# App og API projektopgave i fagene App II og Java IV

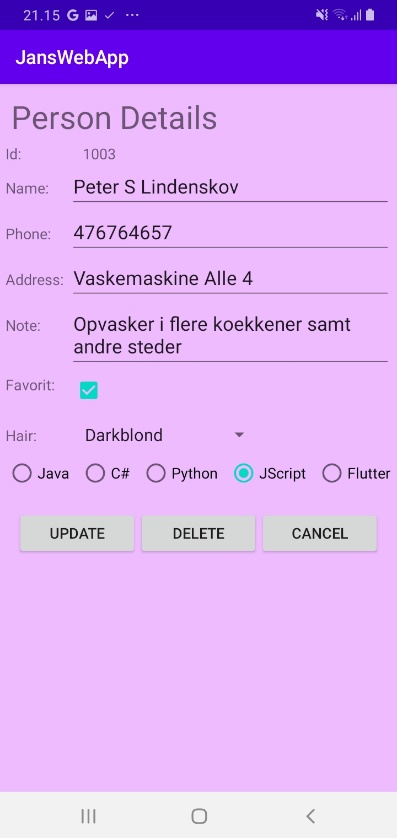
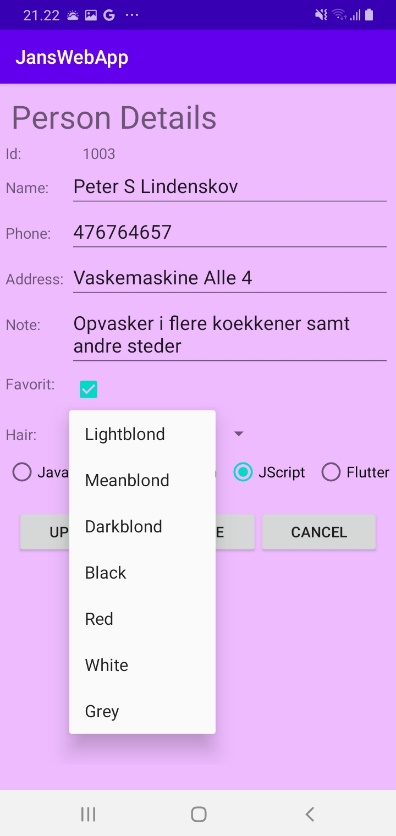
Der skal i denne projektopgave oprettes en database, et Web-API programmeret i Java, som benytter databasen, og en Android klient som benytter Web-API’et og som har en brugerflade som vist herunder.

## Android klient

På forsiden skal der være en customized liste af Personer, som i hvert item viser et uddrag af oplysninger om Personen. Personer der er favoritter skal vise en stjerne. Ved at klikke på stjernen eller der den skal være, skal der toggles imellem favorit og ikke-favorit.

Ved tryk på en Person i listen, skal der skiftes til en side der viser alle detaljer om Person. Hårfarven skal vises som en drop-down med alle muligheder og med den aktuelle valg. Mulighederne i listen skal hentes fra API’et . Programmeringssprog skal vises som radiobuttons med den aktuelle valgt. Radiobuttons skal oprettes dynamisk med mulighederne hentet fra API’et.

* Det skal være muligt at ændre oplysningerne og trykke på Update, hvorved ændringerne gemmes via API’et, og der vendes tilbage til forsiden og som viser en opdateret liste.
* Ved tryk på Delete skal der vises en dialog, hvor det bekræftes at Personen skal slettes, og der vendes tilbage til forsiden med en opdateret liste.
* Ved tryk på Cancel skal der vendes tilbage til forsiden uden at der er ændret noget.

Ved tryk på forsidens FAB + skal der skiftes til en side, hvor der kan indtastes og vælges oplysninger om en ny Person. Siden skal være som den ovenfor beskrevne detaljeside. Der skal ikke indtastes en Id på Personen, da den skal genereres af databasen. Når der trykkes på Add Person, skal den nye Person indsættes via API’et, som derefter skal levere den oprettede Persons Id, som så skrives i brugerfladen.

Ved tryk på Cancel skal der vendes tilbage til forsiden, som skal vise en opdateret liste. (Eller det samme som før, hvis der trykkes Cancel uden at der er tilføjet en ny Person)

## Database

Databasen skal indeholde en Person-tabel samt en Hårfarve-tabel og en Programmeringssprog-tabel, som begge refereres til fra Person-tabellen. Hårfarve-tabellen skal indeholde muligheder for hårfarve og Programmeringssprog-tabellen skal indeholde mulighederne for programmeringssprog.

## Web-API

Web-API’et skal have følgende metoder

* Get alle Personer
* Get en Person på id
* Post en ny Person – denne skal returnere id’et på den nyoprettede Person
* Delete en Person
* Put en ændring på en Person
* Get alle Hårfarver
* Get en Hårfarve på id
* Get alle Programmeringssprog
* Get et Programmeringssprog på id

## Et lettere afgrænset projekt

Hvis det skønnes at det beskrevne projekt bliver svært at nå, kan det afgrænses til kun at have en databasetabel for Person, hvor de tal der skulle referere til Hårfarve og Programmeringssprog konverteres til de tilsvarende betegnelser i Android-programmet. Så skal API’et kun håndtere Personer.  
Radiobuttons skal ikke oprettes dynamisk med mulighederne hentet fra API’et, men være et fast andtal og med faste benævnelser.

Desuden kan det udelades at lade API’et returnere id på en nyoprettet Person.

## Detaljer om Android klienten

For ikke at vise løsningen på opgaven, vises her detaljer som bruger klassen Frugt og ikke Person.

**AndroidManifest.xml**  
Her tilføjes de to linier vist med fed for at kunne bruge internet.

*<?*xml version="1.0" encoding="utf-8"*?>*<manifest xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:tools="http://schemas.android.com/tools">  
  
 **<uses-permission android:name="android.permission.INTERNET"/>**  
 <application  
 **android:usesCleartextTraffic="true"** android:allowBackup="true"

...

**build.gradle(Module :app)**

Der benyttes Retrofit til at konsumere Web-API’et. For at hente disse resourcer tilføjes de to linier vist med fed.

dependencies **{**

**...**  **implementation 'com.squareup.retrofit2:retrofit:2.3.0'  
 implementation 'com.squareup.retrofit2:converter-gson:2.3.0'**  
**}**

**Frugt.java**

Klassen Frugt, som Retrofit skal konvertere til og fra JSON, må kun have fields med navne, hvor første bogstav er lowercase. Selv om de er private og der benyttes set- og get-metoder, kan Retrofit ikke håndtere dem hvis de begynder med stort.  
I øvrigt skal klassen Frugt være fuldstændig identisk med klassen Frugt i API-programmet.

**IFrugtService.java**

Dette interface oprettes for at kunne definere hvilke Java-metoder der skal oprettes og deres tilknytning til URL-endelser for kald af Web-Api.

public interface IFrugtService  
{  
 @GET("Frugt")  
 Call<List<Frugt>> getAllFrugt();

@GET("Frugt/{id}")  
 Call<Frugt> getFrugtById(@Path("id") int id);

@POST("Frugt")  
 Call<Void> addFrugt(@Body Frugt frugt);

...  
}

**ServiceBuilder.java**

Denne klasse oprettes og ved kald af dens metode buildService() med IfrugtService som parameter, returneres et objekt med metoderne i IfrugtService implementeret.

URL’en i klassen vil blive tilføjet den tekst der står i IfrugtService metodernes annotations, og bliver den endelige URL i kaldet af Web-API.

public class ServiceBuilder  
{  
 private static final String *URL* = "http://10.0.2.2:8080/JansApiTest/api/";  
 private static Retrofit *retrofit* =  
 new Retrofit.Builder().baseUrl(*URL*)  
 .addConverterFactory(GsonConverterFactory.*create*()).build();  
  
 public static <F> F buildService(Class<F> serviceType)  
 {  
 return *retrofit*.create(serviceType);  
 }  
}

Brug af de oprettede resourcer til at hente en Frugt på dens id

protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*activity\_main*);  
  
 txtFrugtName = findViewById(R.id.*txtFrugtName*);  
  
**IFrugtService frugtService = ServiceBuilder.*buildService*(IFrugtService.class);  
 Call<Frugt> request = frugtService.getFrugtById(1001);**

**request.enqueue(new Callback<Frugt>() {  
 @Override  
 public void onResponse(Call<Frugt> call, Response<Frugt> response)  
 {  
 Frugt frugt = response.body();  
 txtFrugtName.setText(frugt.getName());  
 }  
 @Override  
 public void onFailure(Call<Frugt> call, Throwable t)  
 {  
 txtFrugtName.setText(t.getMessage());  
 }  
 });**}

En bedre struktur i brugen af Retrofit

Det kunne være rart, hvis man ikke behøver at oprette disse call-back-metoder hver kan Web-API’et skal kontaktes, men bare kunne kalde en metode som vist her under.

protected void onCreate(Bundle savedInstanceState)

{  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*activity\_main*);  
  
 txtFrugtName = findViewById(R.id.*txtFrugtName*);  
  
 **Frugt frugt = ApiLayer.*getFrugtById*(1002);  
 txtFrugtName.setText(frugt.getName());**  
}

**ApiLayer.java**

I denne klasse alle kald til Web-API’et samles, sådan at der hvor de skal bruges kun skal kaldes en metode. Her er getFrugtById() lavet så den indeholder hele klumpen til at kalde Retrofits getFrugtById().

De sædvanlige to linier for at komme i gang med at kontakte Web-API’et er vist med fed.

Det objekt som returneres af getFrugtById() har en metode execute() i stedet for at oprette call-back-metoderne. Den kan returnere data. Der er bare det problem at execute() ikke må kaldes i hovedtråden.

Derfor lægges det ind i en Callable/call(), som er et alternativ til Runnable/run() med den forskel at call() kan returnere data. Dette lægges igen ind i en FutureTask som efterfølgende udføres i en ny tråd.

Når tråden er startet, kan FutureTask’s get() kaldes, og her tåler hovedtråden at der ventes indtil get() returnerer data.

public class ApiLayer  
{  
 public static Frugt getFrugtById(int id)

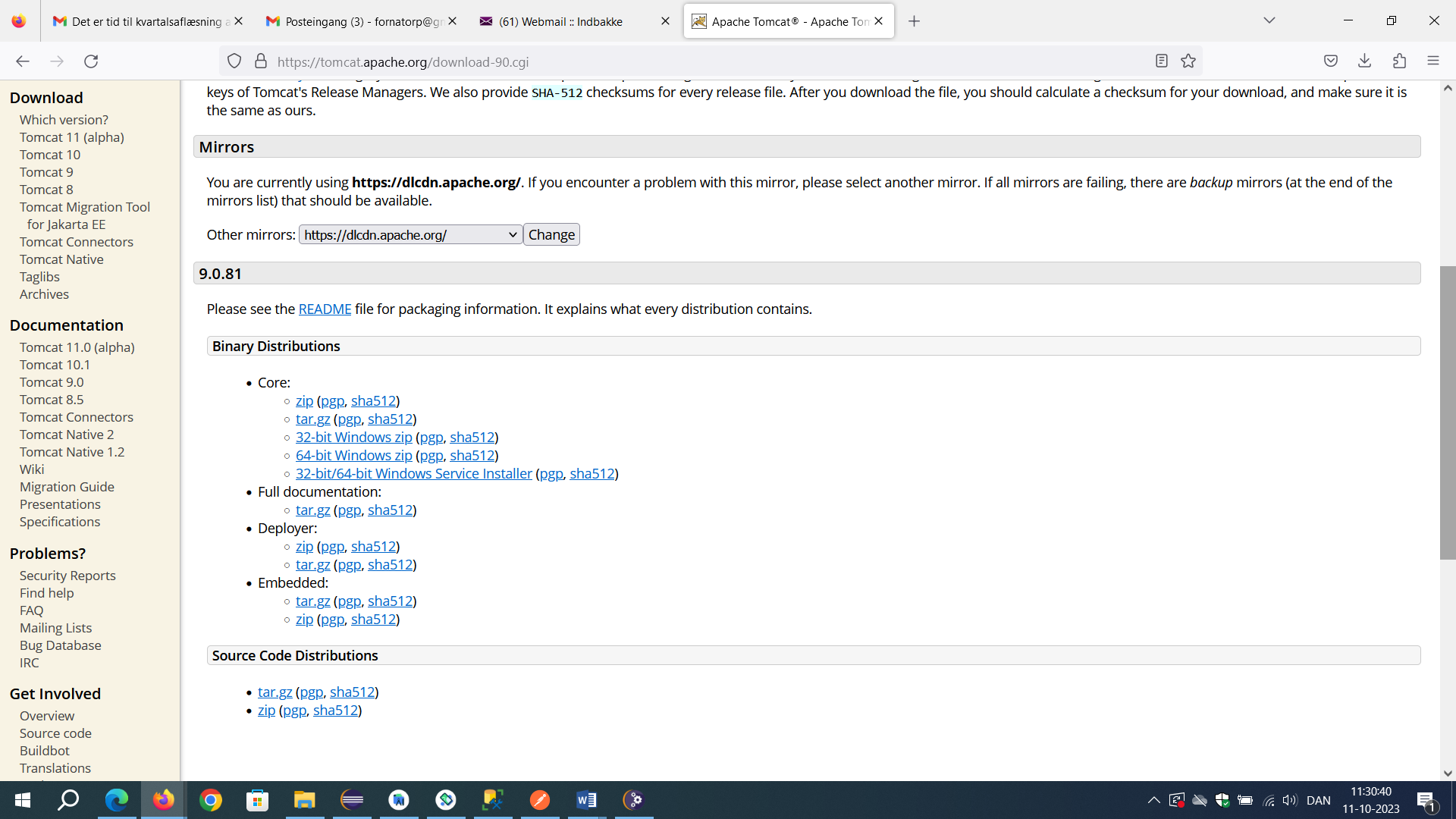
{  
 FutureTask<Frugt> futureTask = new FutureTask<>(new Callable<Frugt>()   
 {  
 @Override  
 public Frugt call() {  
 Frugt f = null;  
 **IFrugtService service = ServiceBuilder.*buildService*(IFrugtService.class);  
 Call<Frugt> request = service.getFrugtById(id);** try {  
 f = request.execute().body(); *// Kaldet sker i ny tråd* } catch (IOException e) {  
 e.printStackTrace();  
 }  
 return f;  
 }  
 });  
 Thread t = new Thread(futureTask);  
 t.start();  
 Frugt frugt = null;  
 try {  
 frugt = futureTask.get();*// will wait for the async completion* } catch (Exception e) {  
 Log.*d*("Thread", e.getMessage());  
 }  
 return frugt;  
 }  
}

## Detaljer om API’et

### Apache Tomcat

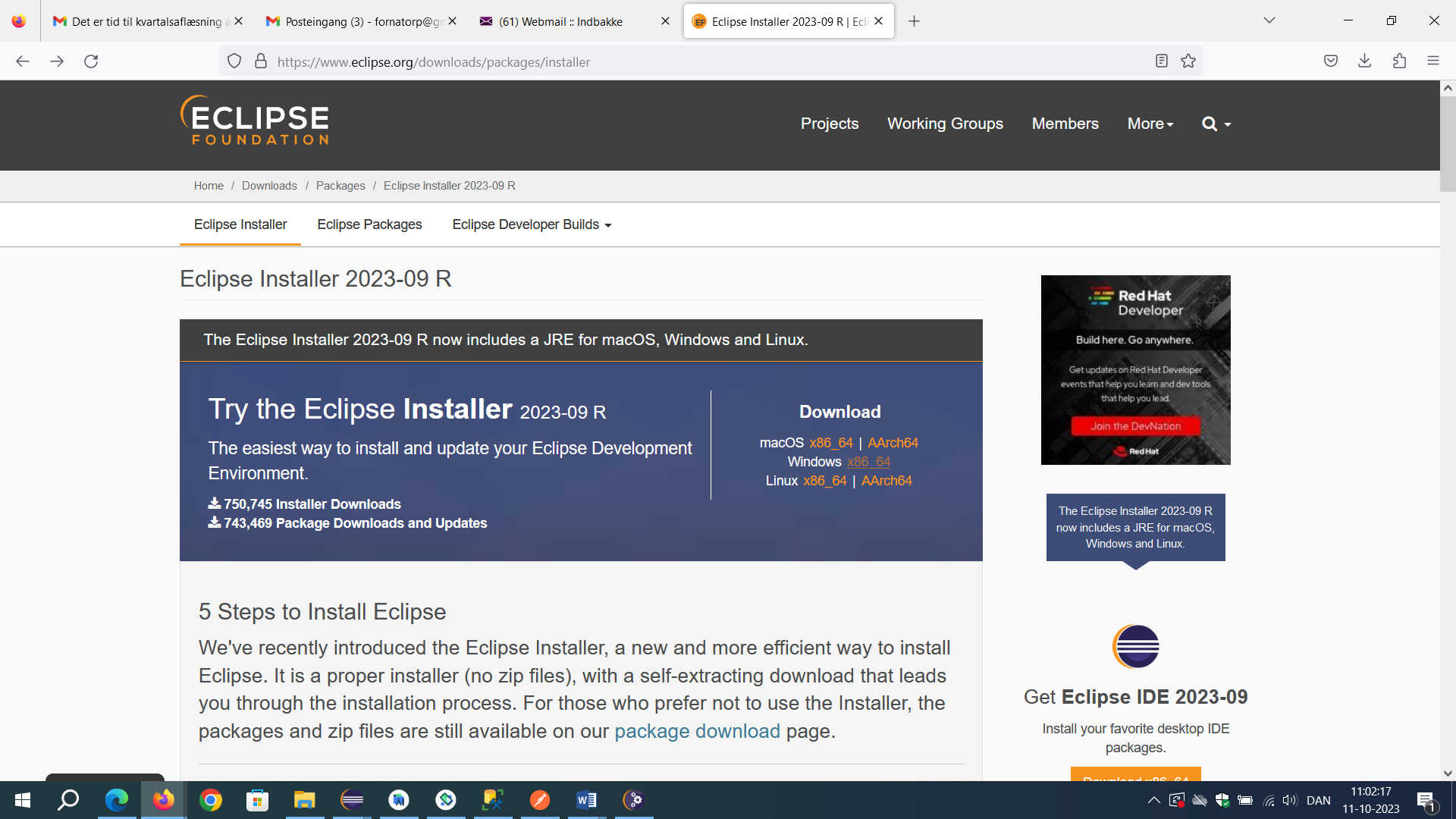
Installer Apache Tomcat 9.

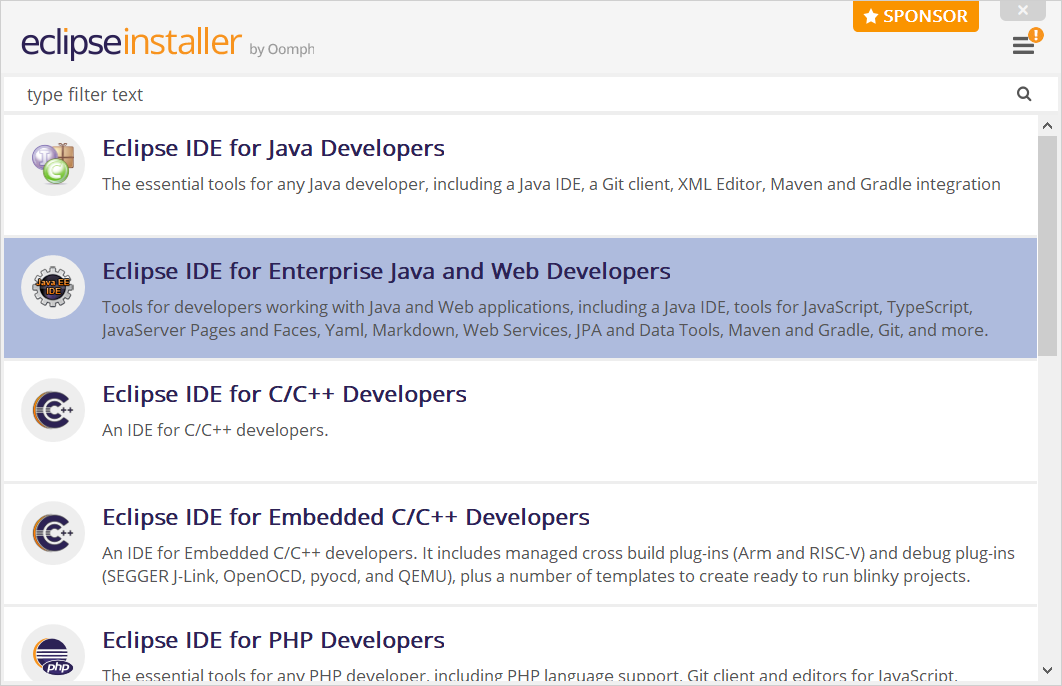
Vælg Windows Service Installer



### Eclipse

Installer Eclipse ved at hente EclipseInstaller, køre den og vælge Eclipse IDE for Enterprise Java.





Opret et nyt Dynamic Web Project

Hvis muligheden ikke er der så gå til Help – Install new Software, og vælg i drop down Work with: All Available Sites.

Når der er fundet alt så klik den nederste til: Web XML JEEE……..

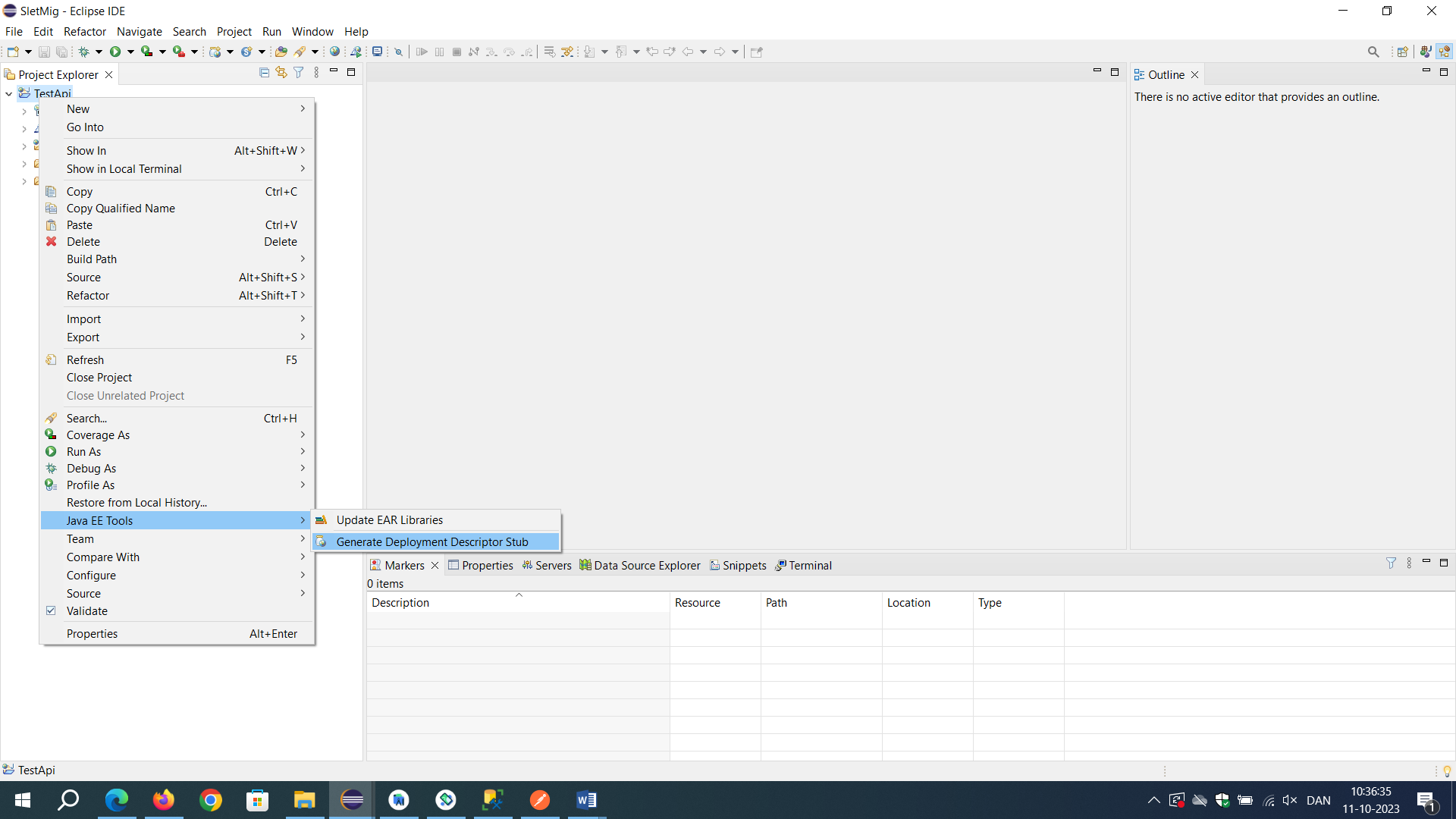
Hvis det ikke kan lade sig gøre, så prøv med at adde følgende site og udfør kun den.

<http://download.eclipse.org/webtools/updates>

**web.xml**

web.xml kaldes også Deployment Descriptor.

Den oprettes ved at højreklikke på projektnavnet og vælge Java EE Tools – Generate Deployment Descriptor Stub.



Og så udskifte indholdet til følgende

<?xml version=*"1.0"* encoding=*"ISO-8859-1"*?>

<web-app xmlns=*"http://java.sun.com/xml/ns/javaee"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee*

*http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd"*

version=*"3.0"*>

<display-name>Hello World</display-name>

<servlet>

<servlet-name>MyApiServlet</servlet-name>

<servlet-class>dk.tec.jaj.example2.ApiServlet</servlet-class>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>MyApiServlet</servlet-name>

<url-pattern>/api/\*</url-pattern>

</servlet-mapping>

</web-app>

**MatchEnum.java**

**public** **enum** MatchEnum

{

***MatchFrugtId***, ***MatchFrugt***, ***MatchNo***;

}

**AnalyzeRequest.java**

**public** **class** AnalyzeRequest

{

MatchEnum match;

**int** id;

**public** MatchEnum getMatch() {

**return** match;

}

**public** **int** getId() {

**return** id;

}

**public** AnalyzeRequest(String pathInfo)

{

Matcher matcher = Pattern.*compile*("/Frugt/([0-9]+)").matcher(pathInfo);

**if**(matcher.find())

{

match = MatchEnum.***MatchFrugtId***;

id = Integer.*parseInt*(matcher.group(1));

}

**else**

{

matcher = Pattern.*compile*("/Frugt").matcher(pathInfo);

**if**(matcher.find())

{

match = MatchEnum.***MatchFrugt***;

}

**else**

{

match = MatchEnum.***MatchNo***;

}

}

}

}

**Frugt.java**

**public** **class** Frugt

{

**int** id;

String name;

String descr;

**int** weight;

String country;

**public** Frugt() {}

**public** Frugt(**int** id, String name, String descr, **int** weight, String country) {

**super**();

**this**.name = name;

**this**.descr = descr;

**this**.weight = weight;

**this**.country = country;

}

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** String getDescr() {

**return** descr;

}

**public** **void** setDescr(String descr) {

**this**.descr = descr;

}

**public** **int** getWeight() {

**return** weight;

}

**public** **void** setWeight(**int** weight) {

**this**.weight = weight;

}

**public** String getCountry() {

**return** country;

}

**public** **void** setCountry(String country) {

**this**.country = country;

}

}

**ApiServlet.java**

**public** **class** ApiServlet **extends** HttpServlet

{

**protected** **void** doGet(HttpServletRequest request, HttpServletResponse response)   
 **throws** ServletException, IOException   
{

PrintWriter out = response.getWriter();

AnalyzeRequest analyze = **new** AnalyzeRequest(request.getPathInfo());

ObjectMapper mapper = **new** ObjectMapper();

DBTools db = **new** DBTools();

**switch**(analyze.getMatch())

{

**case** ***MatchFrugtId***:

Frugt f = db.getFrugtById(analyze.getId());

out.write(mapper.writeValueAsString(f));

**break**;

**case** ***MatchFrugt***:

**break**;

**case** ***MatchNo***:

out.write("No Match");

**break**;

}

}

@Override

**protected** **void** doPost(HttpServletRequest request, HttpServletResponse response)   
 **throws** ServletException, IOException

{

String receivedJSON = request.getReader().readLine();

ObjectMapper mapper = **new** ObjectMapper();

Frugt f = mapper.readValue(receivedJSON, Frugt.**class**);

System.***out***.println(f.getName());

}

}

**DBTools.java**

**public** **class** DBTools

{

**private** String conStr =

"jdbc:sqlserver://localhost;databaseName=FrugtDB;encrypt=true;trustServerCertificate=true";

Connection con;

Statement stmt;

**public** DBTools()

{

**try** {

Class.*forName*("com.microsoft.sqlserver.jdbc.SQLServerDriver");

} **catch** (ClassNotFoundException e) {

e.printStackTrace();

}

}

**private** **void** connect()

{

**try** {

con = DriverManager.*getConnection*(conStr, "sa", "1234");

stmt = con.createStatement();

} **catch** (SQLException e) {

e.printStackTrace();

}

}

**public** Frugt getFrugtById(**int** id)

{

connect();

String selectStr = "Select \* from Frugt where id = " + id;

Frugt frugt = **new** Frugt();

**try** {

ResultSet result = stmt.executeQuery(selectStr);

**if**(result.next())

{

frugt.setId(result.getInt("Id"));

frugt.setName(result.getString("Name"));

frugt.setDescr(result.getString("Descr"));

frugt.setWeight(result.getInt("Weight"));

frugt.setCountry(result.getString("Country"));

con.close();

}

} **catch** (SQLException e) {

e.printStackTrace();

}

**return** frugt;

}

}