Setting up access to the database

Add the PostgreSQL dependency

Add the dependency for the driver for PostgreSQL database to your pom.xml file:

```
    …
<dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
</dependency>
    …
```

Connecting your application to a database requires providing credentials and a database URL. The values from **application.properties** will be used unless they are overridden. In the default properties I specified the database dialect.

```
# src/main/resources/application.properties
…
spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect
```

If you place here values used for connecting to the local database, the app will work on your machine. In case you forget to supersede them on the production, the app won't be able to connect to the "real" database.

Properties explained

```
spring.jpa.database-platform=“…”
```

The dialect to use is detected by the JPA provider. If you prefer to set the dialect yourself, set the `spring.jpa.database-platform` property.

```
spring.datasource.url=“…”
spring.datasource.username=“…”
spring.datasource.password=“…”
```

`spring.datasource.url` is the JDBC URL of the database. Remember to keep the port, database name and credentials consistent with the database configuration. Make sure that the credentials from the dev environment are not accidentally used for the production.

Properties explained

```
javax.persistence.schema-generation.database.action="…"
```

`javax.persistence.schema-generation.database.action` property define which action should be performed automatically on application startup. The default value is none, available options are:

- **create**
  Database creation will be generated.

- **drop**
  Database dropping will be generated.

- **drop-and-create**
  Database dropping will be generated followed by database creation.

Properties explained

`spring.jpa.hibernate.ddl-auto` – the alternative property for creating a schema.

Instead of the `javax.persistence.schema-generation.database.action` property you could use the `spring.jpa.hibernate.ddl-auto` property to apply the JPA features for DDL generation on the application startup. The standard values are **none, validate, update, create, and create-drop**.

Be aware of differences in the default value for this property. As we can read in the Spring Boot documentation:

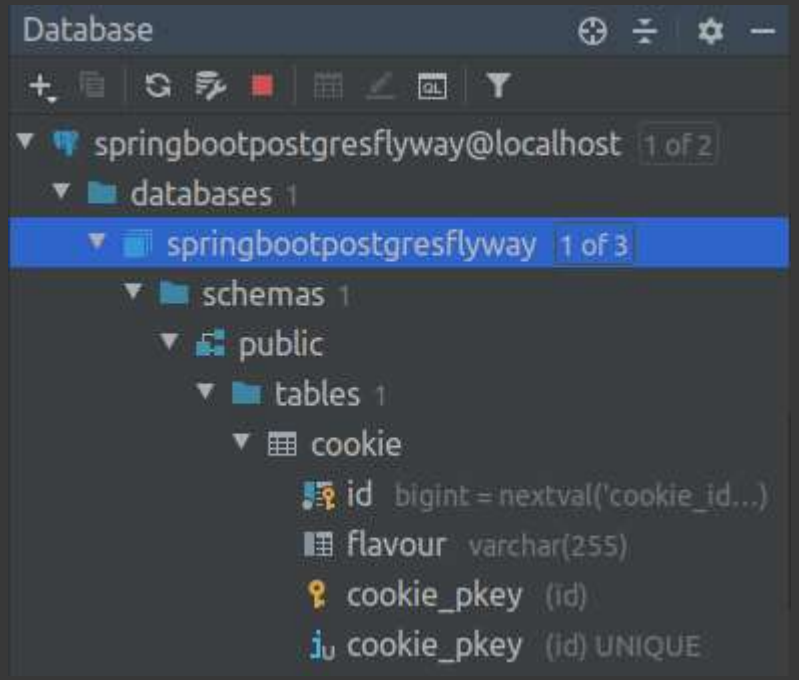
> *"The `spring.jpa.hibernate.ddl-auto` is a special case, because, depending on runtime conditions, it has different defaults. If an embedded database is used and no schema manager (such as Liquibase or Flyway) is handling the DataSource, it defaults to create-drop. In all other cases, it defaults to none"*

Create and verify database schema

We store the DDL script in a file so we can easily examine the commands that were run against the database on application startup. The **schema.sql** file generated in sample project looks like this:

```
-- schema.sql
create table cookie (
    id  bigserial not null,
    flavour varchar(255),
    primary key (id)
);
```

After application start database schema will look like on screenshot:



Disable automatic schema creation once the tables are initialized

You can keep recreating container and volume with the database content until you are satisfied with the outcome and all entities are polished. Once the schema is ready you can remove the property responsible for creating the tables – the `javax.persistence.schema-generation.database.action` property.

However, the schema will evolve during the project lifetime, so it is a good idea to implement a better solution for maintaining it. We explain how to add migrations to the project in the “Database migrations with Flyway” part.