Tunggu hingga proses Maven Install sukses



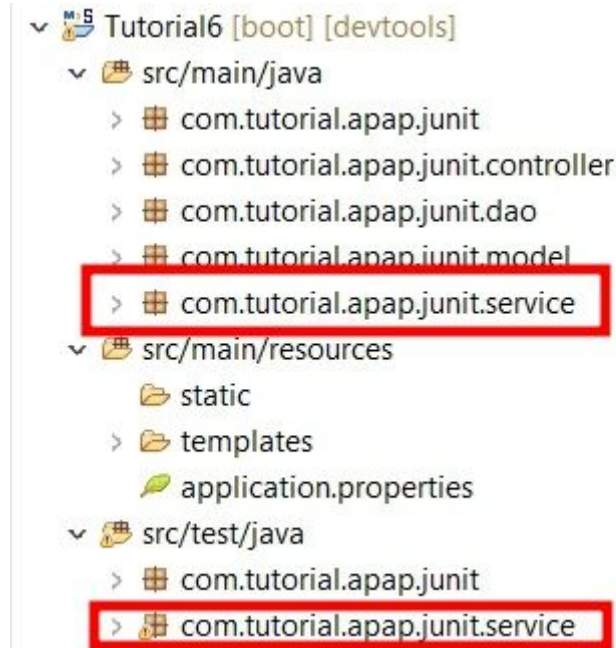
Proses instalasi JUnit dan Mockito telah berhasil!

Membuat *testing* pada Spring Boot

Pada tutorial kali ini, kita akan membuat *testing* pada setiap fungsi yang terdapat pada setiap *service*. Dan berikut merupakan langkah-langkah dalam membuat *testing* menggunakan Mockito dan JUnit

1. Buatlah *package* untuk *service* pada folder *src/test*. Penamaan dari *package* tersebut sesuai dengan penamaan pada *package service* pada *src/main*. Contoh jika pada *src/main* kita tertulis

'com.tutorial.apap.junit.service', maka pada src/test kita tulis 'com.tutorial.apap.junit.service' juga.



2. Setelah itu, buatlah *class* pada *package* yang baru dibuat dengan nama *StudentServiceDatabaseTest* dan masukkan *import* dibawah ini:

Update 1.2:

import static org.assertj.core.internal.bytebuddy.matcher.ElementMatchers.is; dihapus

```
import com.tutorial.apap.junit.dao.StudentMapper;
import com.tutorial.apap.junit.model.StudentModel;
import org.junit.After;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import org.mockito.BDDMockito;
import org.mockito.Mock;
import org.mockito.Mockito;
import org.mockito.MockitoAnnotations;

import java.util.ArrayList;
import java.util.List;

import static org.hamcrest.Matchers.notNullValue;
import static org.hamcrest.core.IsEqual.equalTo;
import static org.junit.Assert.*;

public class StudentServiceDatabaseTest {

    private StudentService studentService = new StudentServiceDatabase();
```



```

@Mock
private StudentMapper studentMapper;

@Before
public void setUp(){
    MockitoAnnotations.initMocks(this);
    this.studentService = new StudentServiceDatabase(this.studentMapper);
}

@Test
public void selectStudent() {
    // Given
    StudentModel studentModel = new StudentModel("1506737823", "Chanek", 3.5);
    StudentModel check = new StudentModel("1506737823", "Chanek", 3.5);
    BDDMockito.given(studentMapper.selectStudent("1506737823")).willReturn(studentModel);

    // When
    StudentModel test = studentService.selectStudent("1506737823");

    // Then
    assertThat(test, notNullValue()); // Check if Not Null
    assertThat(test, equalTo(check)); // Check if Same
}
}

```

Berdasarkan *code* diatas, dapat dilihat bahwa pada di awal terdapat inisialisasi Database. Dan terdapat anotasi `@Mock` pada `StudentMapper` yang berguna untuk membuat *dummy* pada `StudentMapper` agar hasil dari `StudentMapper` dapat dimanipulasi sesuai dengan *test* yang ingin dilakukan.

Lalu terdapat anotasi `@Before` yang digunakan untuk melakukan persiapan sebelum *testing*. Dan pada *method* ini terdapat `MockitoAnnotations.initMocks()` yang digunakan untuk mengaktifkan semua anotasi `Mock` yang terdapat pada kelas ini. Dan terdapat inisialisasi `StudentServiceDatabase` dengan `StudentMapper` yang telah di *mock*.

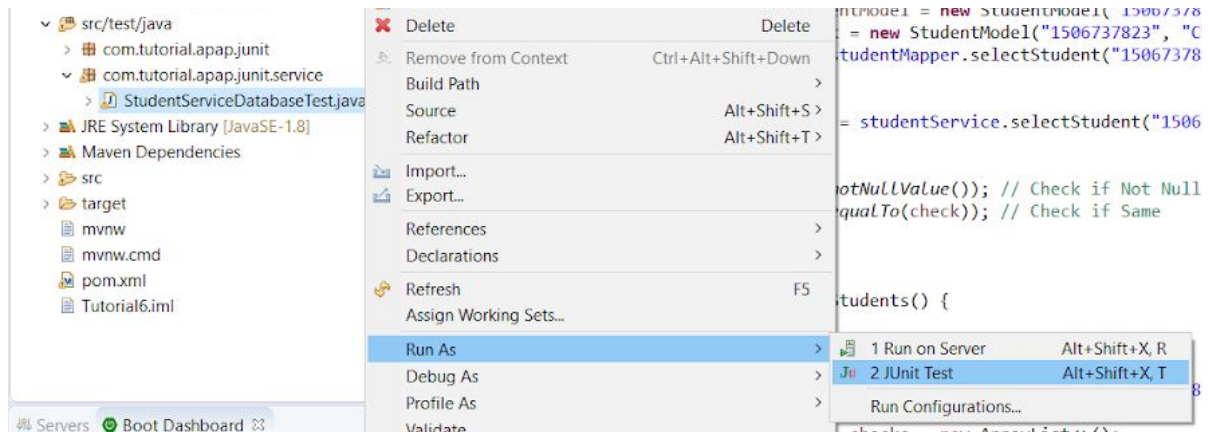
Lalu pada *method* `selectStudent` terdapat 3 segmen yaitu *Given*, *When*, dan *Then* dimana hal ini merupakan *logic* utama yang akan dilakukan pada setiap *testing*.

- Pada *Given*, pengguna akan menginisialisasi objek yang akan dites dan juga objek yang akan jadi parameter pengecekannya. Setelah itu `BDDMockito.given()` akan melakukan manipulasi terhadap *function* yang akan dipanggil dengan *return value* sesuai dengan isi dari `willReturn()`.
- Lalu pada *When* akan dilakukan pengetesan dengan *method* yang ingin digunakan. Pada *code* diatas kita akan mengetes apakah `selectStudent()` akan berjalan dengan memasukkannya pada objek baru..
- Dan pada *Then* akan dilakukan pengecekan terhadap objek baru apakah dia *null* ataupun sesuai dengan objek *check* yang telah disiapkan sebelumnya.

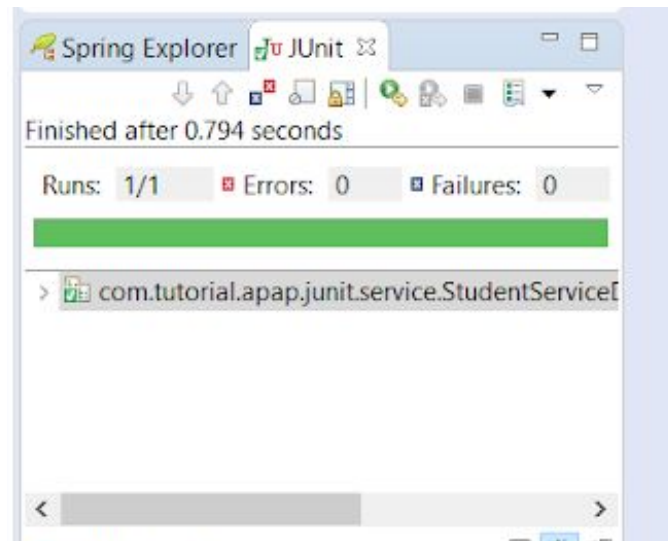
3. *Test* telah berhasil dibuat!

Menjalankan *testing* pada Spring Boot

1. Setelah itu run JUnit nya dengan klik kanan pada kelas *test* diatas dan run as JUnit Test.



2. Lalu hasil *test* akan berjalan dan hasilnya dapat dilihat pada bagian kanan STS.



3. *Test* telah berhasil dilakukan!

Latihan Unit Testing

1. Tambahkan *code* berikut pada `StudentServiceDatabaseTest` dan **jelaskan apa yang terjadi pada *method* tersebut berikut dengan hasil *testing*-nya!** (*Screenshot* hanya opsional dan tidak akan mempengaruhi nilai)

```
@Test
public void selectAllStudents() {

    // Given
    List<StudentModel> studentModels = new ArrayList<>();
    StudentModel studentModel = new StudentModel("1506737823", "Chanek", 3.5);
    studentModels.add(studentModel);

    List<StudentModel> checks = new ArrayList<>();
    StudentModel check = new StudentModel("1506737823", "Chanek", 3.5);
    checks.add(check);

    BDDMockito.given(studentMapper.selectAllStudents()).willReturn(studentModels);

    // When
    List<StudentModel> test = studentService.selectAllStudents();

    // Then
    assertThat(test, notNullValue()); // Check if Not Null
    assertThat(test.isEmpty(), equalTo(false)); // Check kalo ngga kosong
    assertThat(test.size(), equalTo(1)); // Check if Size same
    assertThat(test, equalTo(checks)); // Check kalo konten sama
}
```

2. Tambahkan *code* berikut pada `StudentServiceDatabaseTest` dan **jelaskan apa yang terjadi pada *method* tersebut berikut dengan hasil *testing*-nya!** (*Screenshot* hanya opsional dan tidak akan mempengaruhi nilai)

```
@Test
public void addStudent() {

    // Given
    StudentModel studentModel = new StudentModel("1506737823", "Chanek", 3.5);
    StudentModel check = new StudentModel("1506737823", "Chanek", 3.5);
    BDDMockito.given(studentService.addStudent(studentModel)).willReturn(true);

    // When
    boolean test = studentService.addStudent(studentModel);

    // Then
    BDDMockito.then(studentMapper).should().addStudent(check);
    assertThat(test, equalTo(true)); // Check if Same
}
```

3. Lakukan *testing* terhadap *method delete student* dan *update student*! (cukup lampirkan *screenshot* akhir dari *testing*)

Load Testing

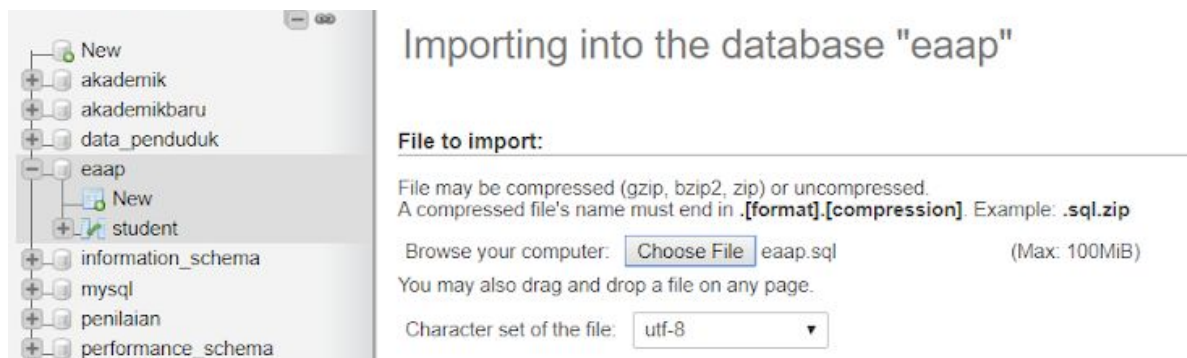
Penjelasan

Load testing merupakan sebuah proses dalam *software engineering* untuk melakukan pengujian respon dari sebuah sistem atau aplikasi dengan memberikan *request* tertentu. *Load testing* dilakukan untuk mengetahui performa dan respon dari suatu sistem atau aplikasi baik dalam kondisi respon normal maupun dalam kondisi respon yang ekstrim. *Load testing* dapat membantu kita untuk menentukan kapasitas maksimum dari sistem atau aplikasi yang kita buat.

Pada tutorial kali ini Anda akan mencoba untuk mengukur performa sebuah aplikasi web sederhana berdasarkan hasil tutorial sebelum ini (*Unit Testing*).

Tutorial

Sebelum melakukan *testing*, harap unduh SQL yang terdapat pada *scele*. Dan *import sql* menuju PHPMyAdmin maupun *client* lainnya pada *database* yang telah dibuat. Pada contoh tertulis apabila kita menggunakan *database* dengan nama eaap.



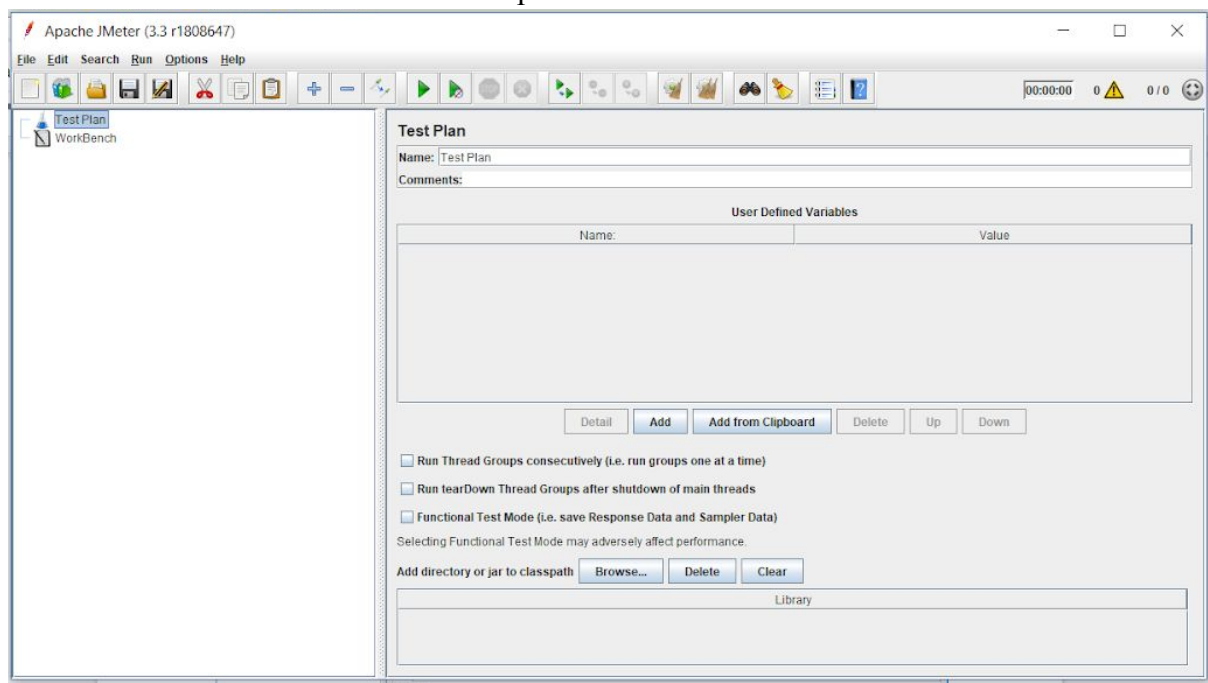
Setelah melakukan hal ini, jumlah *row* pada tabel *student* akan bertambah.

Load Testing Dengan Apache JMeter

Pada dunia nyata, tentunya sebuah *web* akan diakses oleh banyak pengguna di waktu yang bersamaan. Pada sesi kuliah, Anda sudah mempelajari bahwa salah satu parameter yang digunakan untuk mengukur performa adalah dengan *response time*. Oleh karena itu, Anda akan melakukan *load testing* untuk mengukur performa web yang Anda buat. *Load testing* akan mengakses web Anda secara bersamaan (*multithreading*) dengan parameter yang telah ditentukan dan menghitung *response time*-nya. Pada tutorial ini kita akan menggunakan aplikasi Apache Jmeter untuk melakukan *load testing*. Tahapannya adalah sebagai berikut:

1. Unduh program Apache JMeter pada http://jmeter.apache.org/download_jmeter.cgi
2. *Extract* file yang didownload dan jalankan ApacheJMeter.jar yang ada pada folder apache-jmeter-3.0\bin

Tampilan Awal Jmeter



3. Pada tulisan Test Plan, klik kanan, Add > Threads (Users) > Thread Group
4. Properti yang perlu diisi adalah:
 - *Number of Threads (Users)*
Jumlah *thread* yang akan mengakses web Anda
 - *Ramp-up Period*
Waktu yang dibutuhkan (dalam detik) untuk mencapai jumlah *thread* yang diinginkan. Jika jumlah *thread* = 10 dan *ramp-up period* = 5, artinya dibutuhkan 5 detik sampai jumlah *thread* menjadi 10. Dengan kata lain, *thread* baru akan muncul setiap 0.5 detik.
 - *Loop Count*
Jumlah percobaan dilakukan

Thread Properties

Number of Threads (users): 50

Ramp-Up Period (in seconds): 10

Loop Count: ☐ Forever 1

☐ Delay Thread creation until needed

☐ Scheduler

5. Selanjutnya klik kanan pada Thread Group, Add > Sampler > HTTP Requests. Isikan Server Name dengan localhost dan path ke program Anda.

HTTP Request

Name: HTTP Request

Comments:

Basic Advanced

Web Server

Protocol [http]: Server Name or IP: localhost Port Number: 8080

HTTP Request

Method: GET Path: student/viewall Content encoding:

☐ Redirect Automatically ☒ Follow Redirects ☒ Use KeepAlive ☐ Use multipart/form-data for POST ☐ Browser-compatible headers

Parameters Body Data Files Upload

6. Untuk memantau hasil, klik kanan pada Test Plan, Add > Listener > View Results in Table

7. Selanjutnya tekan tombol  untuk memulai load testing.

8. Tekan View Results in Table untuk melihat hasil.

Apache JMeter (3.3 r1808647)

File Edit Search Run Options Help

Test Plan

- Thread Group
 - HTTP Request
 - View Results in Table
 - WorkBench

View Results in Table

Name: View Results in Table

Comments:

Write results to file / Read from file

Filename: Browse...

Log/Display Only: ☐ Errors ☐ Successes ☐ Configure

Sample #	Start Time	Thread Name	Label	Sample Time	Status	Bytes	Sent Bytes	Latency	Connec Tim
1	09.49.22.510	Thread Grou...	HTTP Request	1416	✓	383	151	1416	1
2	09.49.22.910	Thread Grou...	HTTP Request	1018	✓	383	151	1018	1
3	09.49.22.709	Thread Grou...	HTTP Request	1227	✓	383	151	1227	1
4	09.49.22.304	Thread Grou...	HTTP Request	1650	✓	383	151	1650	1
5	09.49.23.110	Thread Grou...	HTTP Request	1107	✓	383	151	1107	0
6	09.49.23.711	Thread Grou...	HTTP Request	1355	✓	383	151	1355	1
7	09.49.23.511	Thread Grou...	HTTP Request	1563	✓	383	151	1563	1
8	09.49.24.113	Thread Grou...	HTTP Request	963	✓	383	151	963	0
9	09.49.23.311	Thread Grou...	HTTP Request	1789	✓	383	151	1789	1
10	09.49.23.911	Thread Grou...	HTTP Request	1180	✓	383	151	1180	1
11	09.49.24.712	Thread Grou...	HTTP Request	1193	✓	383	151	1193	1
12	09.49.24.512	Thread Grou...	HTTP Request	1403	✓	383	151	1403	0
13	09.49.24.312	Thread Grou...	HTTP Request	1619	✓	383	151	1619	1
14	09.49.24.913	Thread Grou...	HTTP Request	1030	✓	383	151	1030	1
15	09.49.25.113	Thread Grou...	HTTP Request	833	✓	383	151	833	1
16	09.49.25.916	Thread Grou...	HTTP Request	2366	✓	383	151	2366	0
17	09.49.25.715	Thread Grou...	HTTP Request	2599	✓	383	151	2599	1
18	09.49.26.316	Thread Grou...	HTTP Request	2002	✓	383	151	2002	1
19	09.49.26.314	Thread Grou...	HTTP Request	3050	✓	383	151	3050	0
20	09.49.26.515	Thread Grou...	HTTP Request	2855	✓	383	151	2855	1
21	09.49.26.116	Thread Grou...	HTTP Request	2357	✓	383	151	2357	0
22	09.49.26.916	Thread Grou...	HTTP Request	3162	✓	383	151	3162	0
23	09.49.26.715	Thread Grou...	HTTP Request	3399	✓	383	151	3399	1
24	09.49.26.516	Thread Grou...	HTTP Request	3662	✓	383	151	3662	1
25	09.49.27.116	Thread Grou...	HTTP Request	3625	✓	383	151	3625	0

☐ Scroll automatically? ☐ Child samples? No of Samples 50 Latest Sample 1781 Average 2128 Deviation 793

Yang perlu Anda perhatikan adalah kolom Sample Time (ms). *Sample time* merupakan selisih waktu dari mengirim *request* sampai mendapatkan respons. Perhatikan bahwa pada contoh di atas, awalnya *sample time* sekitar 5 sekon. Namun, setelah jumlah *thread* bertambah dan makin banyak yang mengakses *sample time* bertambah besar mencapai sekitar 20-26 sekon. Pada dunia nyata dimana web akan diakses ribuan sampai jutaan orang di waktu yang bersamaan, tentunya web yang Anda buat akan sangat lambat atau bahkan menyebabkan *server down*.

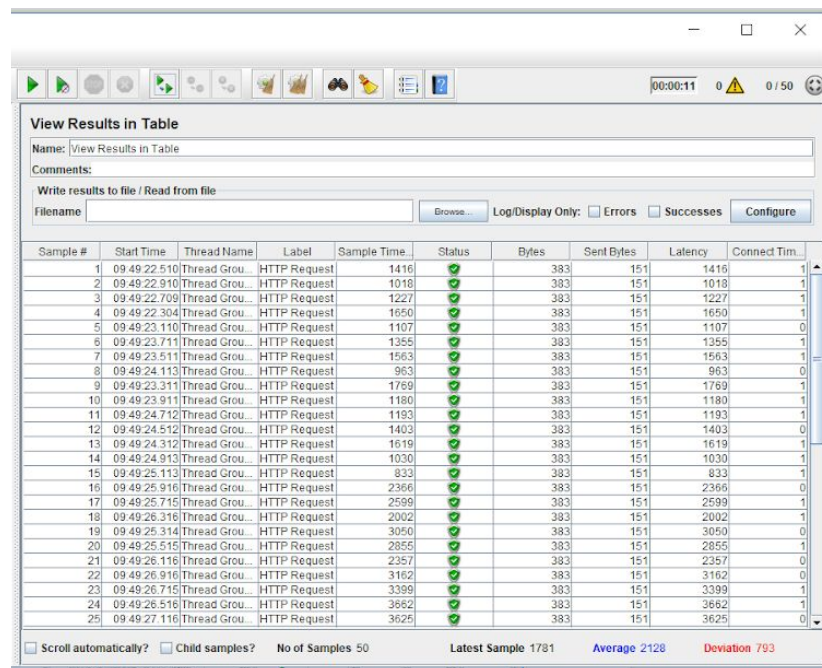
Catatan: Perhatikan bahwa hasil di atas dapat berbeda-beda tergantung pada performa komputer Anda masing-masing.

9. Tugas Anda adalah melakukan **langkah optimisasi** (bisa dari web, query, ataupun basis data) agar *sample time* menjadi jauh lebih cepat. Silahkan lakukan pengujian kembali dengan menggunakan JMeter setelah program Anda dioptimasi.

Latihan Load Testing

Soal latihan:

1. Lakukan load testing pada semua kemungkinan hasil dari fitur **selectStudent()** dengan kondisi **berhasil ditemukan**, **selectStudent()** dengan kondisi **gagal ditemukan**, dan **selectAllStudents()** menggunakan Apache JMeter. Gunakan Number of Threads sama dengan 100 dan Ramp-up period 10 detik. Anda berarti perlu melakukan 3x test, 1 test untuk setiap kemungkinan hasil. Screenshot window JMeter Anda. Contoh:



View Results in Table

Name: View Results in Table

Comments:

Write results to file / Read from file

Filename: Browse...

Log/Display Only: ☐ Errors ☐ Successes

Sample #	Start Time	Thread Name	Label	Sample Time	Status	Bytes	Sent Bytes	Latency	Connected Time
1	09.49.22.510	Thread Gro...	HTTP Request	1416	Success	383	151	1416	1
2	09.49.22.910	Thread Gro...	HTTP Request	1018	Success	383	151	1018	1
3	09.49.22.709	Thread Gro...	HTTP Request	1227	Success	383	151	1227	1
4	09.49.22.304	Thread Gro...	HTTP Request	1650	Success	383	151	1650	1
5	09.49.23.110	Thread Gro...	HTTP Request	1107	Success	383	151	1107	0
6	09.49.23.711	Thread Gro...	HTTP Request	1355	Success	383	151	1355	1
7	09.49.23.511	Thread Gro...	HTTP Request	1563	Success	383	151	1563	1
8	09.49.24.113	Thread Gro...	HTTP Request	963	Success	383	151	963	0
9	09.49.23.311	Thread Gro...	HTTP Request	1769	Success	383	151	1769	1
10	09.49.23.911	Thread Gro...	HTTP Request	1180	Success	383	151	1180	1
11	09.49.24.712	Thread Gro...	HTTP Request	1193	Success	383	151	1193	1
12	09.49.24.512	Thread Gro...	HTTP Request	1403	Success	383	151	1403	0
13	09.49.24.312	Thread Gro...	HTTP Request	1619	Success	383	151	1619	1
14	09.49.24.913	Thread Gro...	HTTP Request	1030	Success	383	151	1030	1
15	09.49.25.113	Thread Gro...	HTTP Request	833	Success	383	151	833	1
16	09.49.25.916	Thread Gro...	HTTP Request	2366	Success	383	151	2366	0
17	09.49.25.715	Thread Gro...	HTTP Request	2599	Success	383	151	2599	1
18	09.49.26.316	Thread Gro...	HTTP Request	2002	Success	383	151	2002	1
19	09.49.25.314	Thread Gro...	HTTP Request	3050	Success	383	151	3050	0
20	09.49.25.515	Thread Gro...	HTTP Request	2855	Success	383	151	2855	1
21	09.49.26.116	Thread Gro...	HTTP Request	2357	Success	383	151	2357	0
22	09.49.26.916	Thread Gro...	HTTP Request	3162	Success	383	151	3162	0
23	09.49.26.715	Thread Gro...	HTTP Request	3399	Success	383	151	3399	1
24	09.49.26.516	Thread Gro...	HTTP Request	3662	Success	383	151	3662	1
25	09.49.27.116	Thread Gro...	HTTP Request	3625	Success	383	151	3625	0

☐ Scroll automatically? ☐ Child samples? No of Samples 50 Latest Sample 1781 Average 2128 Deviation 793

2. Lakukan optimasi yang mungkin dilakukan baik dari sisi kode yang Anda tuliskan maupun dari sisi database agar hasil load testing dari Apache JMeter bisa lebih cepat. Anda diperbolehkan untuk mengubah beberapa aspek di database seperti index, primary/unique key, dll namun tidak mengubah struktur tabel dan data pada database. Anda bisa mencari tahu dari sumber lain di internet untuk mengetahui apa saja hal yang bisa dioptimasi.
3. Lakukan load testing kembali untuk semua kemungkinan hasil dengan kode dan database yang sudah Anda optimasi. Gunakan Number of Threads sama dengan 100 dan Ramp-up period 10 detik. Screenshot kembali window JMeter Anda.
4. Tulislah ide optimasi Anda, proses Anda melakukan optimasi, perbandingan hasil JMeter sebelum dan sesudah optimasi, hasil dari optimasi Anda apakah berhasil atau tidak, dan hal lainnya yang perlu Anda jelaskan tentang optimasi tersebut pada kode dan database Anda dalam file pdf bernama **tutorial6_writeup.pdf**. Anda wajib menuliskan write-up.

Deliverables

Deliverables dari tutorial ini adalah:

- **File Project yang sudah dioptimasi.**
Buat sebuah **project baru** pada organization /**apap-ekstensi-2018** dengan format nama **tutorial6_NPM**, contoh **tutorial6_1501234567**. Push project Anda ke *repository* GitHub tersebut. **Pastikan anda telah mengerjakan latihan, tidak salah dalam melakukan push repository, tidak melakukan commit menggunakan akun GitHub orang lain, serta tidak melebihi batas waktu yang di telah ditentukan!.**
- Screenshot hasil JMeter fitur soal latihan *Load Testing* nomor 1 sebelum dioptimasi (3 file)
 - jmeter_selectStudentBerhasil
 - jmeter_selectStudentGagal
 - jmeter_selectAllStudents
- Screenshot hasil JMeter fitur soal latihan *Load Testing* nomor 2 setelah dioptimasi (3 file)
 - jmeter_selectStudentBerhasil
 - jmeter_selectStudentGagal
 - jmeter_selectAllStudents
- Screenshot hasil *testing* pada latihan *Unit Testing*. (Cukup satu yang menggambarkan seluruh *testing* berjalan)
- **Write-up**
Buat sebuah file write-up. Jelaskan apa saja **hal yang Anda pelajari** dari tutorial ini. Cantumkan juga penjelasan Anda terhadap **setiap latihan** yang ada di tutorial!

Kami rekomendasikan Anda menggunakan format pdf agar dapat lebih leluasa dalam menuangkan penjelasan Anda dengan dukungan *screenshot* dan kode. Masukkan *file writeup* ke *folder project*. Pastikan *file write-up* juga di-*push* ke *repository*.

Pengumpulan

Buat sebuah **project baru** pada organization /**apap-ekstensi-2018** dengan format nama **tutorial6_NPM**, contoh **tutorial6_1501234567**. Push project Anda ke *repository* GitHub tersebut. **Pastikan anda telah mengerjakan latihan, tidak salah dalam melakukan push repository, tidak melakukan commit menggunakan akun GitHub orang lain, serta tidak melebihi batas waktu yang di telah ditentukan!.**

Deadline

24 Maret 2018 23:59:59

Penalti

Penalti:

- Keterlambatan

Penalti keterlambatan sebesar -10 poin akan ditambahkan setiap 10 menit keterlambatan