Getting started with Dropwizard

Dropwizard is a functionality-focused framework aimed at developing robust and stable Java-based web services. It promises support for modern configuration, ops tools, metrics logging, and more.
Dropwizard's open-source licensing (Apache 2.0) means it has virtually endless community support and resources. Dropwizard is an ideal framework for RESTful web services that require superior stability. Companies like Fidelity, HubSpot, and American Express use Dropwizard for its ability to create safe and reliable web services and APIs.

Course audience
- people with some basic knowledge of Java and SQL
- people with previous knowledge of full profile application server

Course coverage
- Dropwizard combines several Java frameworks
- Cool for building RESTful APIs and micro services
- Created by Coda Hale

Course curriculum
- REST architecture style tour
- Creating a Dropwizard project using CLI, IDE
- "Hello world"
- Running the application
- Checking output using cURL, Postman
- Unit testing
- Basic authentication
- Configuration
- Integration testing
- Migrations
- Hibernate
- Resources

Course expectations
- Build an example application to store bookmarks
- Do assignments at the end of the sections
- Be able to create a simple micro service using Dropwizard

Dropwizard documentation
- https://www.dropwizard.io/en/latest/

Introduction to REST

Introduction
- REST - REpresentional state transfer
- Introduced in Dr. Fielding's thesis "Architectural Styles and the Design of Network-based Software Architectures"
- A Resource
- A representation
- Transfer

REST Basics
- Uniform Resource Identifier (URI)
- HTTP Protocol
- Client: server-side application, in-browser JavaScript application, cURL
- Client-Server
- Stateless

Richardson Maturity Model

Richardson Maturity Model
- Introduced by Leonard Richardson in 2008 QCon talk
- Later explained by Martin Fowler

- Level 0 - single URI and POST method (example: SOAP web service - not RESTful at all)
- Level 1 - resource identifiers
  - /users
  - /users/**1**
  - /users/**Jessica**
  - /users/**1f30f437-0691-4394-83ed-9ea9747d8cf2**
  - /users**?page=10**
  - /users/1/bookmarks
  - /users/1/bookmarks/3
  - /bookmarks
  - /bookmarks/3
- Level 2 - HTTP verbs are used: GET, POST, DELETE, etc.
  - GET /bookmarks
  - GET /bookmarks/3 -> 404 or 200 or ...
  - POST /bookmarks
  - DELETE /bookmarks/3
  - PUT /bookmarks/4

HATEOAS

HATEOAS stands for **H**ypermedia **A**s **T**he **E**ngine **O**f **A**n **A**pplication **S**tate, and a simplistic explanation of what it is would be that you should add hyperlinks to resource representations. Currently, there is no consensus on how hyperlinks should be added, but there is a lot of formats proposed. They include but not limited to HAL, JSON API, JSON-LD, UBER.

Why one would like to add hyperlinks to representations? Let's take a look at a representation of the list of bookmarks of some user.

Creating a Dropwizard project using Maven and CLI

```
mvn archetype:generate \
-DgroupId=com.pros \
-DartifactId=DropBookmarks \
-Dpackage=com.pros.dropbookmarks \
-Dname=DropBookmarks \
-DarchetypeGroupId=io.dropwizard.archetypes \
-DarchetypeArtifactId=java-simple \
-DarchetypeVersion=0.8.2 \
-DinteractiveMode=false
```

```
.
└── DropBookmarks
    ├── pom.xml
    └── src
        ├── main
        │   ├── java
        │   │   └── com
        │   │       └── pros
        │   │           ├── dropbookmarks
        │   │           │   ├── api
        │   │           │   ├── cli
        │   │           │   ├── client
        │   │           │   ├── core
        │   │           │   ├── db
        │   │           │   ├── health
        │   │           │   └── resources
        │   │           ├── DropBookmarksApplication.java
        │   │           └── DropBookmarksConfiguration.java
        │   ├── resources
        │   │   └── assets
        │   └── banner.txt
        └── test
            ├── java
            │   └── com
            │       └── pros
            │           └── dropbookmarks
            │               ├── api
            │               ├── cli
            │               ├── client
            │               ├── core
            │               ├── db
            │               └── resources
            └── resources
                └── fixtures
```

https://tree.nathanfriend.io/

Running a Dropwizard project from CLI and IntelliJ IDEA

Build + Run with maven:

```
astoev-pc/DropBookmarks % mvn package
astoev-pc/DropBookmarks % java -jar target/DropBookmarks-1.0-SNAPSHOT.jar server

anstoev@terminal-MacBook-Pro ~ % curl -w "\n" localhost:8080/greeting

astoev-pc/DropBookmarks % ^C
```

Build+Run with IntelliJ:

Run > Edit Configurations... > Add New Configuration (⌘N) > Application > MyAppName

Name: DropBookmarks
Main class: org.pros.DropBookmarksApplication
Program arguments: server
Program arguments: server config.yml

^R

Adding tests to the project

```
<dependency>
    <groupId>io.dropwizard</groupId>
    <artifactId>dropwizard-testing</artifactId>
    <version>${dropwizard.version}</version>
    <scope>test</scope>
</dependency>
```

Creating a unit test

```
import org.junit.jupiter.api.Disabled;
import org.junit.jupiter.api.Test;

@Test
@Disabled
public void testGetGreeting() {
    fail("Not yet implemented");
}
```

Testing a resource class using in-memory Jersey

```
<dependency>
    <groupId>org.glassfish.jersey.test-framework</groupId>
    <artifactId>jersey-test-framework-core</artifactId>
    <scope>test</scope>
</dependency>

<dependency>
    <groupId>org.glassfish.jersey.test-framework.providers</groupId>
    <artifactId>jersey-test-framework-provider-grizzly2</artifactId>
    <scope>test</scope>
</dependency>
```

```
@Test
public void testGetGreeting() {
    final String expected = "Hello Dropwizard!";

    Response response = target("/greeting").request().get();

    assertEquals(
            HttpStatus.OK_200,
            response.getStatus(),
            "Http Response status should be: 200 OK"
    );

    assertEquals(
            MediaType.TEXT_PLAIN,
            response.getHeaderString(HttpHeaders.CONTENT_TYPE),
            "Http Content-Type should be: TEXT_PLAIN"
    );

    String content = response.readEntity(String.class);
    assertEquals(
            expected,
            content,
            String.format("Content of response expected to be: %s", expected)
    );
}
```

Basic Authentication

How to add basic authentication to our project? In a nutshell, basic authentication is when user needs to provide his user and password to gain access to some resource. Adding authentication to Dropwizard project is a 3 step project:

- Register the Authenticator class (User) in the application environment
- Register AuthDynamicFeature and RolesAllowedDynamicFeature in the application environment
- Instruct App to prompt user for credentials for secured method

Adding a class implementing Authenticator interface

```
<dependency>
   <groupId>io.dropwizard</groupId>
   <artifactId>dropwizard-auth</artifactId>
</dependency>
```

User class in existing **core** package.
Create **HelloAuthenticator** class in newly created package **auth**.

```
public class ExampleAuthenticator implements Authenticator<BasicCredentials, User> {
   /**
    * Valid users with mapping user -> roles
    */
   private static final Map<String, Set<String>> VALID_USERS = Map.of(
         "guest", Collections.emptySet(),
         "good-guy", Collections.singleton("BASIC_GUY"),
         "chief-wizard", Set.of("ADMIN", "BASIC_GUY")
   );

   @Override
   public Optional<User> authenticate(BasicCredentials credentials) {
      if (VALID_USERS.containsKey(credentials.getUsername()) &&
"secret".equals(credentials.getPassword())) {
         return Optional.of(new User(credentials.getUsername(),
VALID_USERS.get(credentials.getUsername())));
      }

      return Optional.empty();
   }
}

public class ExampleAuthorizer implements Authorizer<User> {

   @Override
   public boolean authorize(User user, String role) {
      return user.getRoles() != null && user.getRoles().contains(role);
   }
}
```

Register in Application environment

```
environment.jersey().register(new AuthDynamicFeature(new
BasicCredentialAuthFilter.Builder<User>()
      .setAuthenticator(new ExampleAuthenticator())
      .setAuthorizer(new ExampleAuthorizer())
      .setPrefix("Basic")
      .setRealm("SUPER SECRET STUFF")
      .buildAuthFilter()));
```

```
environment.jersey().register(new AuthValueFactoryProvider.Binder<>(User.class));
environment.jersey().register(RolesAllowedDynamicFeature.class);
```

Securing a method

```
@GET
@Produces(MediaType.TEXT_PLAIN)
@Path("/secured")
public String getSecureGreeting(@Auth User user) {

    return "Hello Secure Dropwizard!";
}
```

Checking a secured resource using Postman

```
http://localhost:8080/greeting/secured
Credentials are required to access this resource.
Status: 401 Unauthorized
```

In Postman App: add KEY **Authorization** with VALUE **Basic <base64encodedPassword>** , where you can encode your username and password here: https://www.base64encode.org/. Paste "guest:secret" and encode it. Paste the returned string in the value with prefix "Basic " (don't forget the space). Send request.

```
http://localhost:8080/greeting/secured
Hello Secure Dropwizard!
Status: 200 OK
```

Alternatively, you can choose Basic Auth from the drop down menu in Authorization tab in Postman and write the username and password. Postman will automatically encode them and add them to the Header of the request.

Checking a secured resource using cURL

```
anstoev@terminal-MacBook-Pro ~ % curl -w "\n" localhost:8080/greeting/secured
Credentials are required to access this resource.

anstoev@terminal-MacBook-Pro ~ % curl -w "\n" localhost:8080/greeting/secured -i
HTTP/1.1 401 Unauthorized
Date: Fri, 10 Mar 2023 19:08:45 GMT
WWW-Authenticate: Basic realm="SUPER SECRET STUFF"
Content-Type: text/plain
Content-Length: 49

Credentials are required to access this resource.
Status: 401 Unauthorized

anstoev@terminal-MacBook-Pro ~ % curl -w "\n" localhost:8080/greeting/secured -i \
-H "Authorization: Basic Z3Vlc3Q6c2VjcmV0"

HTTP/1.1 200 OK
Date: Fri, 10 Mar 2023 19:11:13 GMT
Content-Type: text/plain
Vary: Accept-Encoding
Content-Length: 24

Hello Secured Dropwizard!
Status: 200 OK
```

Alternatively, we can use: curl -w "\n" localhost:8080/greeting/secured -i -u guest:secret

Creating a unit test for password-protected resource

1. Create User class that implements Principal interface.
2. Create ExampleAuthorizer and ExampleAuthenticator classes in auth package (create the package, too).
3. In the test class set the TestContainerFactory with GrizzlyWebTestContainerFactory
4. Test

Configuration and HTTPS

Adding a YAML configuration file

We place our .yaml or just .yml configuration files to the root folder of our project.

```
public class DropBookmarksConfiguration extends Configuration {
    @NotEmpty
    private String password;

    @JsonProperty
    public String getPassword() {
        return password;
    }
}
```

Create a constructor in ExampleAuthenticator

```
private final String password;

public ExampleAuthenticator(String password) {
    this.password = password;
}
```

Reading from YAML in Resource and Test Classes

In pom.xml:

```
<dependency>
    <groupId>io.dropwizard</groupId>
    <artifactId>dropwizard-testing</artifactId>
    <version>${dropwizard.version}</version>
    <scope>test</scope>
</dependency>
```

In **build** in pom.xml:

```
        <testResources>
            <testResource>
                <directory>${project.basedir}/src/test/resources</directory>
            </testResource>
            <testResource>
                <directory>${project.basedir}</directory>
                <includes>
                    <include>config.yml</include>
                </includes>
            </testResource>
        </testResources>
```

In Test class:

```
@ExtendWith(DropwizardExtensionsSupport.class)
```

```
public class ExampleResourceTest {
   private static YamlConfigReaderConfiguration configuration;
   private static ObjectMapper objectMapper;
   private final ResourceExtension EXT = ResourceExtension.builder()
         .addResource(new ExampleResource(configuration))
         .build();
   @BeforeAll
   static void setUp() throws ConfigurationException, IOException {
      objectMapper = Jackson.newObjectMapper();
      final Validator validator = Validators.newValidator();
      final YamlConfigurationFactory<YamlConfigReaderConfiguration> factory =
            new YamlConfigurationFactory<>(YamlConfigReaderConfiguration.class, validator,
objectMapper, "dw");

      final File yaml = new File(
            Objects.requireNonNull(
                  Objects.requireNonNull(
                        Thread.currentThread()
                              .getContextClassLoader()
                              .getResource("config.yml")
                  ).getPath()
            )
      );

      configuration = factory.build(yaml);
   }
```

In example resource:

```
@Path("/{parameter: yaml|yml}")
public class ExampleResource {
   private final YamlConfigReaderConfiguration config;

   public ExampleResource(YamlConfigReaderConfiguration configuration) {
      this.config = configuration;
   }
```

In Application class:

```
@Override
public void run(final YamlConfigReaderConfiguration configuration,
            final Environment environment) {
   environment.jersey().register(new ExampleResource(configuration));
}
```

A brief introduction to HTTPS

HTTP is used to make connections to our applications encrypted. HTTPS is used for payment transactions and in the case of our application it may be used to transmit credentials (Base64 encryption can be easily reversed).

From the root directory of our project:

```
anstoev@terminal-MacBook-Pro DropBookmarks % \
keytool -genkeypair \
-keyalg RSA \
-dname "CN=localhost" \
-keystore dropbookmarks.keystore \
-keypass p@ssw0rd \
-storepass p@ssw0rd
Generating 3,072 bit RSA key pair and self-signed certificate (SHA384withRSA) with a validity of
90 days
      for: CN=localhost
```

```
anstoev@terminal-MacBook-Pro DropBookmarks % ls
README.md                    config-prod.yml              dependency-reduced-pom.xml      pom.xml
target
config-dev.yml               config.yml                   dropbookmarks.keystore          src
```

Adding HTTPS

```
## YAML Template.
#password
password: p@ssw0rd

server:
  applicationConnectors:
    - type: http
      port: 8080
    - type: https
      port: 8443
      keyStorePath: dropbookmarks.keystore
      keyStorePassword: p@ssw0rd
      validateCerts: false
```

```
curl -w "\n" https://localhost:8443/greeting -k
```

https://localhost:8443/greeting
(Advanced > proceed to localhost unsafely)

Adding an Integration Test

Integration Testing using HTTPS

Configuring Database Connection

```
database:
  driverClass: com.mysql.jdbc.Driver
  url: jdbc:mysql://localhost:3306/dropbookmarks?
allowPublicKeyRetrieval=true&useSSL=false&createDatabaseIfNotExist=true&serverTimezone=UT
C&useUnicode=true&characterEncoding=UTF-8
  user: root
  password: root
```

In pom.xml:

```
<dependency>
    <groupId>com.mysql</groupId>
    <artifactId>mysql-connector-j</artifactId>
    <version>8.0.32</version>
</dependency>
```

Database Migrations with Liquibase

Introduction to Database Migrations with Liquibase

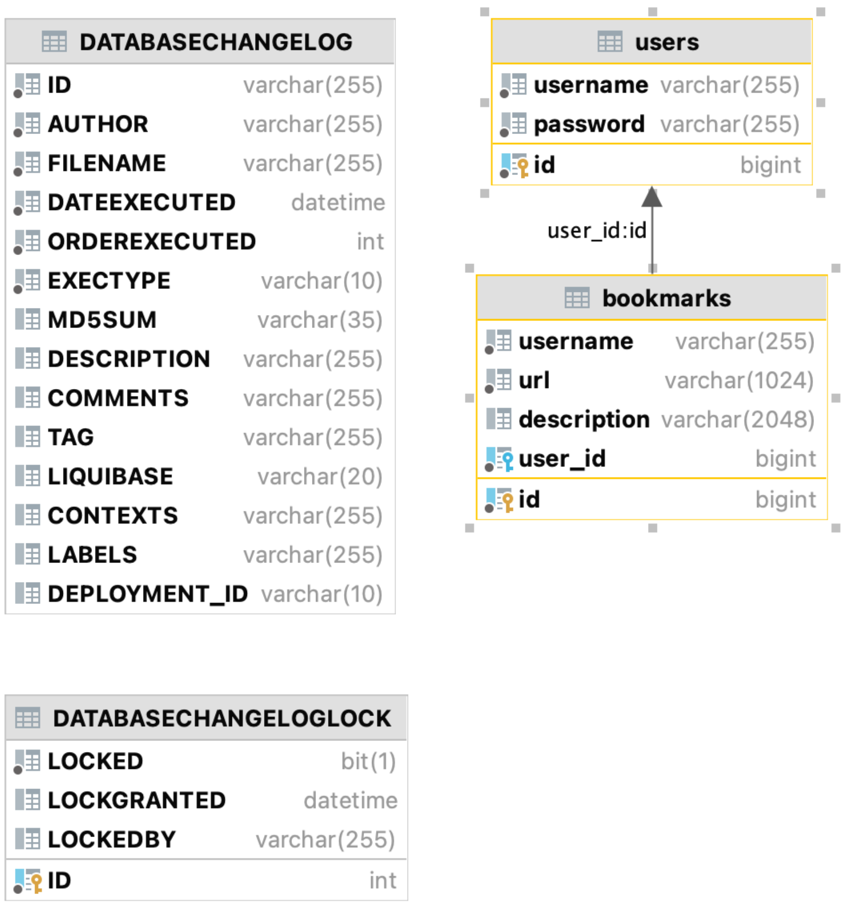
https://www.liquibase.org/

Adding Liquibase Maven Support

/usr/local/opt/liquibase
Add Liquibase to path
mvn liquibase:update

mvn liquibase:dropAll

Adding more changesets



DATABASECHANGELOG	
ID	varchar(255)
AUTHOR	varchar(255)
FILENAME	varchar(255)
DATEEXECUTED	datetime
ORDEREXECUTED	int
EXECTYPE	varchar(10)
MD5SUM	varchar(35)
DESCRIPTION	varchar(255)
COMMENTS	varchar(255)
TAG	varchar(255)
LIQUIBASE	varchar(20)
CONTEXTS	varchar(255)
LABELS	varchar(255)
DEPLOYMENT_ID	varchar(10)

users	
username	varchar(255)
password	varchar(255)
id	bigint

bookmarks	
username	varchar(255)
url	varchar(1024)
description	varchar(2048)
user_id	bigint
id	bigint

DATABASECHANGELOGLOCK	
LOCKED	bit(1)
LOCKGRANTED	datetime
LOCKEDBY	varchar(255)
ID	int

Applying database refactoring using Maven

```
.
└─── DropBookmarks
     ├─── pom.xml
     └─── src
          └─── main
               ├─── java
               |    └─── ...
               ├─── resources
               |    ├─── assets
               |    └─── db
               |         └─── changelog
               |              ├─── 01-CreateTables.xml
               |              └─── 02-AddUsers.xml
               ├─── 00-migrations
               └─── banner.txt
```

In plugins in pom.xml:

```
<plugin>
    <groupId>org.liquibase</groupId>
    <artifactId>liquibase-maven-plugin</artifactId>
    <version>4.5.0</version>
    <configuration>
        <contexts>DEV</contexts>
        <changeSetPath>classpath:db.changelog</changeSetPath>
        <changeLogFile>00-migrations.xml</changeLogFile>
        <driver>com.mysql.cj.jdbc.Driver</driver>
        <url>jdbc:mysql://localhost:3306/dropbookmarks?createDatabaseIfNotExist=true</url>
        <username>root</username>
        <password>root</password>
    </configuration>
</plugin>
```

In dependencies:

```
<dependency>
    <groupId>org.liquibase</groupId>
    <artifactId>liquibase-core</artifactId>
    <version>4.19.1</version>
</dependency>
```

00-migrations.sql

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<databaseChangeLog logicalFilePath="db.changelog-master.xml" xmlns="http://www.liquibase.org/xml/ns/
dbchangelog"
                   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                   xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog http://www.liquibase.org/xml/ns/
dbchangelog/dbchangelog-3.4.xsd">

<!--    <include file="./db.changelogs/01-CreateTables.xml" relativeToChangelogFile="true"/>-->
<!--    <include file="./db.changelogs/02-AddUsers.xml" relativeToChangelogFile="true"/>-->

    <includeAll path="src/main/resources/db.changelogs/"/>
</databaseChangeLog>
```

01-CreateTables.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<databaseChangeLog logicalFilePath="db.changelog-master.xml" xmlns="http://www.liquibase.org/xml/ns/
dbchangelog"
                   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                   xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog http://www.liquibase.org/xml/ns/
dbchangelog/dbchangelog-3.4.xsd">

    <changeSet id="1" author="astoev" context="DEV">
        <createTable tableName="users">
            <column name="id" type="bigint" autoIncrement="true">
                <constraints primaryKey="true" nullable="false"/>
            </column>
            <column name="username" type="varchar(255)">
                <constraints nullable="false"/>
            </column>
            <column name="password" type="varchar(255)">
                <constraints nullable="false"/>
            </column>
        </createTable>
        <comment>A script to create a users table</comment>
    </changeSet>

    <changeSet id="2" author="astoev" context="DEV">
        <createTable tableName="bookmarks">
            <column name="id" type="bigint" autoIncrement="true">
                <constraints primaryKey="true" nullable="false"/>
            </column>
            <column name="username" type="varchar(255)">
                <constraints nullable="false"/>
            </column>
            <column name="url" type="varchar(1024)">
                <constraints nullable="false"/>
            </column>
            <column name="description" type="varchar(2048)"/>
            <column name="user_id" type="bigint">
                <constraints nullable="false"
                             foreignKeyName="fk_users_id"
                             referencedTableName="users"
                             referencedColumnNames="id"/>
            </column>
        </createTable>
        <comment>A script to create a bookmarks table</comment>
    </changeSet>
</databaseChangeLog>
```

02-AddUsers.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<databaseChangeLog logicalFilePath="db.changelog-master.xml" xmlns="http://www.liquibase.org/xml/ns/
dbchangelog"
                   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                   xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog http://www.liquibase.org/xml/ns/
dbchangelog/dbchangelog-3.4.xsd">

    <changeSet id="3" author="astoev" context="DEV">

        <insert tableName="users">
            <column name="id" value="1"/>
            <column name="username" value="pesho"/>
            <column name="password" value="1234"/>
        </insert>

        <insert tableName="users">
            <column name="id" value="2"/>
            <column name="username" value="maria"/>
            <column name="password" value="1234"/>
        </insert>

        <insert tableName="users">
            <column name="id" value="3"/>
            <column name="username" value="gosho"/>
            <column name="password" value="1234"/>
        </insert>

        <rollback>
            <delete tableName="users">
                <where>id &lt; 4</where>
            </delete>
        </rollback>

        <comment>A script to add data</comment>
    </changeSet>

</databaseChangeLog>
```

Instruct liquibase to output SQL generator from out change log to a file:

```
mvn liquibase:update \
-Dliquibase.contexts=DEV
```

(before each mvn liquibase:{command})

```
cd target/liquibase
```

Applying database refactoring using Maven

```
mvn liquibase:rollback \
-Dliquibase.rollbackCount=1
```

```
-Dliquibase.rollbackTag=1.0
"-Dliquibase.rollbackDate=Jun 03, 2017"
```

In resources:
liquibase.properties

In pom.xml:

```
<plugin>
    ...
    <configuration>
        <propertyFile>src/main/resources/liquibase.properties</propertyFile>
```

```
mvn liquibase:generateChangeLog
```

Adding Dropwizard Liquibase support:

```
<dependency>
    <groupId>io.dropwizard</groupId>
    <artifactId>dropwizard-migrations</artifactId>
</dependency>
```

```
@Override
public void initialize(final Bootstrap<DropBookmarksConfiguration> bootstrap) {
    bootstrap.addBundle(new MigrationsBundle<>() {
        @Override
        public DataSourceFactory getDataSourceFactory(DropBookmarksConfiguration
configuration) {
            return configuration.getDataSourceFactory();
        }
    });
}
```

```
mvn package
```

Applying database refactorings using application's CLI:

```
java -jar target/DropBookmarks-1.0-SNAPSHOT.jar \
db status config.yml
```

```
java -jar target/DropBookmarks-1.0-SNAPSHOT.jar \
db migrate -i DEV config.yml
```

Connecting to a Relational Database

https://www.dropwizard.io/en/latest/manual/hibernate.html

Adding Hibernate support to the Application class

```
private final HibernateBundle<DropBookmarksConfiguration> hibernate = new
HibernateBundle<>(UserEntity.class, BookmarkEntity.class) {
   @Override
   public PooledDataSourceFactory getReadSourceFactory(DropBookmarksConfiguration
configuration) { return getDataSourceFactory(configuration); }
   @Override
   public DataSourceFactory getDataSourceFactory(DropBookmarksConfiguration configuration) {
      return configuration.getDataSourceFactory(); }
};

@Override
public void initialize(final Bootstrap<DropBookmarksConfiguration> bootstrap) {
   bootstrap.addBundle(hibernate);
}

@Override
public void run(final DropBookmarksConfiguration configuration,
            final Environment environment) {

   final UserDAO daoUserEntity = new UserDAO(hibernate.getSessionFactory());
   environment.jersey().register(new UserResource(daoUserEntity));

   final BookmarkDAO daoBookmarkEntity = new BookmarkDAO(hibernate.getSessionFactory());
   environment.jersey().register(new BookmarkResource(daoBookmarkEntity));
...
```

Using database for authentication

```
<dependency>
   <groupId>org.jasypt</groupId>
   <artifactId>jasypt</artifactId>
   <version>1.9.2</version>
</dependency>

private final PasswordEncryptor passwordEncryptor
      = new BasicPasswordEncryptor();
```

Creating Resources

How Dropwizard deals with exceptions in resource methods

```
   /**
    * Method looks for a bookmark by id and User id and returns the bookmark or
    * throws NotFoundException otherwise.
    *
    * @param id the id of a bookmark.
    * @param user the id of the owner.
    * @return Bookmark
    */
   private Bookmark findBookmarkOrTrowException(IntParam id,
         @Auth User user) {
      Bookmark bookmark = bookmarkDAO.findByIdAndUserId(
              id.get(), user.getId()
      ).orElseThrow(()
              -> new NotFoundException("Bookmark requested was not found."));
      return bookmark;
   }
```