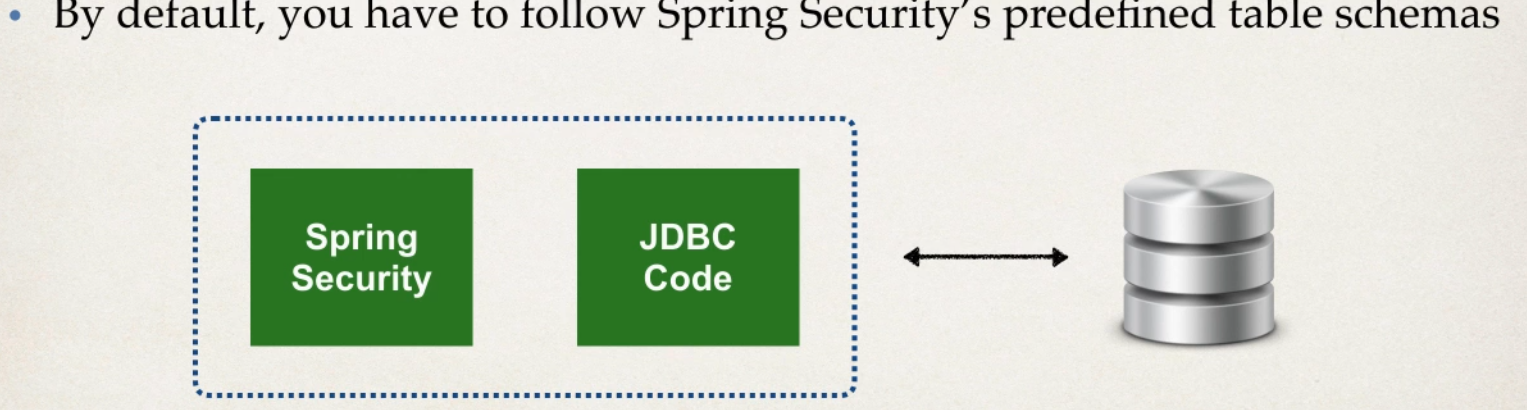
Spring Security Data base Support

Spring security can read data from user account info form DB

By Default you have to follow Spring security’s predefined table schemas



Customize Data Base Access with Spring Security

1 can also customize the table schemas

2 usefull if we have custom tables specific to project

3 ours will be responsible for developing the code to access the data

JDBC,Hibernate etc.,

Default DB in Spring Security

Develeopment Process

1 Develop SQL script to set up table

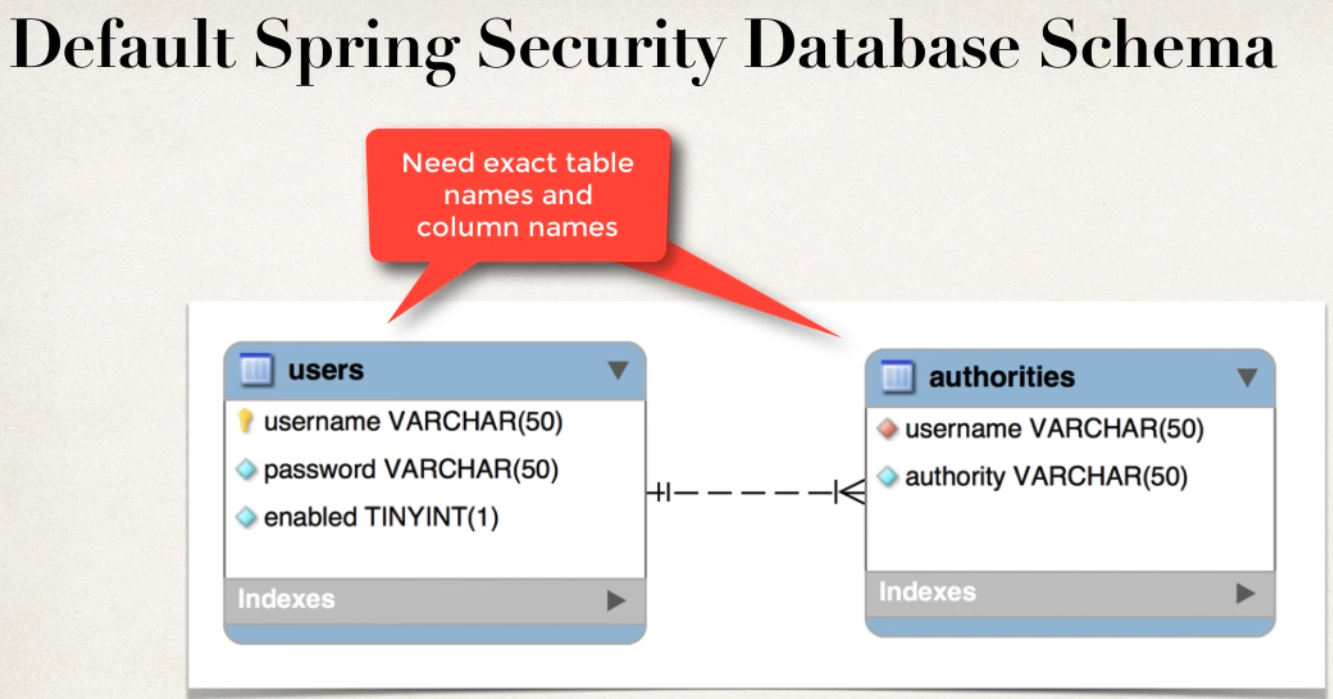
2 Add DB support to MAVEN POM file

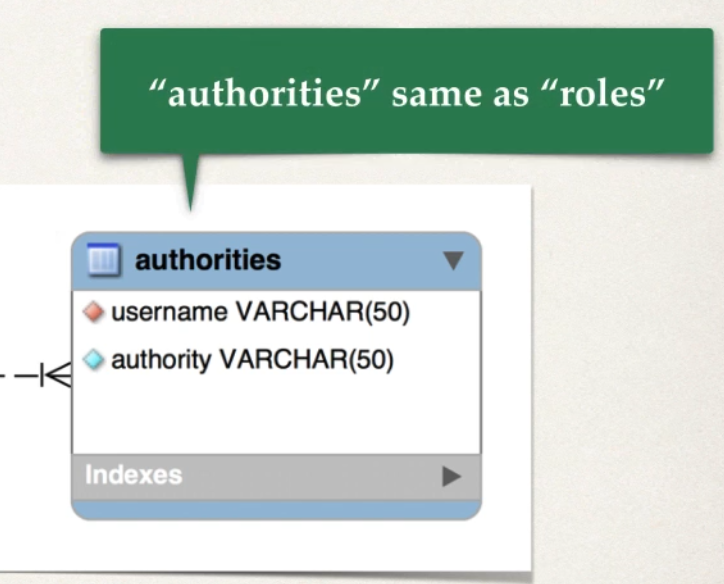
3 create JDBC property file

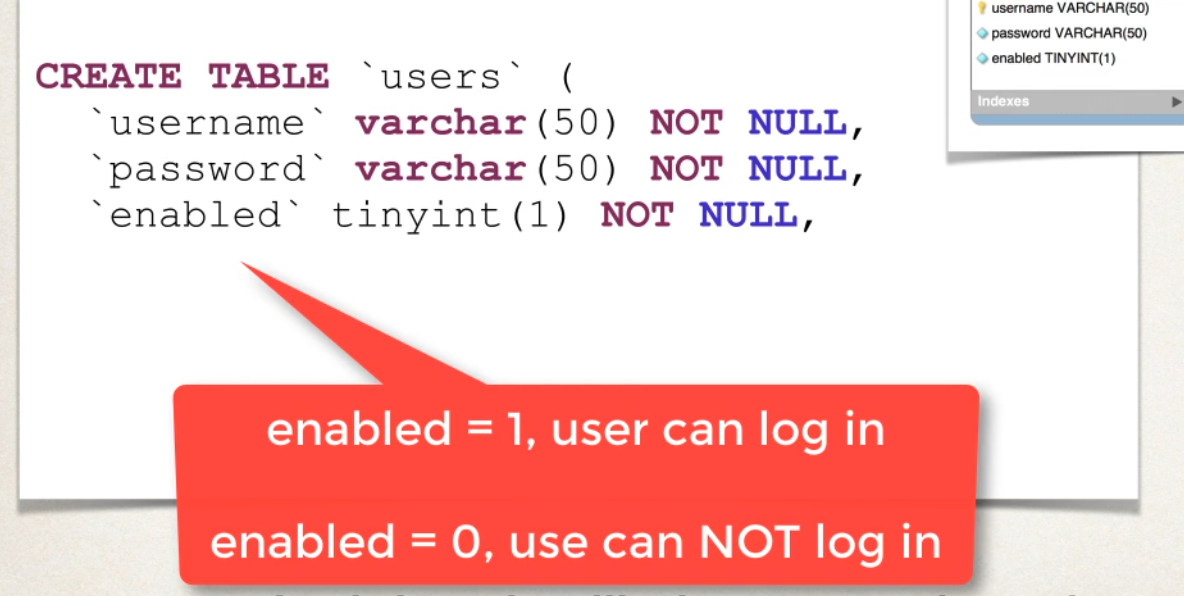
4 Define Data Source in Spring Configurtaion

5 Update Spring Security Configuration to use JDBC

Step 1







create table users (

username varchar(50) not null,

password varchar(50) not null,

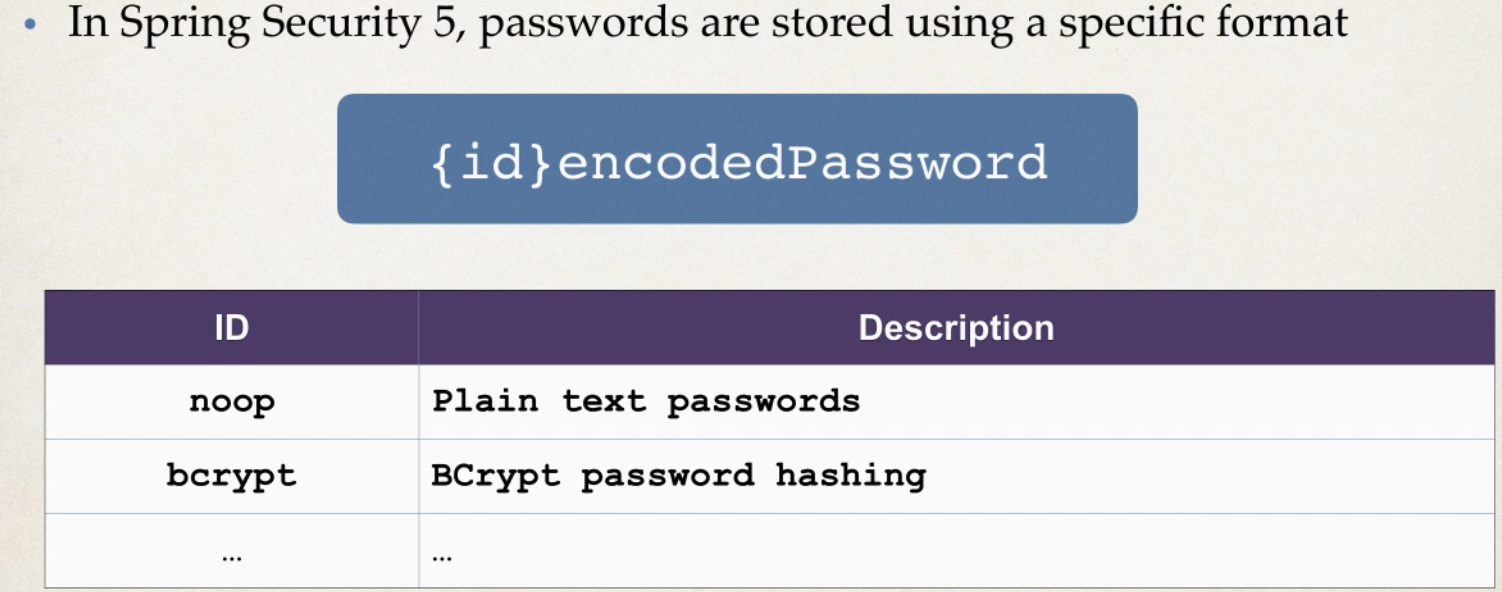
enabled NUMERIC(1) not NULL,

PRIMARY key (username)

)

Spring Security password storage

In Spring Security 5 password stored using specific format



Bcrypt is most widly used n recommnded from spring security also

insert into users values

('sachin','{noop}password',1),

('manasa','{noop}password',1),

('darsahn','{noop}password',1),

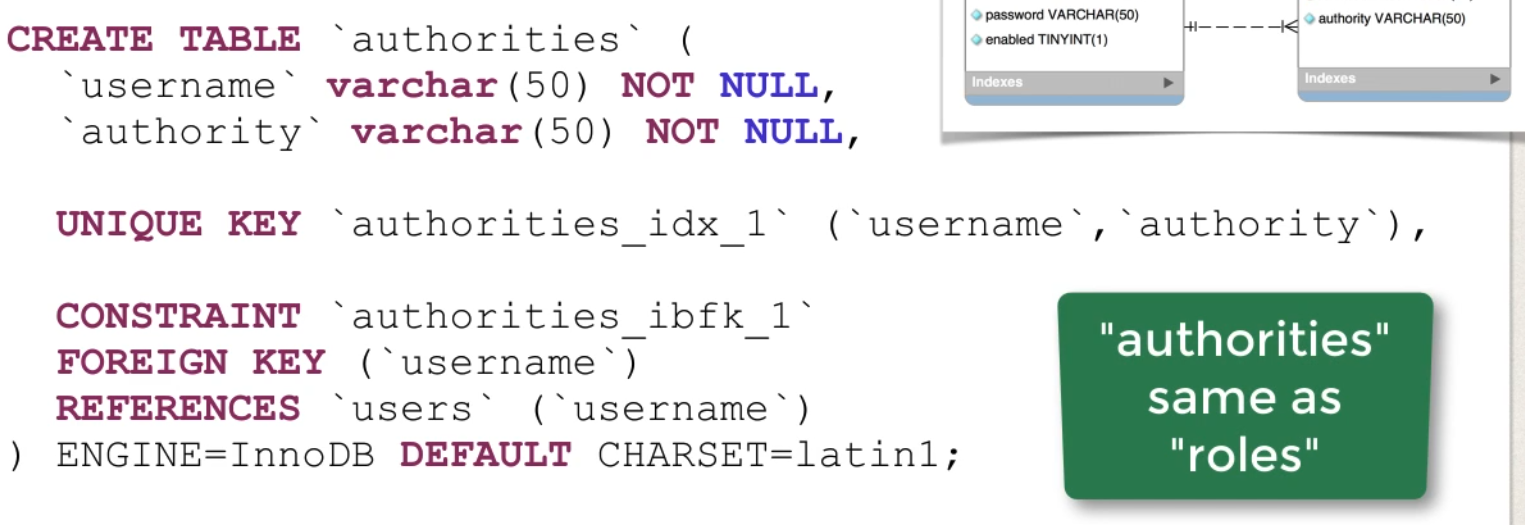
('sachinhs','{noop}password',1);



CREATE table authorities(username varchar(50) not null,authority varchar(50) not null,

CONSTRAINT authority\_foreign\_key FOREIGN KEY (username) REFERENCES users(username)

);



CREATE table authorities(username varchar(50) not null,authority varchar(50) not null,

UNIQUE KEY authority\_unique\_key (username,authority),

CONSTRAINT authority\_foreign\_key FOREIGN KEY (username) REFERENCES users(username)

);

insert into authorities VALUES

('sachin','ROLE\_EMPLOYEE'),

('darshan','ROLE\_EMPLOYEE'),

('manasa','ROLE\_EMPLOYEE'),

('sachinhs','ROLE\_EMPLOYEE'),

('sachin','ROLE\_ADMIN'),

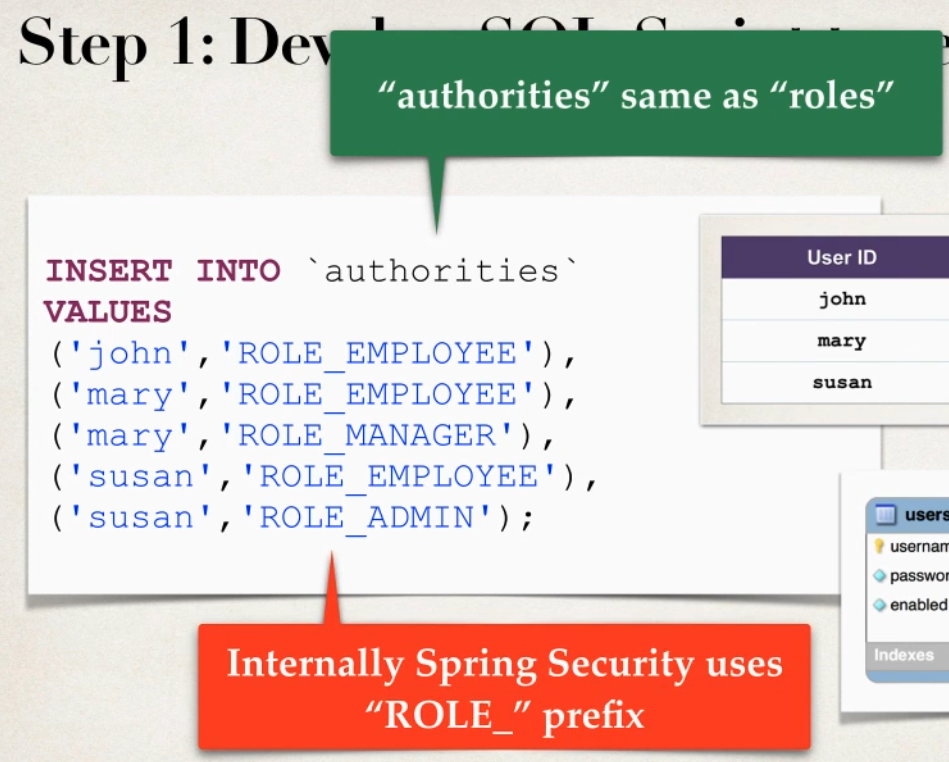
('manasa','ROLE\_MANAGER'),

('sachinhs','ROLE\_MANAGER'),

('sachinhs','ROLE\_ADMIN'),

('sachinhs','ROLE\_DEVELOPER'),

('sachinhs','ROLE\_IT');



Step 2 Add DB support in Maven

1 Postgress connection

<!-- postgresql -->

<dependency>

<groupId>org.postgresql</groupId>

<artifactId>postgresql</artifactId>

<version>${postgresql.connector.version}</version>

</dependency>

2 DB connection Pool

<!-- C3PO DB connection Pool -->

<dependency>

<groupId>com.mchange</groupId>

<artifactId>c3p0</artifactId>

<version>0.9.5.2</version>

</dependency>

Step 3 Create a JDBC Property file

Property file : src/main/resources/persistence-mysql.properties

Place this in property file

# JDBC connection Properties

jdbc.driver=org.postgresql.Driver

jdbc.url=jdbc:postgresql://localhost:5432/personal?currentSchema=crm

jdbc.user=postgres

jdbc.password=sachin

# Connection pool properties

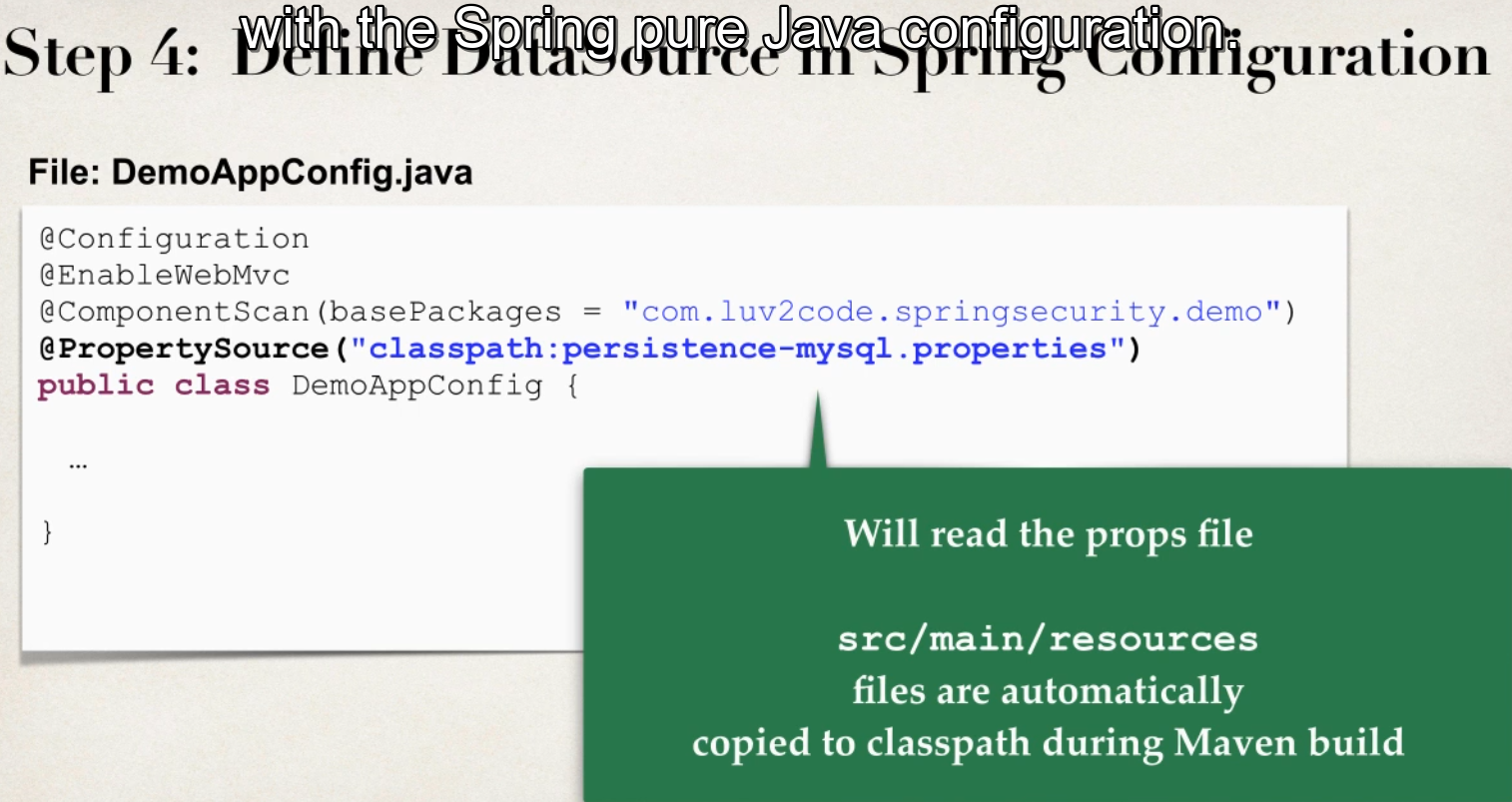
connection.pool.initialPoolSize=5

connection.pool.minPoolSize=5

connection.pool.maxPoolSize=20

connection.pool.maxIdleTime=3000

Step 4 Reading Property file and defing Data Source



- autowire Envoirnament class its spring helper class

- which will hold data read from properties files

- define data source object

- import javax.sql.DataSource;

// set up variable to hold the properties

@Autowired

private Environment env;

@Bean

public DataSource securityDataSource() {

// Create Connection Pool

ComboPooledDataSource securityDataSource=new ComboPooledDataSource();

// set JDBC driver

try {

myLogger.info(">>>> JDBC DRIVER =="+ env.getProperty("jdbc.driver"));

myLogger.info(">>>> JDBC URL =="+ env.getProperty("jdbc.url"));

myLogger.info(">>>> JDBC USER =="+ env.getProperty("jdbc.user"));

securityDataSource.setDriverClass(env.getProperty("jdbc.driver"));

securityDataSource.setJdbcUrl(env.getProperty("jdbc.url"));

securityDataSource.setUser(env.getProperty("jdbc.user"));

securityDataSource.setPassword(env.getProperty("jdbc.password"));

// set connection pool props

securityDataSource.setInitialPoolSize(Integer.parseInt(env.getProperty("connection.pool.initialPoolSize")));

securityDataSource.setMinPoolSize(Integer.parseInt(env.getProperty("connection.pool.minPoolSize")));

securityDataSource.setMaxPoolSize(Integer.parseInt(env.getProperty("connection.pool.maxPoolSize")));

securityDataSource.setMaxIdleTime(Integer.parseInt(env.getProperty("connection.pool.maxIdleTime")));

return securityDataSource;

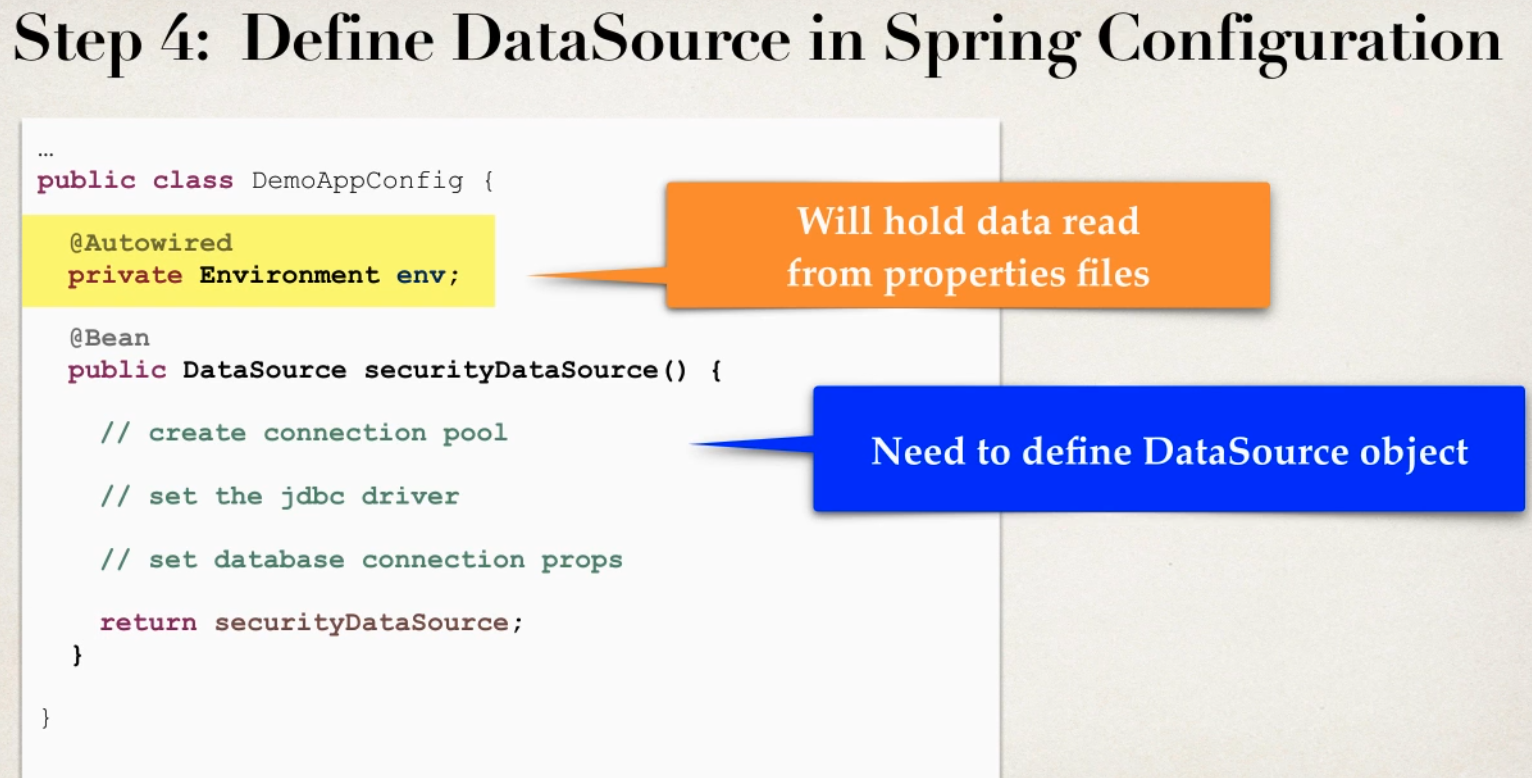
}

catch (Exception e) {

throw new RuntimeException(e);

}

}



Step 5 Update Spring Security to use JDBC

Inject Data Source we just Configured in step 4 into Spring Config Class

