SWEN90007 - Software Design and Architecture

Semester 2 – 2020 – Additional Learning Resource for Students

Deploying a Java Web App to Heroku

This guide illustrates how to deploy a Tomcat-based Java Web Application hosted in GitHub to Heroku. We will use a Maven-enabled version of the simple LMS application created in workshop 5.

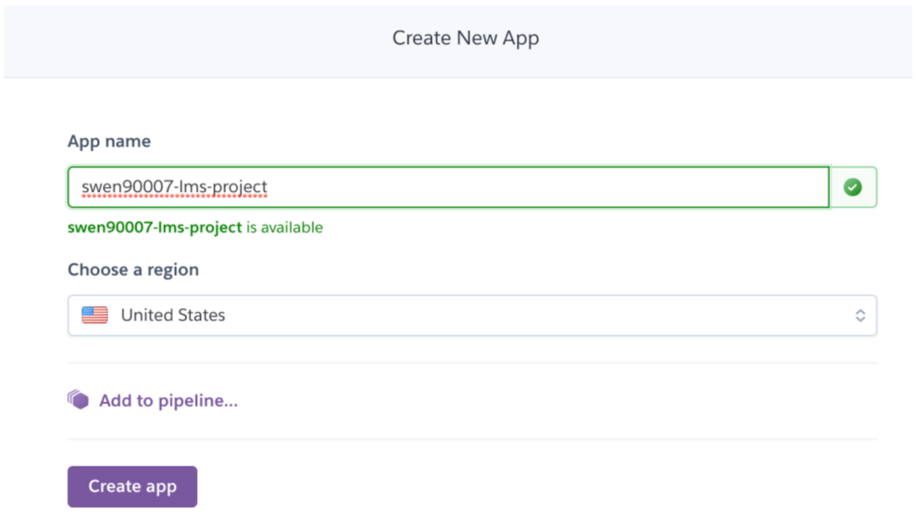
You can access the GitHub repo here: https://github.com/mariaars/swen90007-lms-project

To replicate the steps in this tutorial, you can clone the LMS project and push it into a repository in your GitHub account.

Creating a Heroku App

Create a free Heroku account (https://signup.heroku.com/) and access the Heroku dashboard here: https://dashboard.heroku.com/apps.

Create a new Heroku App. Enter your app's name. For this example, we will use the default deployment region (United States).



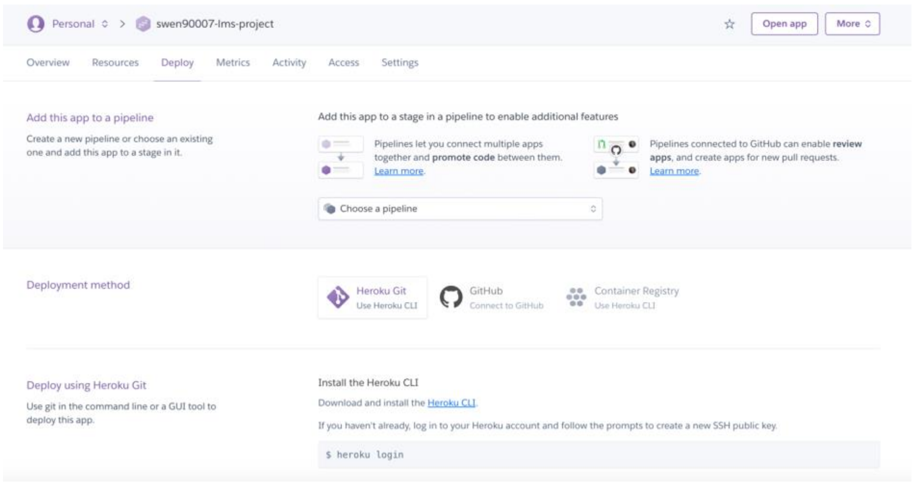
Create New App

App name: swen90007-lms-project

swen90007-lms-project is available

Choose a region: United States

After creating your Heroku app, you should see a screen like this:


Personal > swen90007-lms-project

Open app | More

Overview | Resources | Deploy | Metrics | Activity | Access | Settings

Add this app to a pipeline

Create a new pipeline or choose an existing one and add this app to a stage in it.

Add this app to a stage in a pipeline to enable additional features

Pipelines let you connect multiple apps together and **promote code** between them. Learn more.

Pipelines connected to GitHub can enable **review apps**, and create apps for new pull requests. Learn more.

Choose a pipeline

Deployment method

Heroku Git — Use Heroku CLI | GitHub — Connect to GitHub | Container Registry — Use Heroku CLI

Deploy using Heroku Git

Use git in the command line or a GUI tool to deploy this app.

Install the Heroku CLI

Download and install the Heroku CLI.

If you haven't already, log in to your Heroku account and follow the prompts to create a new SSH public key.

```
$ heroku login
```

At this point your App is just a shell, we now need to deploy something to it.

Preparing the LMS Project

Before deploying the LMS project to the Heroku App, we need to make a few changes to the code.

Database Connection URL and Credentials

Our application won’t be using a local Postgres DB anymore, instead, it will access a Heroku-managed Postgres DB. This DB will be automatically created for you by Heroku when you first deploy the app (we will learn how to access it and set it up later in this guide).

In this step, we are going to change the project’s code to connect to Heroku’s Postgres DB. We will use the environment variable JDBC_DATABASE_URL created by Heroku to access the connection’s URL (it includes the DB’s username and password).

The getDBConnection() method in DBConnectin.java now looks like this:

```
private static Connection getDBConnection() {
    try {
        DriverManager.registerDriver(new org.postgresql.Driver());
        String dbUrl = System.getenv("JDBC_DATABASE_URL");
        Connection dbConnection = DriverManager.getConnection(dbUrl);
        return dbConnection;
    } catch (SQLException e) {
        System.out.println(e.getMessage());
    }
    System.out.println("Connection problem");
    return null;

}
```

Commented [EAO1]: they will have access to this updated version on GitHub, correct? otherwise we could keep this as a text.

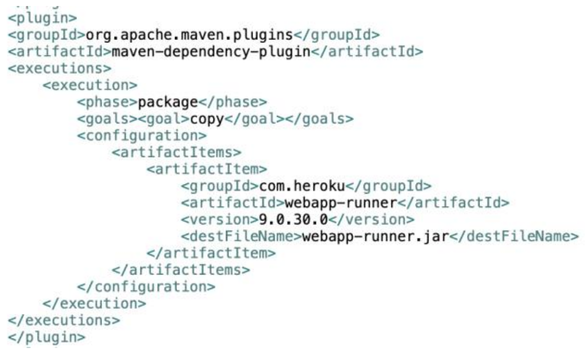
Commented [MRR2R1]: Yes they do have access to it ☺

You can learn more about connecting to relational databases on Heroku here:
[https://devcenter.heroku.com/articles/heroku-postgresql - connecting-in-java](https://devcenter.heroku.com/articles/heroku-postgresql#connecting-in-java)

https://devcenter.heroku.com/articles/connecting-to-relational-databases-on-heroku-with-java

Configuring Maven

In this example, we will be using Webapp Runner to launch our application in a Tomcat container. To do this, we will configure Maven to download Webapp runner for us. We do this by including the following plugin in the project's pom.xml file:



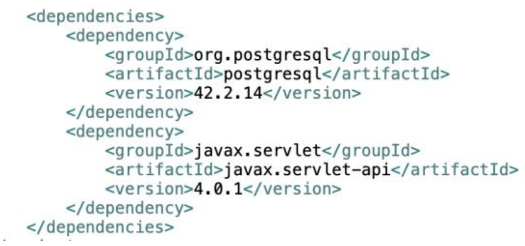
```
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-dependency-plugin</artifactId>
<executions>
    <execution>
        <phase>package</phase>
        <goals><goal>copy</goal></goals>
        <configuration>
            <artifactItems>
                <artifactItem>
                    <groupId>com.heroku</groupId>
                    <artifactId>webapp-runner</artifactId>
                    <version>9.0.30.0</version>
                    <destFileName>webapp-runner.jar</destFileName>
                </artifactItem>
            </artifactItems>
        </configuration>
    </execution>
</executions>
</plugin>
```

Commented [EK3]: if they are using mvn, they should add all other dependencies :

```
 <dependencies>
  <dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>3.0.1</version>
  </dependency>
  <!--
https://mvnrepository.com/artifact/org.postgresql/postgresql -->
<dependency>
  <groupId>org.postgresql</groupId>
  <artifactId>postgresql</artifactId>
  <version>42.2.5</version>
</dependency>

 </dependencies>
```

Commented [MRR4R3]:

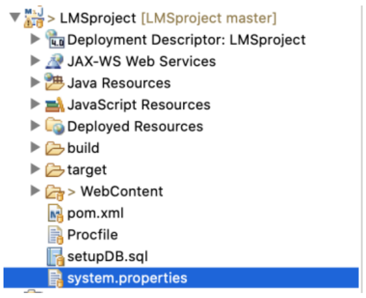
Note: Ensure your pom.xml file has all the dependencies needed for the project. Check the source code for a full example of this file.



```
<dependencies>
    <dependency>
        <groupId>org.postgresql</groupId>
        <artifactId>postgresql</artifactId>
        <version>42.2.14</version>
    </dependency>
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>javax.servlet-api</artifactId>
        <version>4.0.1</version>
    </dependency>
</dependencies>
```

Configuring the Java Runtime Version

In this step we will specify the Java version we want Heroku to use when deploying our app. To do this, we need to add a *system.properties* file to the root directory of the project.



```
> LMSproject [LMSproject master]
    Deployment Descriptor: LMSproject
    JAX-WS Web Services
    Java Resources
    JavaScript Resources
    Deployed Resources
    build
    target
    > WebContent
    pom.xml
    Procfile
    setupDB.sql
    system.properties
```

For this project, we will be using Java 11 (for no particular reason). The *system.properties* file contains the following line:

```
java.runtime.version=11
```

And the pom.xml specifies release 11:

```
<plugin>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>3.8.0</version>
    <configuration>
        <release>11</release>
    </configuration>
</plugin>
```

Creating a Procfile

A Procfile (case-sensitive, no extension) is used to specify how an application will be executed in Heroku. Create a Procfile in the project's root directory with the following line in it:

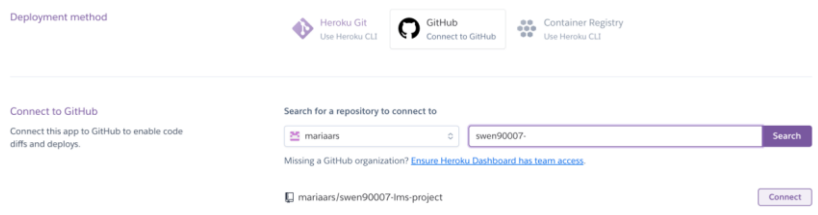
```
web: java $JAVA_OPTS -jar target/dependency/webapp-runner.jar --port $PORT target/*.war
```

The LMS project is now ready to be deployed in Heroku!

Linking Heroku and GitHub

In this step we will link the project's GitHub Repo with the Heroku app.

In your app's dashboard, go to the *Deployment method* section in the *Deploy* tab and select GitHub. Follow the prompts to connect to your GitHub account. Once this is done you can search for the repo you want to use for deployment and connect to it:



Connect to GitHub

Connect this app to GitHub to enable code diffs and deploys.

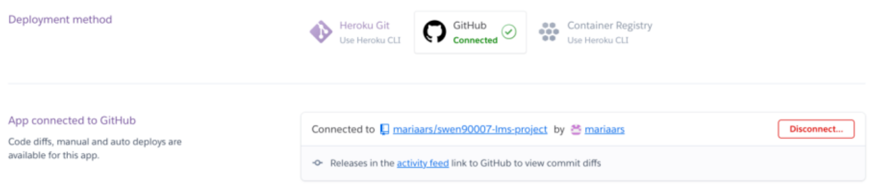
Search for a repository to connect to

mariaars | swen90007- | Search

Missing a GitHub organization? Ensure Heroku Dashboard has team access.

mariaars/swen90007-lms-project | Connect

Once you have successfully connected the app to GitHub you should see it in the dashboard:


Deployment method

Heroku Git
Use Heroku CLI

GitHub
Connected

Container Registry
Use Heroku CLI

App connected to GitHub
Code diffs, manual and auto deploys are available for this app.

Connected to mariaars/swen90007-lms-project by mariaars

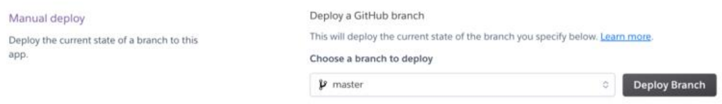
Disconnect...

Releases in the activity feed link to GitHub to view commit diffs

You now have the option of automatically or manually deploying the code in your GitHub repo to your Heroku app. The next section guides you through the manual deployment option.

Deploying the LMS Project

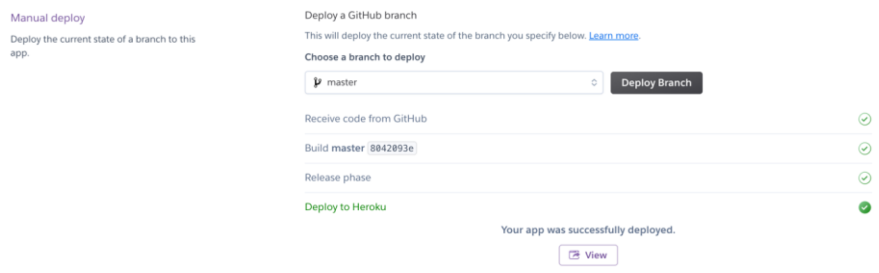
Once you have successfully connected your app to GitHub, you will have the option to manually deploy the app (scroll down to the bottom of the page to see it):


Manual deploy
Deploy the current state of a branch to this app.

Deploy a GitHub branch
This will deploy the current state of the branch you specify below. Learn more.
Choose a branch to deploy
master
Deploy Branch

Simply select the branch you want to deploy and click Deploy Branch! Heroku will do all the hard work for you.

If the process was successful you will see something like this:


Manual deploy
Deploy the current state of a branch to this app.

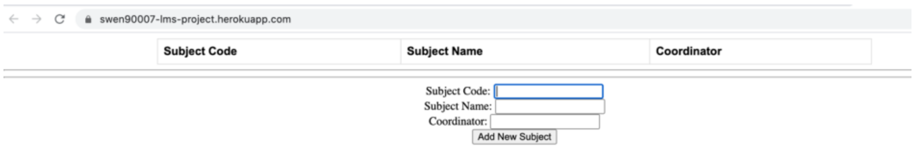
Deploy a GitHub branch
This will deploy the current state of the branch you specify below. Learn more.
Choose a branch to deploy
master
Deploy Branch

Receive code from GitHub ✓
Build master 8042093e ✓
Release phase ✓
Deploy to Heroku ✓

Your app was successfully deployed.
View

Click on View to access your app! You can also access your app using the *Open app* button on the top right corner of the dashboard.

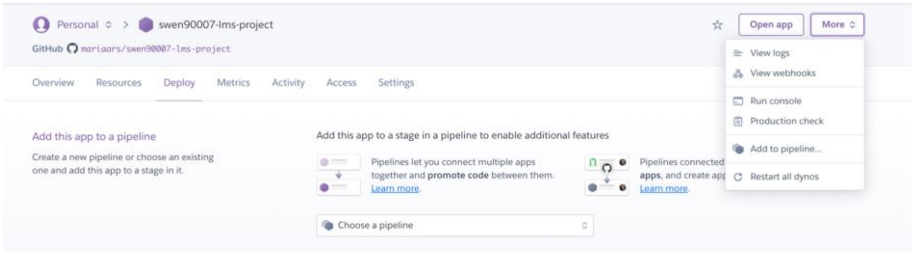
You should see the homepage of the LMS project:


swen90007-lms-project.herokuapp.com

Subject Code	Subject Name	Coordinator

Subject Code:
Subject Name:
Coordinator:
Add New Subject

If you had any issues, you may want to check your application logs. This can be done using the Heroku-cli (not covered in this guide) or by clicking in the *More* button on the top right corner of the dashboard.


Personal > swen90007-lms-project
GitHub mariaars/swen90007-lms-project

Add this app to a pipeline
Create a new pipeline or choose an existing one and add this app to a stage in it.

Add this app to a stage in a pipeline to enable additional features
Pipelines let you connect multiple apps together and **promote code** between them. Learn more.
Pipelines connected **apps**, and create app Learn more.
Choose a pipeline

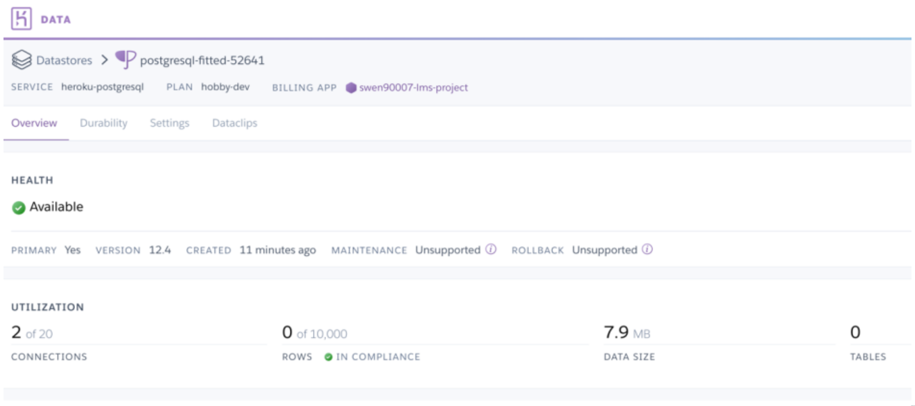
It's now time to create the *subjects* table in the DB so that we can use our newly deployed app.

Setting up the Database

Go to the *Resources* tab in your app's Dashboard. Click on the Heroku Postgres add-on to access the DB details.

Note: If a Postgres add-on was not created automatically after deploying your app, you will have to create one manually (https://devcenter.heroku.com/articles/heroku-postgresql#provisioning-heroku-postgres).

Commented [EK5]: they need to install Heroku Postgres add-on, to be able to access it


DATA
Datastores > postgresql-fitted-52641
SERVICE heroku-postgresql PLAN hobby-dev BILLING APP swen90007-lms-project

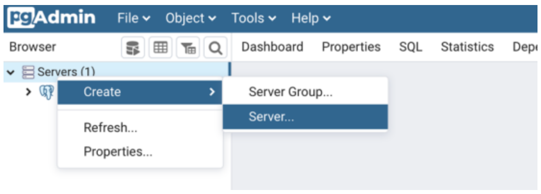
HEALTH
Available

PRIMARY Yes VERSION 12.4 CREATED 11 minutes ago MAINTENANCE Unsupported ROLLBACK Unsupported

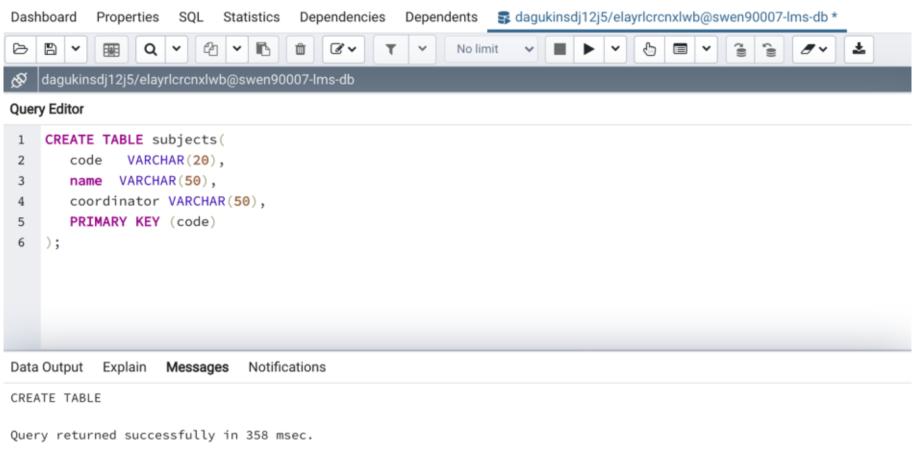
UTILIZATION
2 of 20 CONNECTIONS
0 of 10,000 ROWS IN COMPLIANCE
7.9 MB DATA SIZE
0 TABLES

Go to the *Settings* tab and click on *View credentials*. You can use this information to connect and set up your DB as needed. In this example, we are going to use PGAdmin 4 for this purpose.

Create a new server and use your DB credentials to connect to it.



Create the subjects table:



```
CREATE TABLE subjects(
   code    VARCHAR(20),
   name   VARCHAR(50),
   coordinator VARCHAR(50),
   PRIMARY KEY (code)
);
```

```
CREATE TABLE

Query returned successfully in 358 msec.
```

Keep in mind that the DB URL (incl. credentials) is managed by Heroku and can change under some circumstances. This is one of the reasons for using the JDBC_DATABASE_URL env var in the code, instead of the hardcoded URL. You can read more about this here: https://devcenter.heroku.com/articles/connecting-to-heroku-postgres-databases-from-outside-of-heroku

That's the deployment done. Try adding a few subjects now ☺

App deployment via WAR File

You can also deploy your application by creating a WAR file and submitting it to Heroku. We will outline the main steps in this section. For more details on this approach refer to: https://devcenter.heroku.com/articles/war-deployment#deployment-with-the-heroku-cli.

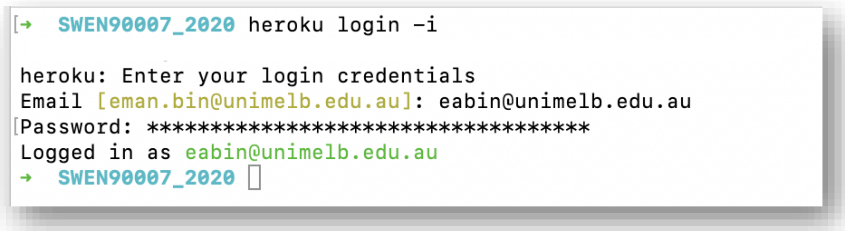
For this example, you can use the Maven-less code given to you as part of Workshop 5.

Create a Heroku app following the steps in the **Creating a Heroku App section**.

Install the Heroku CLI by following these instructions: https://devcenter.heroku.com/articles/heroku-cli.

Open a terminal and login to Heroku using the command:

```
heroku login -i
```



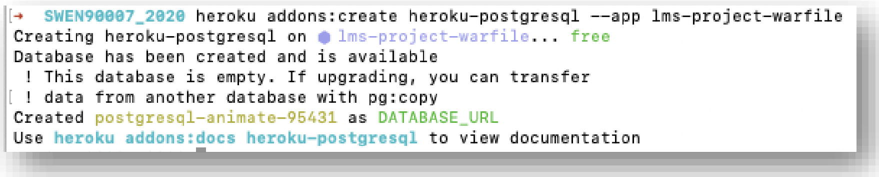
```
[➜  SWEN90007_2020 heroku login -i

heroku: Enter your login credentials
Email [eman.bin@unimelb.edu.au]: eabin@unimelb.edu.au
[Password: ***********************************
Logged in as eabin@unimelb.edu.au
➜  SWEN90007_2020 
```

Install the CLI WAR deployment plugin:

```
heroku plugins:install heroku-cli-deploy
```

Create the Postgres database that will be used by your application:

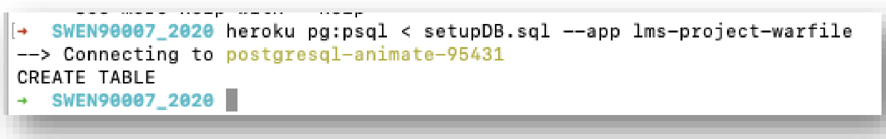
```
heroku addons:create heroku-postgresql --app <App_name>
```



```
[➜  SWEN90007_2020 heroku addons:create heroku-postgresql --app lms-project-warfile
Creating heroku-postgresql on ⬢ lms-project-warfile... free
Database has been created and is available
 ! This database is empty. If upgrading, you can transfer
[ ! data from another database with pg:copy
Created postgresql-animate-95431 as DATABASE_URL
Use heroku addons:docs heroku-postgresql to view documentation
```

Setup the database using the setupDB.sql file in the LMS sample project:

```
heroku pg:psql < setupDB.sql --app <App_name>
```



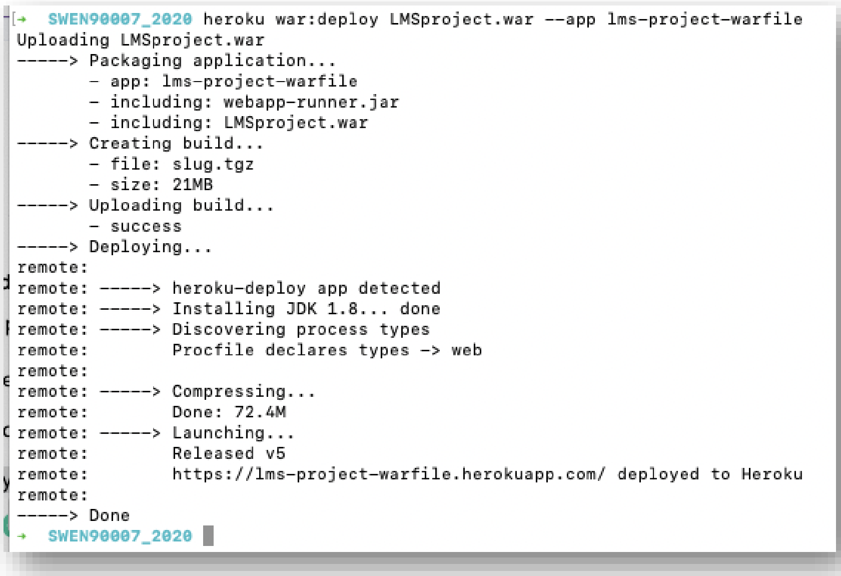
```
[→  SWEN90007_2020 heroku pg:psql < setupDB.sql --app lms-project-warfile
--> Connecting to postgresql-animate-95431
CREATE TABLE
→  SWEN90007_2020
```

Update the code to connect to the newly created DB by retrieving the connection URL from the JDBC_DATABASE_URL environment variable (see the **Database Connection URL and Credentials** section).

Build a WAR file of the LMS sample app. In Eclipse, you can do this by right-clicking on the project -> File -> Export -> Web -> WAR file.

Now that we have the app packaged as a WAR file, a DB, and Heroku's app configured, we are ready to deploy:

```
heroku war:deploy <path_to_war_file> --app <app_name>
```



```
[→  SWEN90007_2020 heroku war:deploy LMSproject.war --app lms-project-warfile
Uploading LMSproject.war
-----> Packaging application...
       - app: lms-project-warfile
       - including: webapp-runner.jar
       - including: LMSproject.war
-----> Creating build...
       - file: slug.tgz
       - size: 21MB
-----> Uploading build...
       - success
-----> Deploying...
remote:
remote: -----> heroku-deploy app detected
remote: -----> Installing JDK 1.8... done
remote: -----> Discovering process types
remote:        Procfile declares types -> web
remote:
remote: -----> Compressing...
remote:        Done: 72.4M
remote: -----> Launching...
remote:        Released v5
remote:        https://lms-project-warfile.herokuapp.com/ deployed to Heroku
remote:
-----> Done
→  SWEN90007_2020
```

Access your app (e.g. https://lms-project-warfile.herokuapp.com/).

More resources are available in Heroku's dev center: https://devcenter.heroku.com/categories/java-support.