Java Web Jdbc Hacking Exercises

The Hacking Exercises

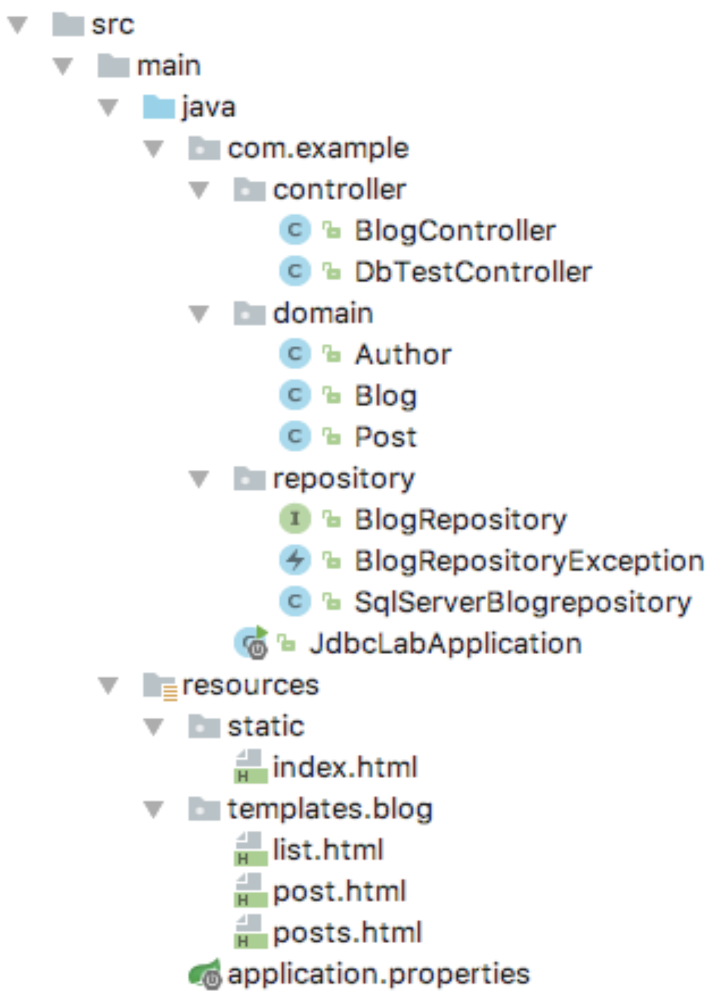
In this exercise, we will work on a Starter Project that is a Blogging platform where celebrity bloggers put their blogs.

We will learn from it, extend it with new functionality, and we can also use it as a starter project to test whatever we feel like trying out.

It is an example of a project that use Java, Web technology and JDBC to access a database, so it's the perfect example of what we will develop in the coming mini projects.

Exercise 1 – Analyze the project

The project structure looks like this:



```
src
  main
    java
      com.example
        controller
          BlogController
          DbTestController
        domain
          Author
          Blog
          Post
        repository
          BlogRepository
          BlogRepositoryException
          SqlServerBlogrepository
        JdbcLabApplication
    resources
      static
        index.html
      templates.blog
        list.html
        post.html
        posts.html
      application.properties
```

Look at the controllers in the **java/com/example/controller** package.

The **DBTestController** is just for checking the database, and the **BlogController** is for handling the blog. We will add code to the **BlogController**. Look at the controllers and see if you understand what they do.

Look at the Domain objects in the **java/com/example/domain** package.

This is the domain objects that hold data from the database, and this is what is returned from the repository when looking things up in the database.

Look at the repository classes in the **java/com/example/repository** package.

The **BlogRepository** is an Interface that is implemented by the **JdbcBlogRepository** where the implementation code is. Look at the code in the **JdbcBlogRepository**. Do you understand how the repository communicates with the database?

Notice that we get an autowired **Datasource** in the **JdbcBlogRepository** that Spring automatically creates for us from the connection string we specify in the configuration file **application.properties**.

The **JdbcLabApplication** is just an application class that starts up Spring Boot, and initiates the H2 memory database.

The database used in this project is an embedded H2 memory database, so a running database is included in the project and is running in parallel with the project Java code. The project is therefore self-contained with its own database. We don't need to set up another database, but we can easily use another database instead if we want to.

Look at the resources. We have a static **index.html** file under static, and some Thymeleaf templates under **resources/templates/blog**. These are the views that are called from the controller methods of the **BlogController** class.

Look at the **application.properties** file. This file is a configuration file. Here we specify some configuration, and especially the jdbc connection string. The H2 connection string can be replaced with a SQL Server connection string if we want to use SQL Server instead.

Read about the H2 database here:

http://www.h2database.com/html/main.html

You can use this starter project to do the exercises in this document, and create your own functionality, or create another project and use this project as an example of how the functionality can be developed. Create your own database for the project or use one that already exists.

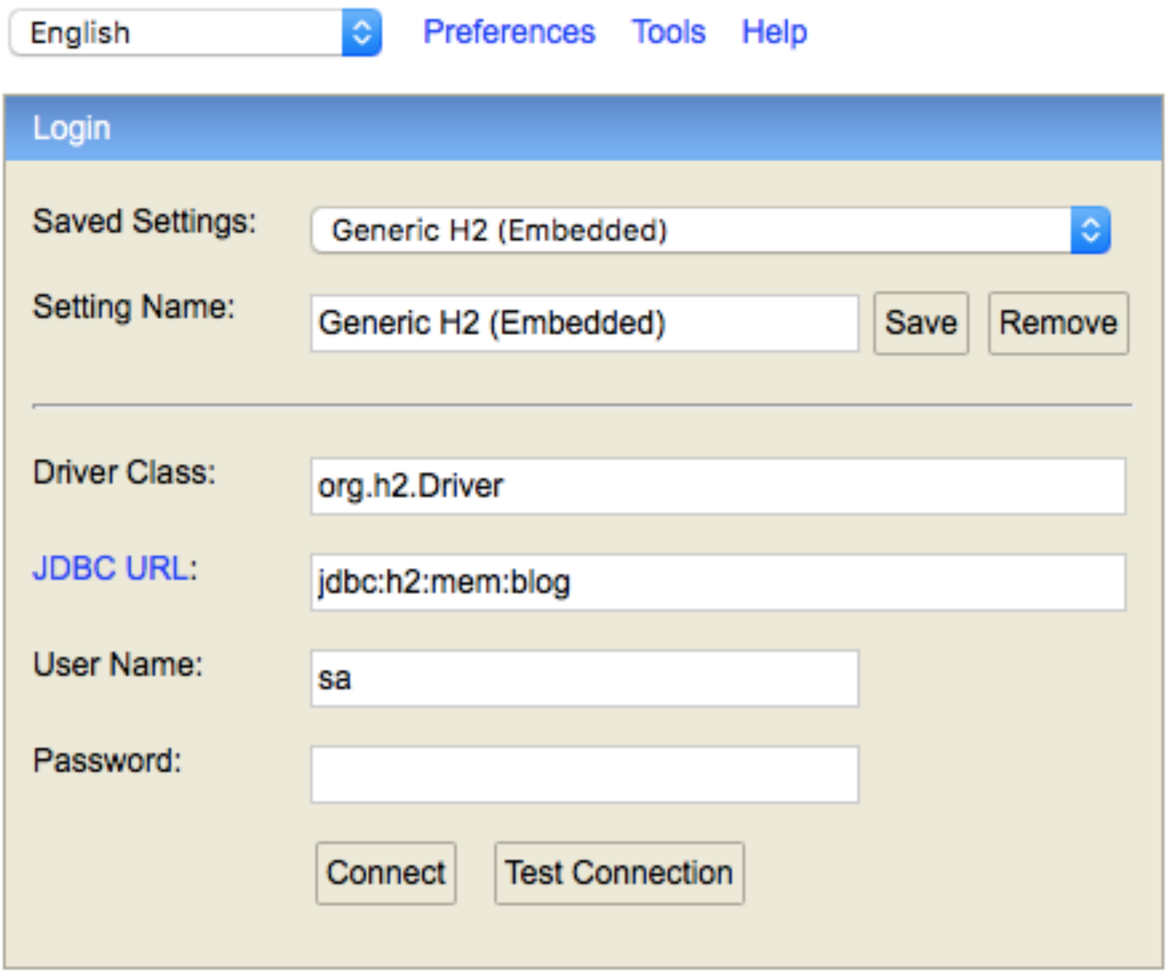
Do what you need to do to prepare for the mini projects, and work with your project group on these exercises if that's how you like to do it!

Exercise 2 – Run the project

Run the project, and you can use the H2 Database Console to look at the database with this URL:

http://localhost:8080/h2

Login to the database like this:



Saved Settings: Generic H2 (Embedded)

Setting Name: Generic H2 (Embedded)

Driver Class: org.h2.Driver

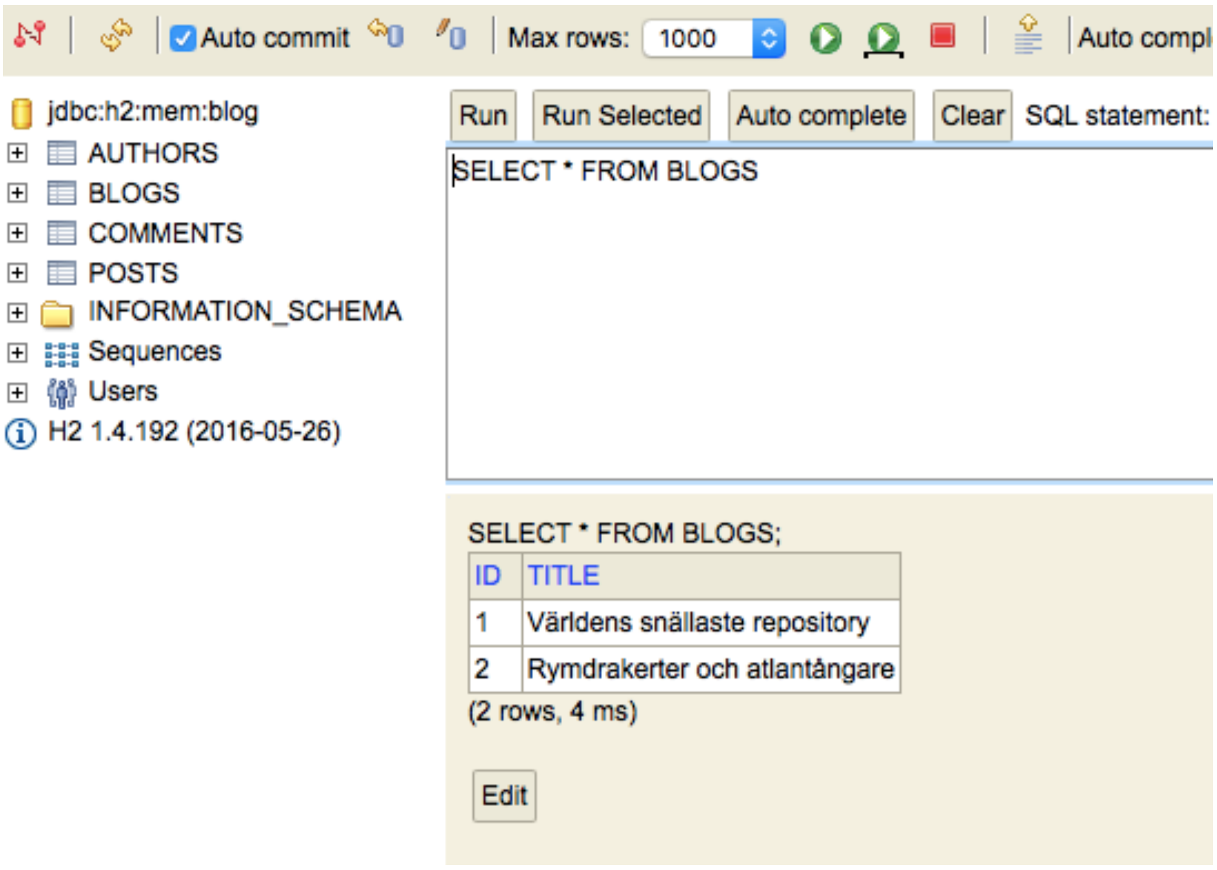
JDBC URL: jdbc:h2:mem:blog

User Name: sa

Password:

Click on the tables on the left to get select statements in the text area to the right, and click Run to run the query. Look at the data in the table below (you must clear the text area between every query)

Look at image below:



```
SELECT * FROM BLOGS
```

SELECT * FROM BLOGS;

ID	TITLE
1	Världens snällaste repository
2	Rymdrakerter och atlantångare

(2 rows, 4 ms)

Go the the URL http://localhost:8080 in a web browser.

Here you can Check the database connectivity, and list the blogs in the project.

Look at the links in the html.

The database check links to **/dbtest**, which is mapped to the **testDb** method in the **DbTestController**.

The List blogs link links to **/blog**, which is mapped to the **listBlogs** method in the **BlogController**. Look at the controllers and you will see what will happen when you click the links before you click.

The **listBlogs** method uses the **BlogRepository** to get all blogs from the blogs table in the database, adds them to the model under the name **blogs**, and specifies the **blog/lists** to be the view template that will generate the view of the blogs. The view creates a list with the blog titles as links with the link to **/blog/[blog_id]**.

These links are mapped to the **listPosts** method in **BlogController** with the GetMapping:

```
@GetMapping("/blog/{blogId}/")
```

The method will get the id of the blog and use it when it calls the **getEntriesInBlog** method in the repository to get all posts of the blog. Look at these methods and see how they work.

The title of the blog in the **blog/posts** view is a link looking like this:

./posts/3/

The **./** in the beginning builds on the current URL which look like this:

http://localhost:8080/blog/2/

So, when clicked it will result in a link that looks like this:

http://localhost:8080/blog/2/posts/3/

This link is mapped to the **listPost** method in **BlogController** with the GetMapping:

```
@GetMapping("/blog/{blogId}/posts/{postId}")
```

This method should get the blog post and all its comments from the database, and put this information in the model.

The view template **blog/post** should present this information in the view.

This will be the next exercise.

Exercise 3 – Add functionality to show the blog post and comments

Look at the other controller methods and add functionality to the **listPost** method to get the post information and all its comments from the database, and add this information to the model.

You should probably also create a new domain object for the comment, so that the comments can be returned from the repository method as these comment domain objects.

You will probably need to add a new method in the **JdbcBlogRepository** to get the comments from the database, and add this method signature in the **BlogRepository** Interface.

Update the **blog/post** template (the **blog/post.html** file) to show the post information and all the comments in a list.

Look at the other methods and view templates for hints of how to do this, and try it out!

Exercise 4 – Add the name of the Author in the view of the blog posts

There is a table with the name **Authors** that is not used so far.

Add a method in the **JdbcBlogRepository** that gets the **Author** for a specified **blog_id**, and add a corresponding method signature in the **BlogRepository** Interface. Look at how the blog posts are handled, the author table has a foreign key to the blog table, just as the posts table has.

You should probably also create a new domain object in the domain package to hold information about the **Author**, so that the method in the repository can return this object.

Use the new repository method in the **listPosts** and **listPost** methods in the **BlogController** to get the **Author** for the blog, and add the **Author** to the model.

Update the corresponding view templates to show the **Author** information.

Exercise 5 – Add the functionality for anyone to add comments to a blog post

Add a form to the **blog/post** view so that someone reading the blog post can add a comment to the blog post.

You probably need to add hidden parameters in the form for the blogId and the postId, and get these in the PostMapping method so that the method can know what blog and post the comment belongs to.

Add a hidden form input like this:

Create a method in the controller with a **PostMapping** that handle the form submission. The form should let the user add a text for the comment and a name. The controller method should add a date and call a method in the repository that you also have to create.

Create the method in the repository that inserts the comment to the comments table.

Try to add some comments to the blog posts!

Exercise 6 – Create the possibility to create a new blog

Create a form on the list blog view template where a web user can create a new blog by specifying the title of the blog and click a submit button.

Create a controller method that is mapped to the URL from the form in the view. This method should create a new Blog object from the request parameter that is coming in with the call to the method, and send this blog object to a method (that you must also create) in the **JdbcBlogRepository,** that creates a new post in the blog table with the new title.

Create the method in the repository that inserts the new post to the blog table.

Stretch Task 1 – Create the possibility for the user to write new blog posts

Create the view template, controller method, and repository method to insert new blog posts to the posts table. The solution to this exercise looks a lot like the previous one.

Stretch Task 2 – Secure the functionality in exercise 6 and 7 with a Login

Don't let anyone create a blog or a blog post. Protect these methods and views with a login by moving the forms to separate views and check if the user is logged in before showing these views. If anyone tries to access this functionality send them to a login page with a login form (like in the Login and cookies exercises).

Only an existing author should be able to create a new blog and new blog posts for that blog.

Check that the username and password that the user specifies in the login form corresponds to the username and password of an author in the author table. You could try execute a select where username and password is equal to the specified login information, and if you get one author back, then you have a match!

When writing a new blog post, you should check that it is the author of the blog that is logged in! Only the author of the blog should be able to write a blog post, it's not enough that any author of the blogging platform is logging in.

A stretch task for this stretch task would be to give the author a possibility to add another author to a new blog, and possibly remove himself/herself as author. To do this, create the necessary view, controller method and repository method, and protect everything with the login.

Strech Task 3 – Change the database

Change the database by changing the application.properties file. Comment out he H2 configuration and add a connection URL in the following format:

```
spring.datasource.url=jdbc:sqlserver://185.21.146.24;databasename=DDDDDDDD;
user=XXXXXXX;password=YYYYYYY
```

Spring will use the **spring.datasource.url** to create a new **DataSource** object that we are autowiring into the **JdbcBlogRepository**. We are connecting to the database with this **DataSource** object, and will not need to create it ourselves.

Also, you have to comment out the **public void run(String… strings)** method in the **JdbcLabApplication** class. This method is for setting up the H2 database every time the application starts up, and if you use another database you probably don't want this method to execute.

So, comment out this method and change the connection string, and the application will use another database.

For everything to work you must create the necessary tables, but for the database connectivity check, you must only get the connection string right.

And for the first page with the list of blogs to work, you only need the blog table in the new database. Try this out and see if it works!

Stretch Task 4 – Add some design to the web pages

Add some **CSS** to make the web pages look a little better. You can add static **CSS** files or **JS** files to the **resources/static** directory and they will be accessible directly from the browser and any html file. The **resources/static** directory is the root of the web server, so if you add a **style.css** file here, the link in the html file will simply be "**style.css**".

Stretch Task 5 – Add some other functionality to this starter application

Try to add your own functionality, and just use this application as a start point!

Stretch Task 6 – Create another application

Create another application and just use this application as an inspiration!