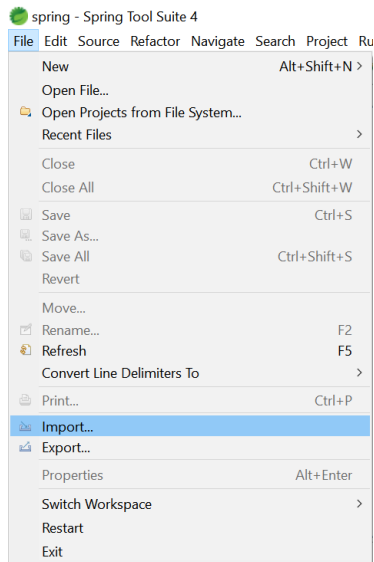


Tutorial 06 – Test

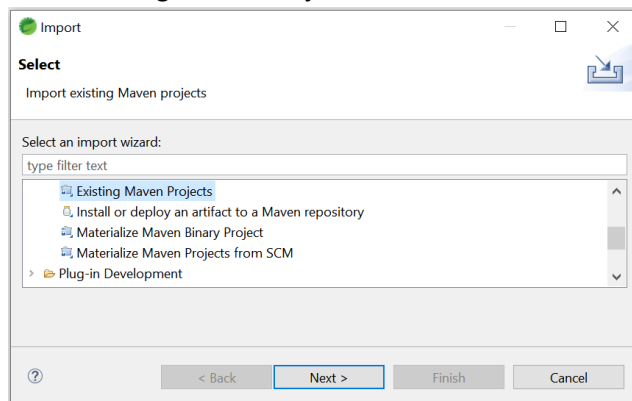
Tutorial ini mengasumsikan penggunaan pada *Windows*, tanpa melarang penggunaan OS lain, silakan menyesuaikan.

1. Pendahuluan

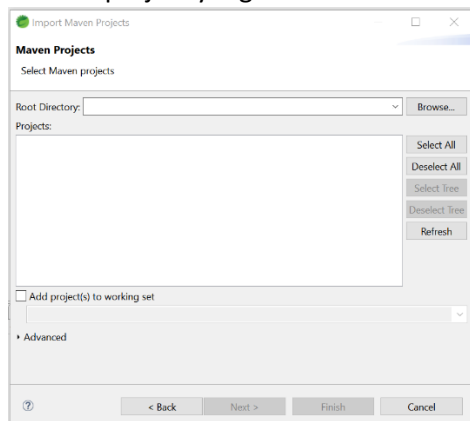
- Unduh Maven Project yang ada di Scele lalu ekstrak.
- Jalankan STS
- Klik File >> Import



- Pilih “Existing Maven Project” >> Next



- Browse project yang sudah diekstrak >> Finish, tunggu sampai proses *import* selesai.



- Sesuaikan **application.properties** dengan *environment* Anda. DB bisa gunakan tu04/tu05.

2. Instalasi JUnit dan Mockito

- Anda tidak perlu menambah dependency JUnit dan Mockito jika sudah memiliki dependency “spring-boot-starter-test” pada pom.xml project Anda.

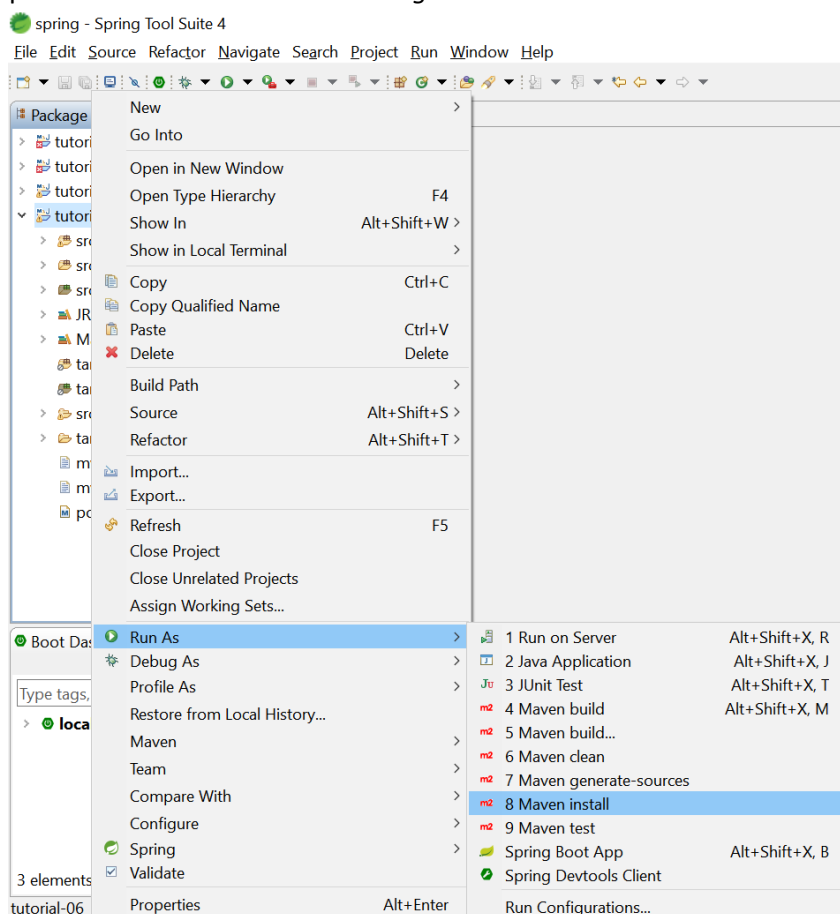
```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
  <scope>test</scope>
</dependency>
```

- Jika dependency diatas tidak ada maka perlu menambahkan dependency sendiri, misalnya:

```
<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>
```

- Lakukan Maven Install, klik kanan pada project >> Run As >> Maven install, tunggu hingga proses selesai. Pastikan DB *running*!



3. Testing untuk JPA (Repository)

- Buatlah *package* untuk *repository* pada folder `src/test/java`. Nama *package* sama dengan *package* pada folder `src/main/java`.

tutorial-06 [boot] [devtools]

```
▼ src/main/java
  > com.apap.tu06
  > com.apap.tu06.controller
  > com.apap.tu06.model
  > com.apap.tu06.repository
  > com.apap.tu06.service
> src/main/resources
▼ src/test/java
  > com.apap.tu06
    com.apap.tu06.repository
```

- Pada *package* tersebut buat *class* dengan nama `FlightDbTest` dengan spesifikasi sbb.:

```
package com.apap.tu06.repository;
```

```
import static org.junit.Assert.assertThat;
```

```
import java.sql.Date;
import java.util.Optional;
```

```
import com.apap.tu06.model.FlightModel;
import com.apap.tu06.model.PilotModel;

import org.hamcrest.Matchers;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.jdbc.AutoConfigureTestDatabase;
import org.springframework.boot.test.autoconfigure.jdbc.AutoConfigureTestDatabase.Replace;
import org.springframework.boot.test.autoconfigure.orm.jpa.DataJpaTest;
import org.springframework.boot.test.autoconfigure.orm.jpa.TestEntityManager;
import org.springframework.test.context.junit4.SpringRunner;
```

```
@RunWith(SpringRunner.class)
@DataJpaTest
@AutoConfigureTestDatabase(replace = Replace.NONE)
public class FlightDbTest {
    @Autowired
    private TestEntityManager entityManager;

    @Autowired
    private FlightDb flightDb;

    @Test
    public void whenFindByFlightNumber_thenReturnFlight() {
        // Given
        PilotModel pilotModel = new PilotModel();
        pilotModel.setLicenseNumber("1234");
        pilotModel.setName("Anto");
        pilotModel.setFlyHour(50);
        entityManager.persist(pilotModel);
        entityManager.flush();
        FlightModel flightModel = new FlightModel();
        flightModel.setFlightNumber("I765");
        flightModel.setOrigin("Jakarta");
        flightModel.setDestination("Bali");
        flightModel.setTime(new Date(new java.util.Date().getTime()));
        flightModel.setPilot(pilotModel);
        entityManager.persist(flightModel);
        entityManager.flush();

        // When
        Optional<FlightModel> found = flightDb.findByFlightNumber(flightModel.getFlightNumber());

        // Then
        assertThat(found.get(), Matchers.notNullValue()); // Check if not null
        assertThat(found.get(), Matchers.equalTo(flightModel)); // Check if same
    }
}
```

Terdapat 3 segmen yaitu **Given**, **When**, dan **Then** dimana hal ini merupakan logic utama yang akan dilakukan pada setiap testing:

- Given, inisiasi objek yang akan menjadi sebuah entitas pada database untuk dites.
- When, dilakukan pemanggilan method yang ingin dites.
- Then, dilakukan pengecekan terhadap objek baru apakah dia null atau sesuai dengan objek awal yang telah diinisiasi yang telah disiapkan sebelumnya.

Pengujian diatas dilakukan untuk mengetes apakah fungsi findByFlightNumber() menghasilkan object yang sesuai dengan ekspektasi.

- 1) Mengapa perlu menginisiasi object PilotModel, sedangkan yang di test hanya FlightModel?
- 2) Jelaskan apa yang akan terjadi jika object PilotModel dihapus dan tidak dilakukan setPilot pada FlightModel?

4. Testing untuk Service

- Buatlah *package* untuk *service* pada folder src/test/java. Nama *package* sama dengan package pada folder src/main/java.

tutorial-06 [boot] [devtools]

```
▼ src/main/java
  > com.apap.tu06
  > com.apap.tu06.controller
  > com.apap.tu06.model
  > com.apap.tu06.repository
  > com.apap.tu06.service
  > src/main/resources
▼ src/test/java
  > com.apap.tu06
  > com.apap.tu06.repository
  com.apap.tu06.service
```

- Pada package tersebut buat *class* dengan nama FlightServiceTest dengan spesifikasi sbb.:
package com.apap.tu06.service;

```
import static org.junit.Assert.assertThat;

import java.sql.Date;
import java.util.Optional;

import com.apap.tu06.model.FlightModel;
import com.apap.tu06.repository.FlightDb;

import org.hamcrest.Matchers;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.mockito.Mockito;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.TestConfiguration;
import org.springframework.boot.test.mock.mockito.MockBean;
import org.springframework.context.annotation.Bean;
import org.springframework.test.context.junit4.SpringRunner;
```

```

@RunWith(SpringRunner.class)
public class FlightServiceTest {
    @Autowired
    private FlightService flightService;

    @MockBean
    private FlightDb flightDb;

    @TestConfiguration // Membatasi scope Bean yang didefinisikan menjadi local class
    static class FlightServiceTestContextConfiguration {
        @Bean // Initiate flightService sebagai Bean
        public FlightService flightService() {
            return new FlightServiceImpl();
        }
    }

    @Test
    public void whenValidFlightNumber_thenFlightShouldBeFound() {
        // Given
        FlightModel flightModel = new FlightModel();
        flightModel.setFlightNumber("I765");
        flightModel.setOrigin("Jakarta");
        flightModel.setDestination("Bali");
        flightModel.setTime(new Date(new java.util.Date().getTime()));
        Optional<FlightModel> flight = Optional.of(flightModel);
        Mockito.when(flightDb.findByFlightNumber(flight.get().getFlightNumber())).thenReturn(flight);

        // When
        Optional<FlightModel> found = flightService.getFlightDetailByFlightNumber(flight.get().getFlightNumber());

        // Then
        assertThat(found, Matchers.notNullValue()); // Check if not null
        assertThat(found.get().getFlightNumber(), Matchers.equalTo(flightModel.getFlightNumber())); //Check if same
    }
}

```

Anotasi `@MockBean` pada `FlightDb` berguna untuk membuat dummy pada `FlightDb` agar hasil dari pemanggilan method JPA dapat dimanipulasi sesuai dengan test yang ingin dilakukan. Anotasi `@TestConfiguration` dan `@Bean` berguna untuk menginisiasi `FlightService` yang merupakan interface dengan class implementasinya, serta membatasi scope penggunaan dari `FlightService` tersebut.


Pengujian diatas dilakukan untuk mengetes apakah fungsi `getFlightDetailByFlightNumber()` menghasilkan object yang sesuai dengan ekspektasi, dan apakah manipulasi (jika ada) yang dilakukan diservice berjalan dengan semestinya.













3) Jelaskan apa yang dilakukan oleh code

```
Mockito.when(flightDb.findByFlightNumber(flight.get().getFlightNumber())).thenReturn(flight);
```

5. Testing untuk Controller

- Buatlah *package* untuk *controller* pada folder `src/test/java`. Nama *package* sama dengan package pada folder `src/main/java`.

 tutorial-06 [boot] [devtools]

- ▼  src/main/java
 - >  com.apap.tu06
 - >  com.apap.tu06.controller
 - >  com.apap.tu06.model
 - >  com.apap.tu06.repository
 - >  com.apap.tu06.service
 - >  src/main/resources
- ▼  src/test/java
 - >  com.apap.tu06
 -  com.apap.tu06.controller
 - >  com.apap.tu06.repository
 - >  com.apap.tu06.service

- Pada package tersebut buat *class* dengan nama `FlightControllerTest` dengan spesifikasi sbb.:
`package` com.apap.tu06.controller;

```
import java.sql.Date;
import java.util.Optional;

import com.apap.tu06.model.FlightModel;
import com.apap.tu06.service.FlightService;
import com.apap.tu06.service.PilotService;

import org.hamcrest.Matchers;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.mockito.Mockito;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;
import org.springframework.boot.test.mock.mockito.MockBean;
import org.springframework.http.MediaType;
import org.springframework.test.context.junit4.SpringRunner;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.request.MockMvcRequestBuilders;
import org.springframework.test.web.servlet.result.MockMvcResultMatchers;

@RunWith(SpringRunner.class)
@WebMvcTest(FlightController.class)
public class FlightControllerTest {
    @Autowired
    private MockMvc mvc;

    @MockBean
    private FlightService flightService;

    @MockBean
    private PilotService pilotService;

    @Test
    public void givenFlightNumber_whenViewFlight_thenReturnJsonFlight() throws Exception {
        // Given
        FlightModel flightModel = new FlightModel();
        flightModel.setFlightNumber("I765");
        flightModel.setOrigin("Jakarta");
        flightModel.setDestination("Bali");
        flightModel.setTime(new Date(new java.util.Date().getTime()));
        Optional<FlightModel> flight = Optional.of(flightModel);
        Mockito.when(flightService.getFlightDetailByFlightNumber(flight.get().getFlightNumber())).thenReturn(flight);

        // When
        mvc.perform(MockMvcRequestBuilders.get("/flight/view")
            .param("flightNumber", flight.get().getFlightNumber())
            .contentType(MediaType.APPLICATION_JSON))
        // Then
        .andExpect(MockMvcResultMatchers.status().isOk())
        .andExpect(MockMvcResultMatchers.jsonPath("$.flightNumber", Matchers.is(flight.get().getFlightNumber())));
    }
}
```

Pengujian diatas dilakukan untuk mengetes apakah route “/flight/view” mengembalikan hasil yang sesuai dengan ekspektasi.

4) Jelaskan apa yang dilakukan oleh code

```
Mockito.when(flightService.getFlightDetailByFlightNumber(flight.get().getFlightNumber())).thenReturn(flight);
```

5) Jelaskan apa yang ditest oleh code

```
.andExpect(MockMvcResultMatchers.status().isOk())
```

6) Jelaskan apa yang ditest oleh code

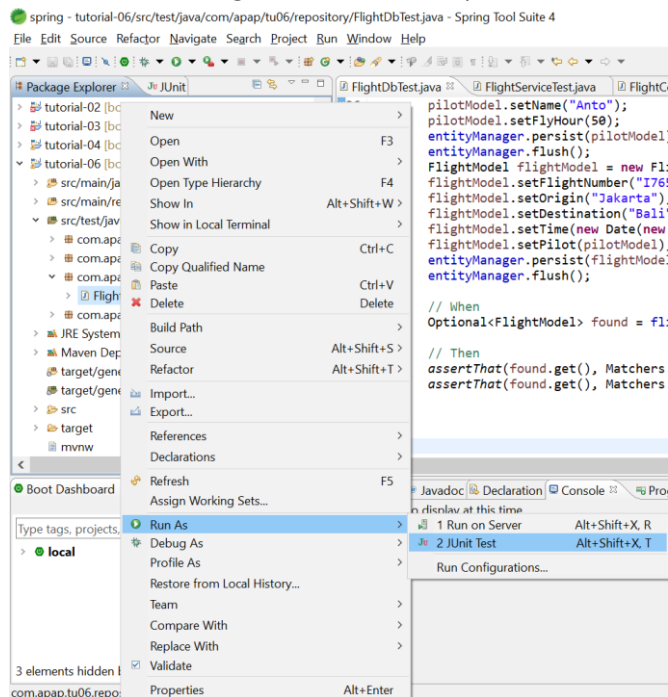
```
.andExpect(MockMvcResultMatchers.jsonPath("$.flightNumber", Matchers.is(flight.get().getFlightNumber())));
```

7) Jelaskan anotasi `@ResponseBody` yang ada pada route “/flight/view”

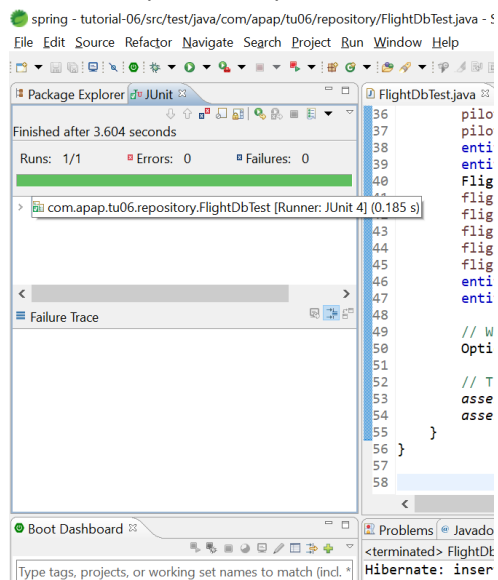
```
@RequestMapping(value = "/flight/view", method = RequestMethod.GET)
private @ResponseBody FlightModel view(@RequestParam(value = "flightNumber") String flightNumber, Model model) {
    FlightModel archive = flightService.getFlightDetailByFlightNumber(flightNumber).get();
    return archive;
}
```

6. Melakukan Testing (Unit Testing)

- Jalankan JUnit dengan cara klik kanan pada *class* >> Run As >> JUnit Test



- Hasil tes dapat dilihat pada tab JUnit

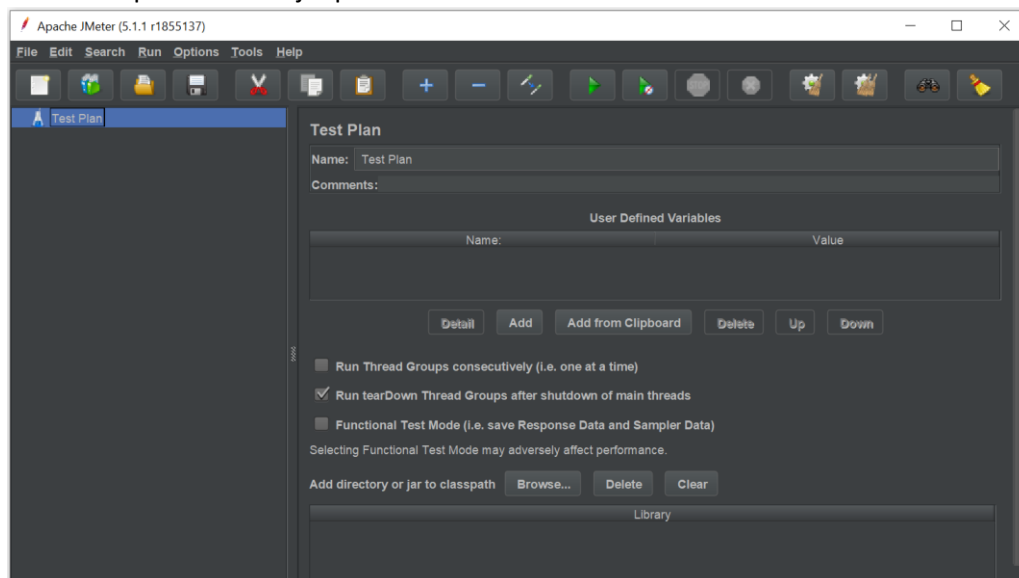


- Jalankan tes untuk semua *test class* yang sudah dibuat.

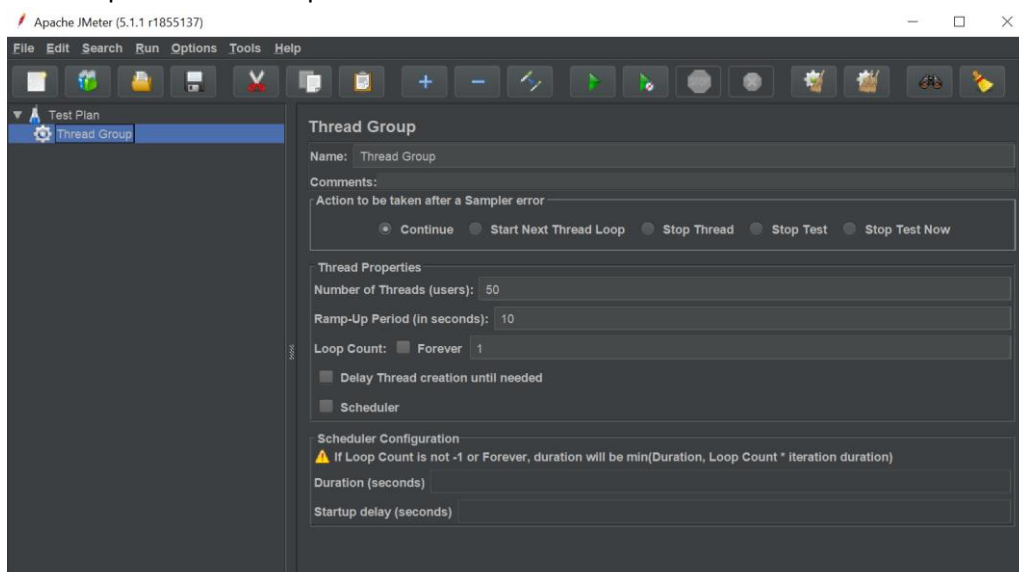
7. Load Testing – Apache JMeter

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.

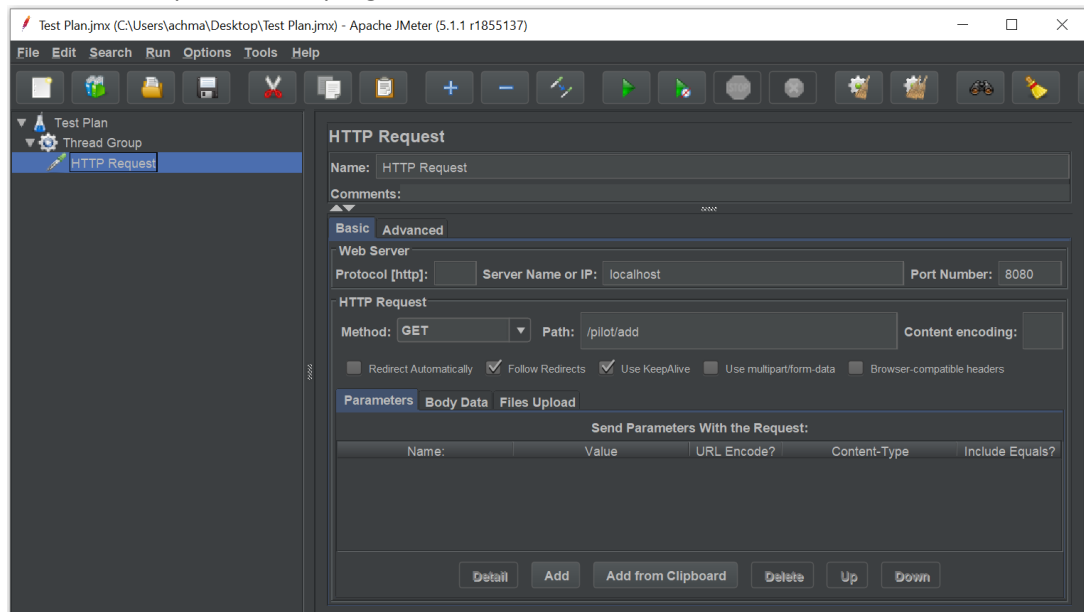
- Unduh SQL yang ada di Scele lalu import pada DB yang anda gunakan. Pastikan tidak ada ID yang bentrok, data yang ada pada DB dapat diubah atau dihapus terlebih dahulu.
- Unduh *binary release* (.zip) Apache JMeter pada link dibawah ini lalu ekstrak.
https://jmeter.apache.org/download_jmeter.cgi
- Jalankan ApacheJMeter.jar pada folder bin.



- Klik kanan pada Test Plan >> Add >> Threads (Users) >> Thread Group
- Properti yang perlu diisi (lihat gambar):
 - o Number of Threads (users): Jumlah *thread* yang akan mengakses *web* Anda.
 - o Ramp-Up Period (in seconds): Waktu yang dibutuhkan untuk mencapai jumlah *thread*. Jika *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.
 - o Loop Count: Jumlah percobaan dilakukan



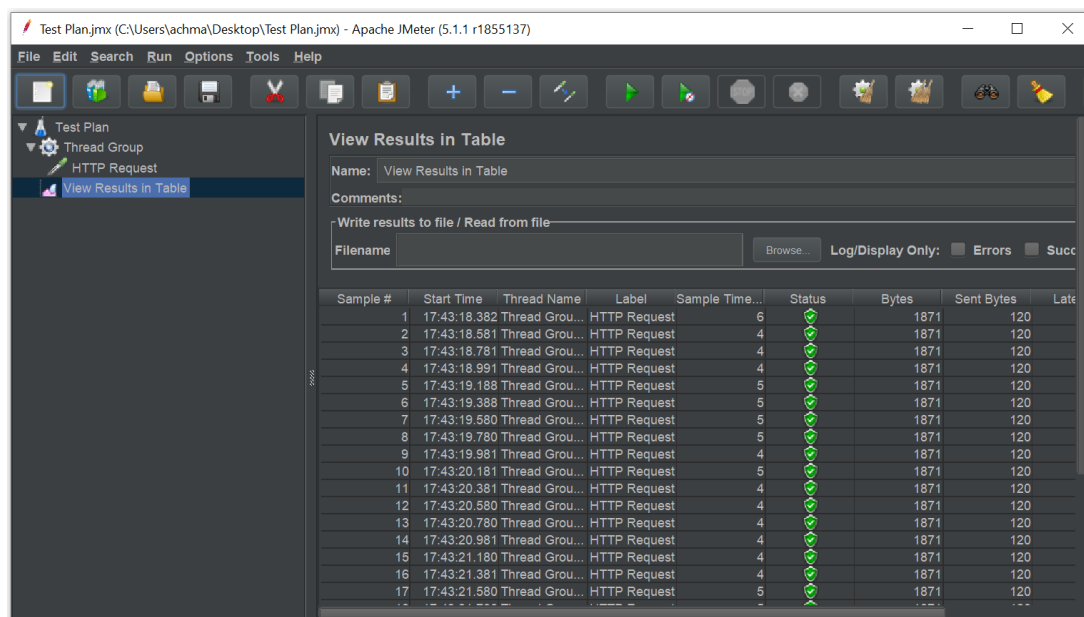
- Klik kanan pada Thread Group >> Add >> Sampler >> HTTP Request. Isi Server Name dengan localhost dan path/link ke program Anda.



- Untuk memantau hasil, klik kanan pada Test Plan >> Add >> Listener >> View Results in Table.
- Klik tombol *start* untuk memulai *load testing*.



- Pilih “View Results in Table” untuk melihat hasil.



Yang perlu Anda perhatikan adalah kolom Sample Time (ms). *Sample time* merupakan selisih waktu dari mengirim *request* sampai mendapatkan *respons*. Semakin besar jumlah *thread* atau semakin banyak akses maka *sample time* bertambah besar. 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:** Hasil di atas dapat berbeda-beda tergantung pada performa komputer Anda masing-masing.

- Lakukan 3 percobaan berbeda dengan mengganti *Number of Threads*, *Ramp-Up Period*, *Loop Count*, atau *link* yang diakses. Jelaskan percobaan anda pada laporan!

Pengumpulan

1. *Screen capture* pengerjaan tutorial yang menunjukkan bahwa Unit Testing (langkah 3-6) dan Load Testing (langkah 7, termasuk penjelasan dari 3 percobaan) berhasil diimplementasi dan dilakukan. Jawablah tujuh pertanyaan/perintah yang ditandai dengan warna hijau. Pengumpulan dituliskan dalam satu file dengan format **npm_nama-lengkap-anda.pdf** dan unggah ke submission slot yang disediakan di Scele.
2. Folder tutorial-06 dikumpulkan dengan cara push ke GitHub (<https://github.com/achmad-f-abka/apap>).