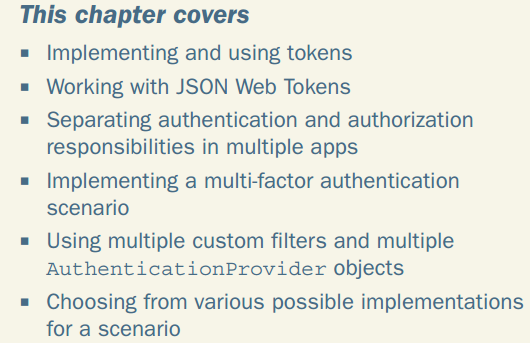
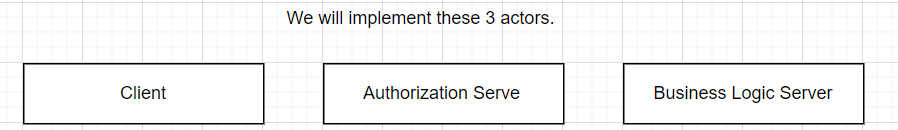
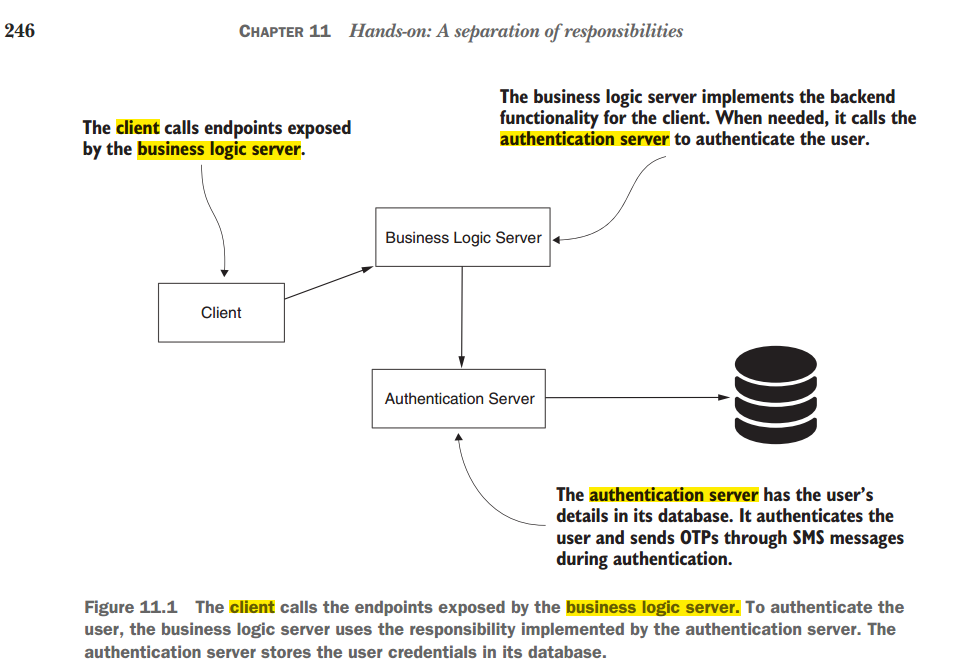
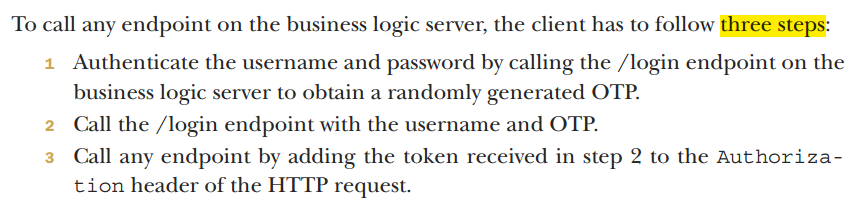
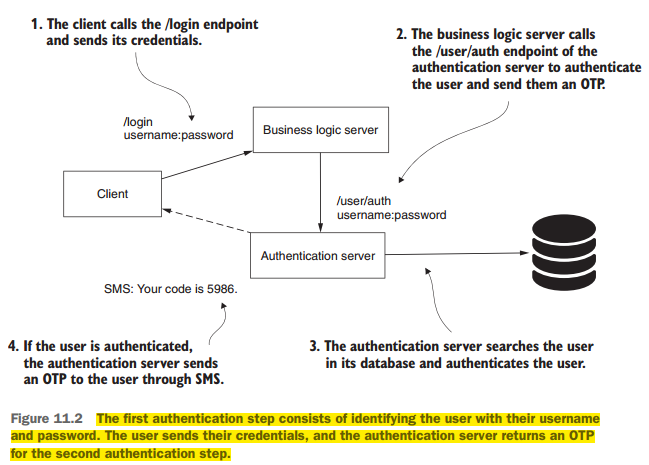
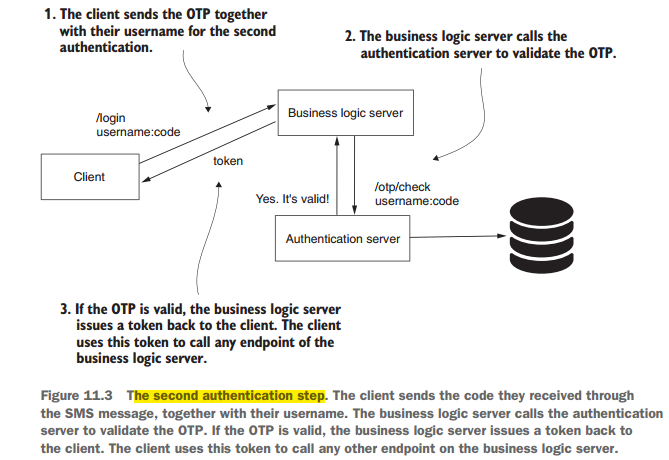
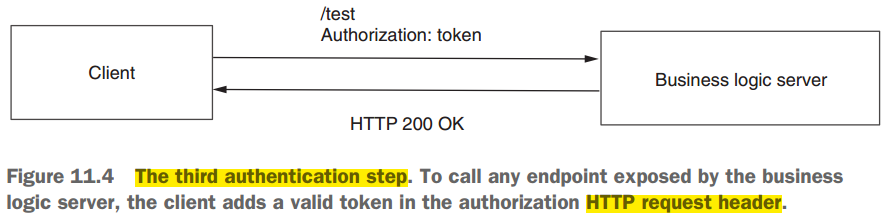
1. 
2. **Agenda**:
   1. We will design a system of three actors : A client, Authentication Server, The Business Logic Server.  
      

11.1 The scenario and requirements of the example

1. **Agenda**:
   1. Requirement for the system and how to approach and then the best option would be selected.
2. Three Components:
   1. Client:
      1. Application consuming the backend.
      2. Either Mobile App or UI implemented using vue.js or angular.  
         **NOTE**: We will use cURL.
   2. **Authorization Server**:
      1. An app with a DB of user credentials.
      2. Responsible to authenticate a user using credentials and send them an OTP through SMS.   
         **NOTE**: We will not be sending OTP through SMS but reading from DB.  
         **NOTE**: Alter on, we will be sending OTP using a service such as AWS SNS, Twillio, or others.
   3. **The business logic server**:
      1. The app exposing endpoints that our client consumes.
      2. We want to secure them.
         1. Authenticate (username & password), send OTP via SMS.
3. 
4. 
   1. This is the flow:
      1. Client sends Username and password to Business Logic Server.
      2. Business Logic Server sends a request for an OTP to an Authorization Server.
      3. Authorization Server will send a random OTP to the client by SMS.  
         **NOTE**: This way of authentication is called “Multi-factor Authentication”
      4. Then client then can call /login with Username and OTP on Business Logic Server.
      5. The business logic server validates the code with the username and the code.
      6. On successful authentication, the client gets a token, using which it can call secured end-points.



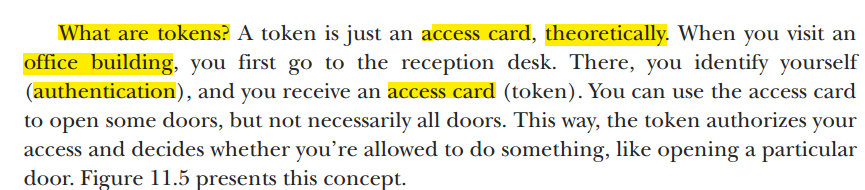
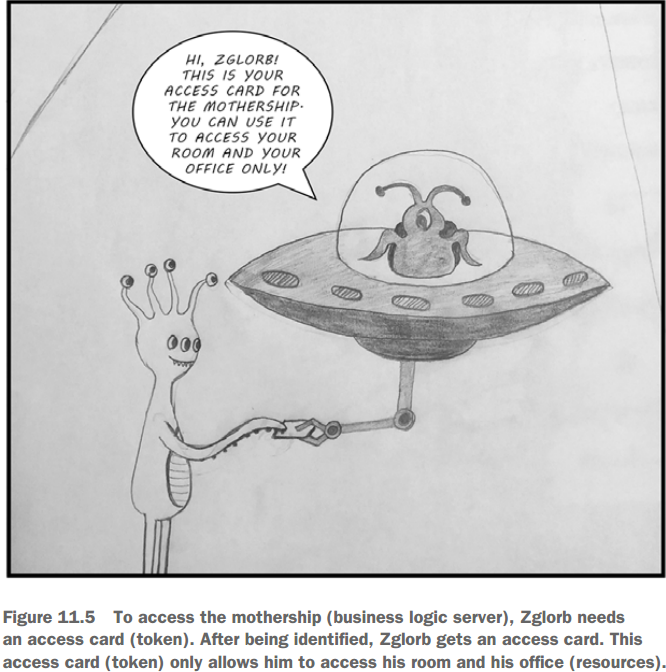
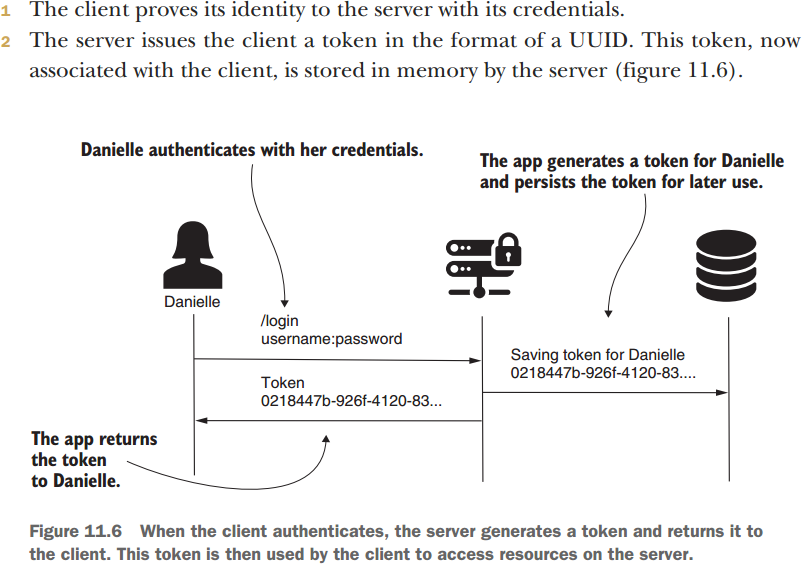
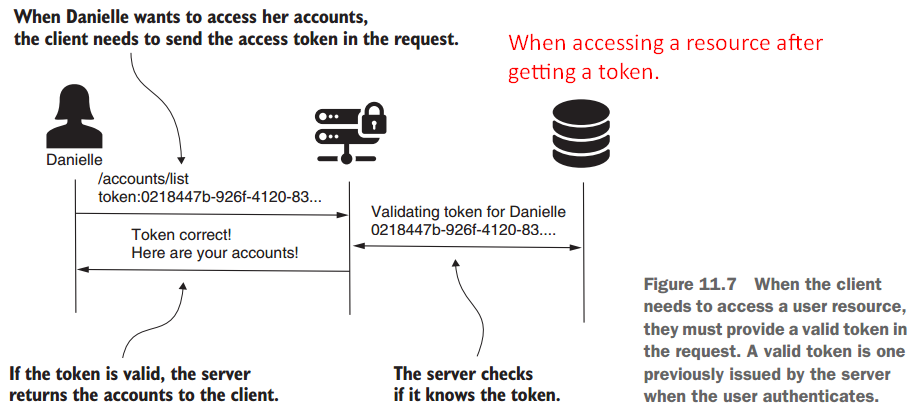
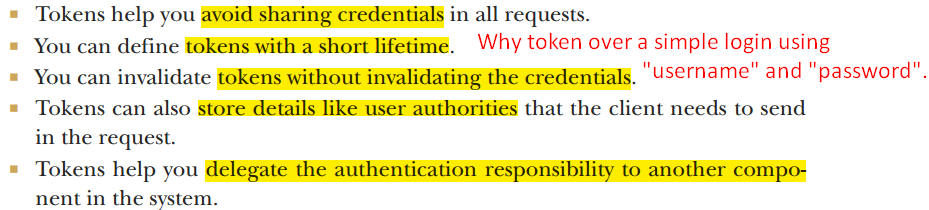
Some Arguments about this Architecture:

* + - 1. Client should share its credentials with Authentication Server not with Business Logic Server.  
         **Answer**: For simplification.
      2. MFA can be more easily implemented with 3rd party management system such as Okta.  
         **Answer**: To teach you how to define custom filters.

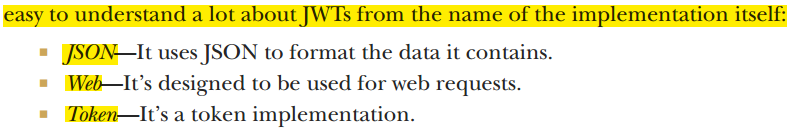
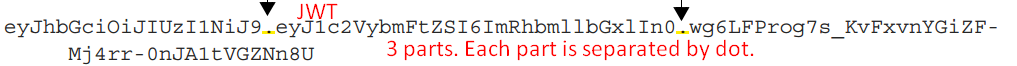
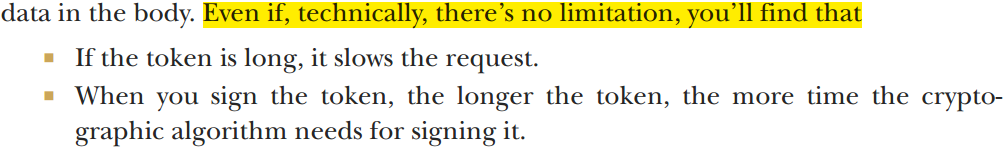
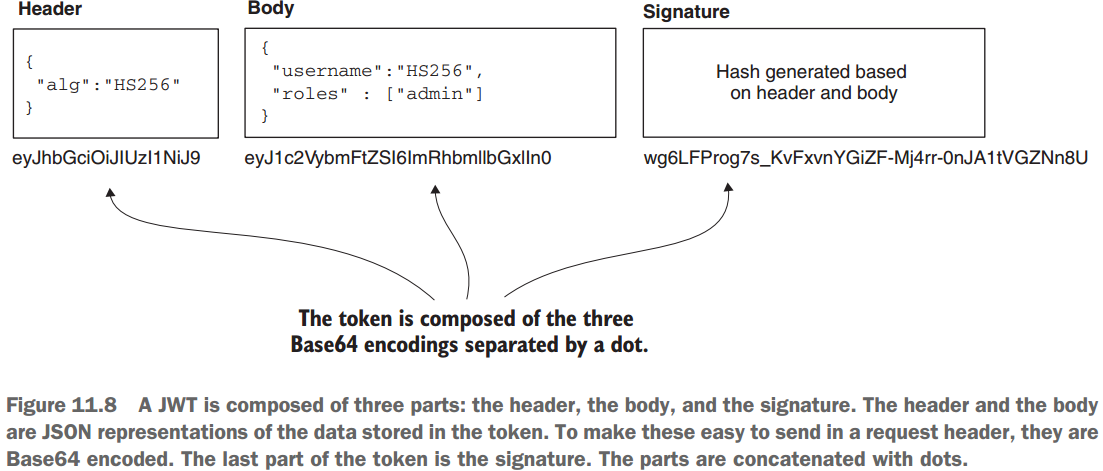
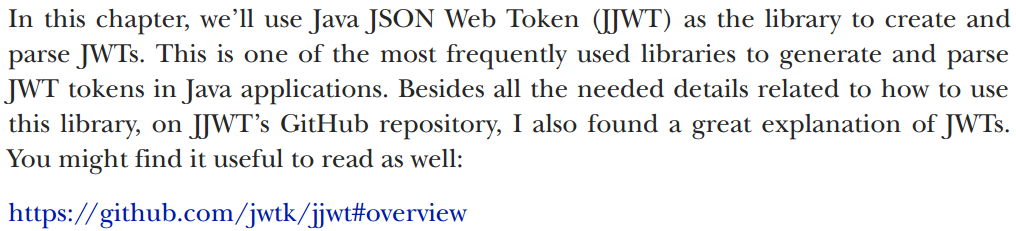
11.2 Implementing and Using Tokens

1. A token is similar to access card.
2. An application obtains a token as a result of the process of authentication and to access resources.
3. Endpoints represent the resources in a web application.
4. A token is string sent by HTTP header by client to access a particular endpoint=resource
5. Token may be a plain string like UUID or complex shape like a JWT.
6. Today, tokens are often used in authentication and authorization architectures. So understand.  
   Every common in Oauth Architecture.

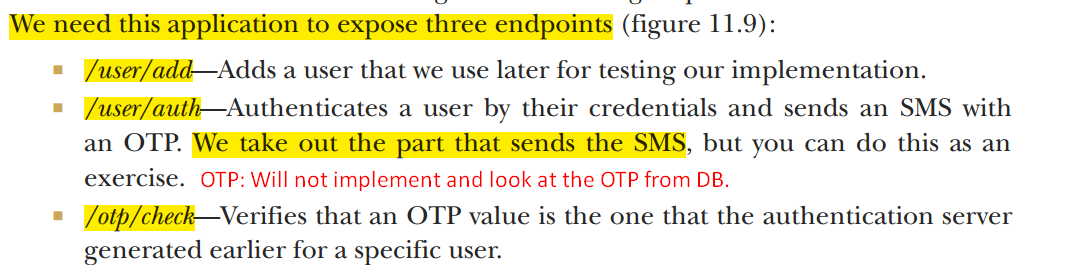
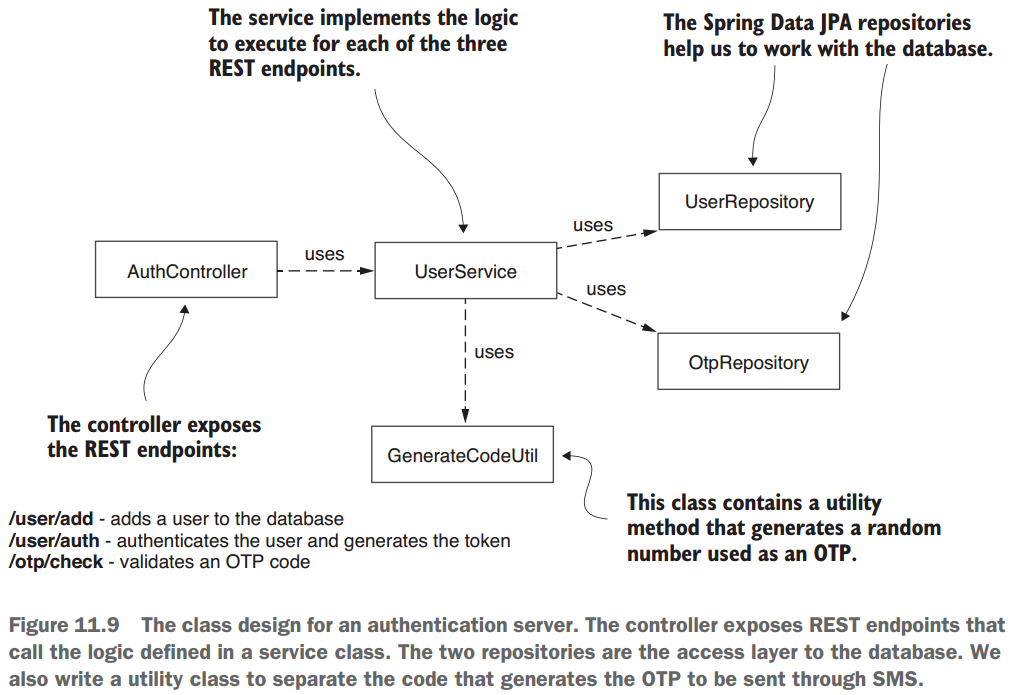
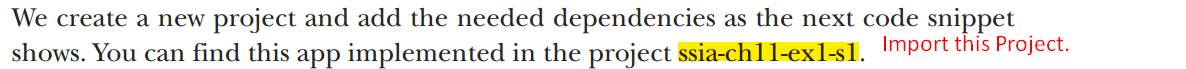
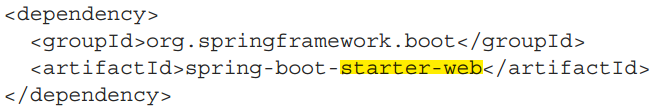
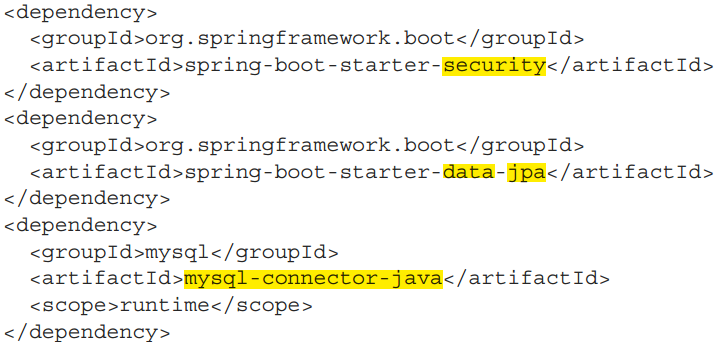
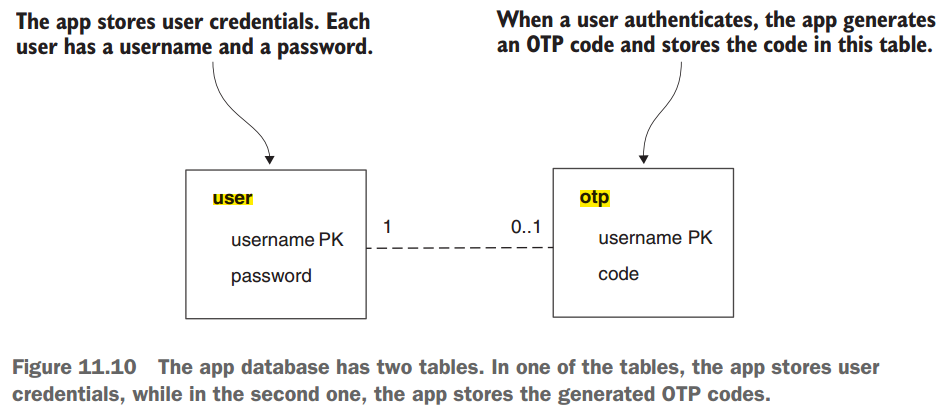
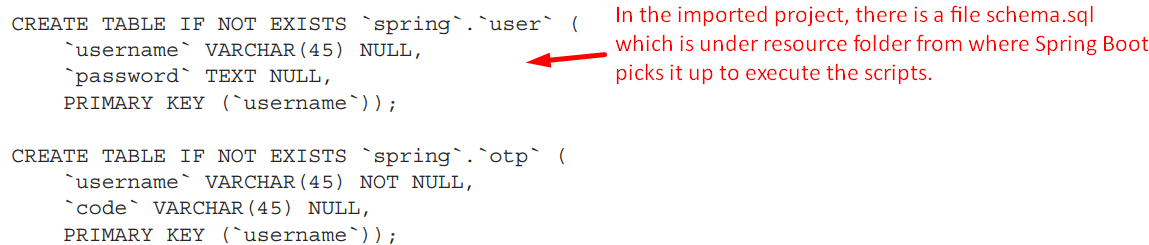
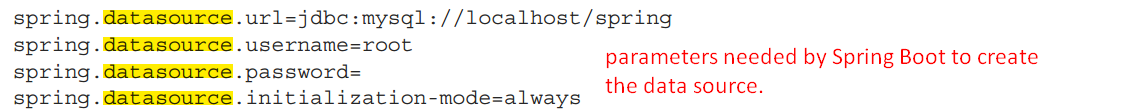
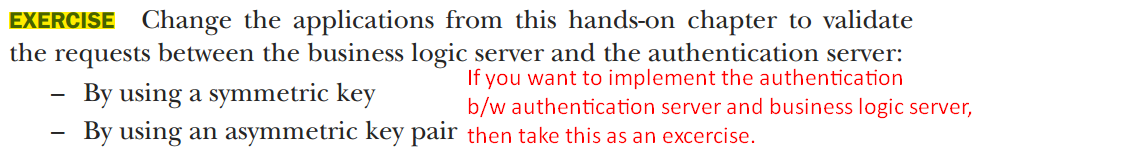
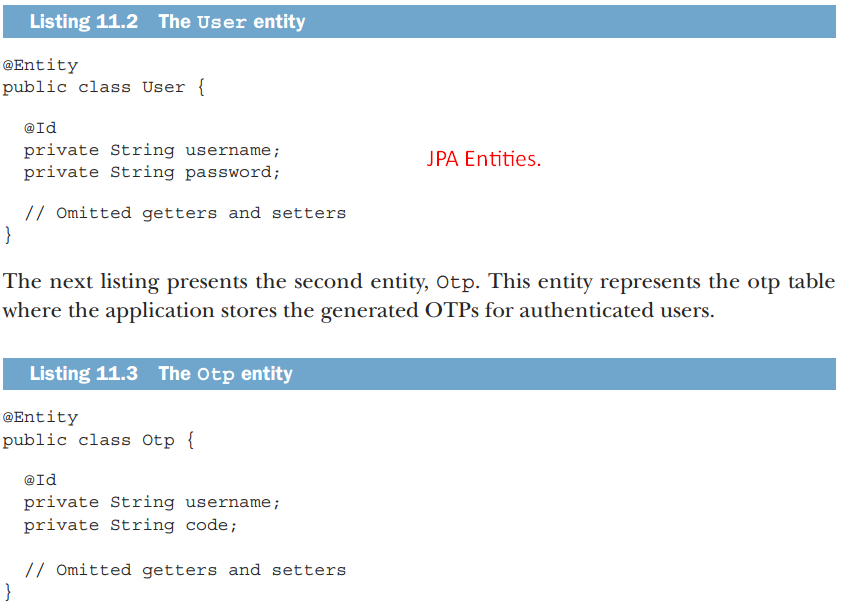
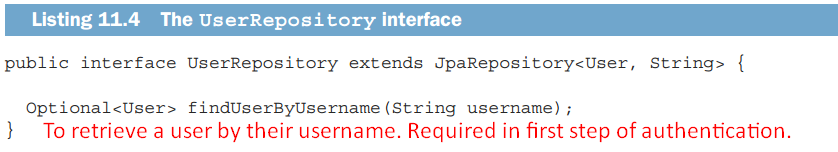
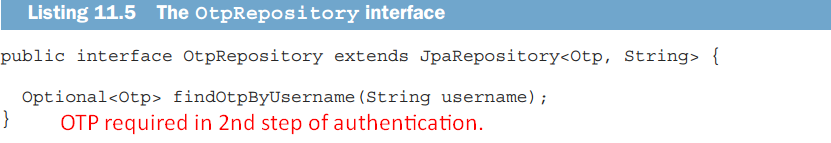
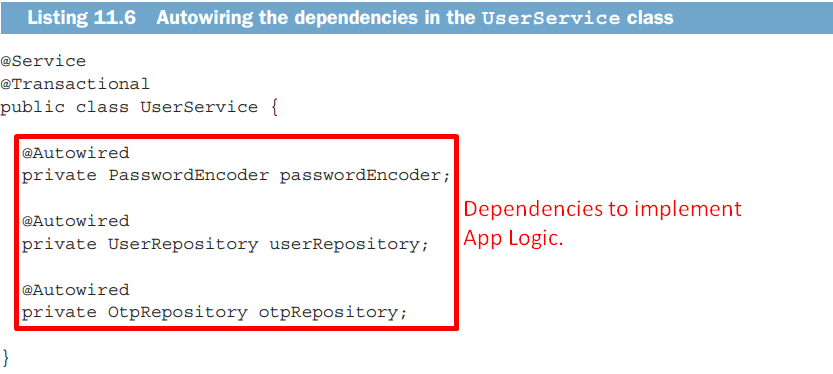
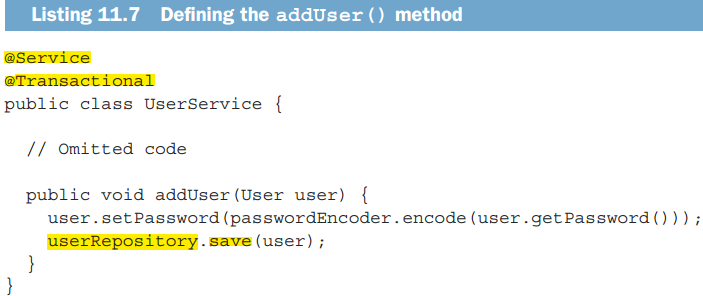
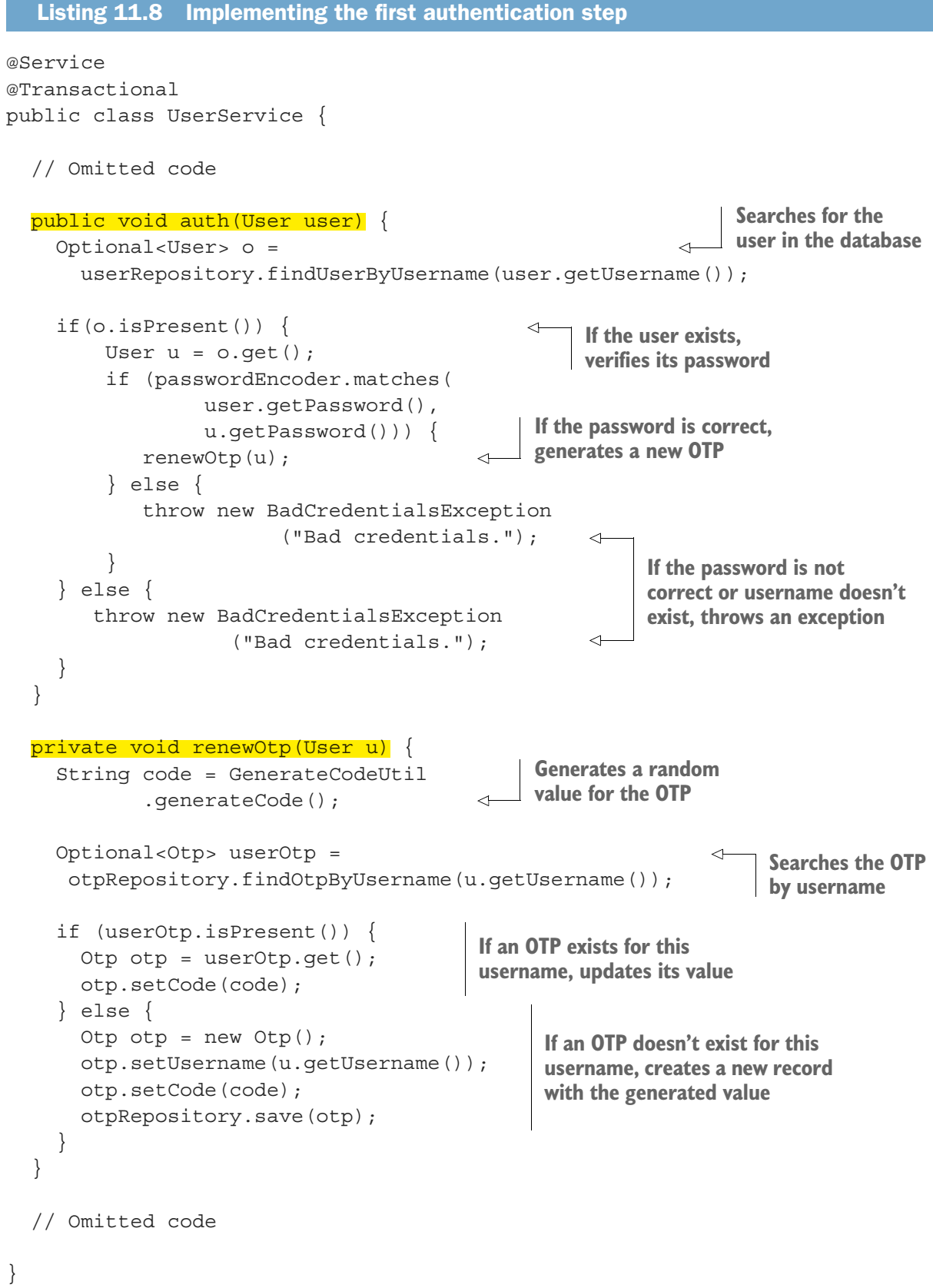
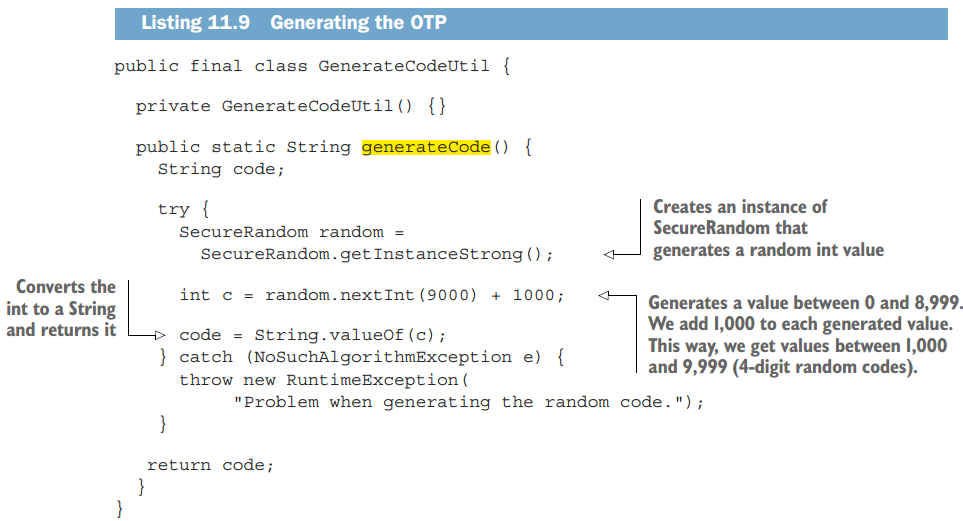
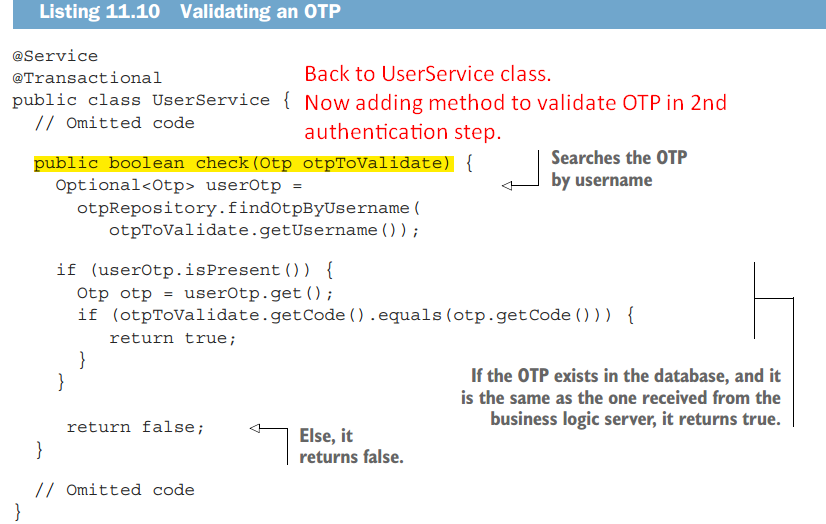
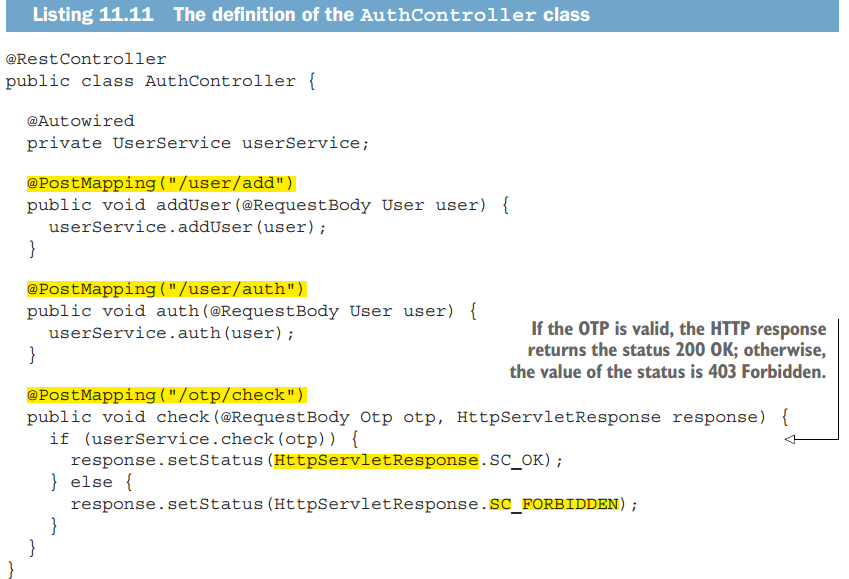
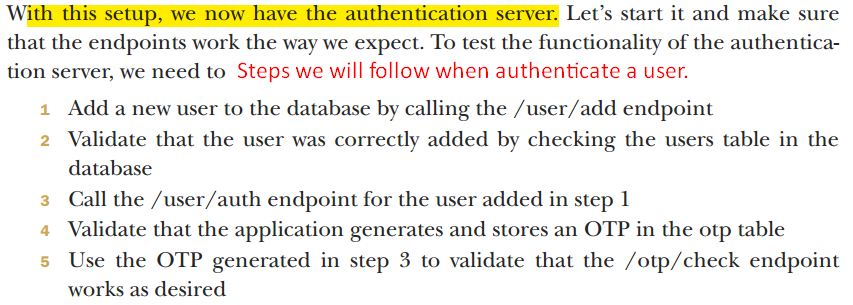
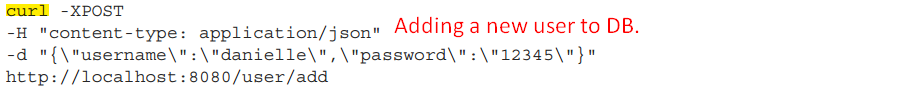
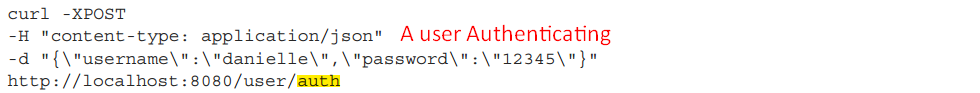
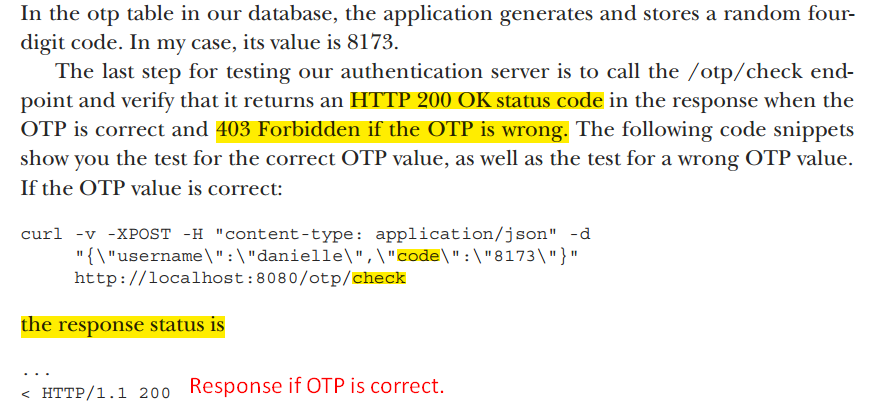
11.2.1 What is a token?

1. Tokens provide a method that an app uses to prove it has authenticated a user which allows the user to access the app’s resource.
2. 
3. 
4.  
5. 
6. Advantages of tokens  
   
   1. **Tokens can also store details needed in the request**:
      1. Storing authorities and roles. So, replacing server-side session into client-side session which offers us better flexibility for horizontal scaling.
   2. Tokens help you delegate the authentication responsibility to another component in the system.
      1. Your system doesn’t need to manage its own users.   
         Instead it allows users to authenticate using credentials from accounts they have on other platforms such as GitHub, Twitter.
      2. Even if we implement our own authentication system, we can make the implementation separate.   
         It helps us enhance scalability and making the system more natural to understand and develop.   
         Read more about it in this book:  
         https://livebook.manning.com/book/api-security-in-action/chapter-5/  
         https://livebook.manning.com/book/api-security-in-action/chapter-6/

11.2.2 What is JSON Web Token?

1. **Agenda**:
   1. More specific implementation of tokens - the JWT.
   2. This token implementation has benefits so quote common in today’s application.
2. Token can be anything like
   1. UUID
   2. Access Card
   3. Sticker you receive when you buy a ticket in a museum.
3. Let’s see what a token looks like and why a JWT is special.
4. 
5. 
6. **Header**:
   1. Formatted as JSON.
   2. Base64 Encoded.
   3. Metadata related to token.
      1. Name of the algo used to sign the token.
7. **Body**:
   1. Formatted as JSON.
   2. Base64 Encoded.
   3. Detail related to token
      1. The authorization details.
   4. Don’t include a lot of data in the body.
8. 
9. **Digital Signature**:
   1. Last part of the JWT.
   2. Optional.
   3. When you want to sign the header and body.
   4. Then this signature is used to validate the token.
10. Brief about JWT.
    1. Token Implementation.
    2. Easily transferring data during authentication.
    3. Easily signing data to validate its integrity.
11. More about JWT:  
    https://livebook.manning.com/book/microservices-security-in-action/chapter-7/  
    <https://livebook.manning.com/book/microservices-security-in-action/h-json-webtoken-jwt-/>
12. 
13.   
    <https://github.com/jwtk/jjwt#overview>

11.3 Implementing the Authentication Server

1. Agenda: Implementation.
2. The authentication server connects to a database where it stores the user credentials and the OTPs generated during request authentication events.
3. 
4. More about how to create Rest-EndPoints:
   1. <https://livebook.manning.com/book/spring-in-action-sixth-edition/chapter-6>
5. 
6. 
7. 
8. 
9. 
10. 
11. 
12. **Note**: Spring Security Dependency was added only for **BCryptPasswordEncoder** because we want to use to hash the users’ passwords when stored in the database.
13. **NOTE**: To keep the example short and relevant to our purpose, I don’t implement authentication b/w the business logic server and the authentication server.
14. 
15. Because we use Spring Data JPA, we need to write the **JPA entities** and then the **repositories**, and because we have two tables, we define two JPS entities and two repository interfaces.
16. 
17. 
18. 
19. 
20. 
21. 
22. 
23. 
24. 
25. 
26. 
27. 
28. More details on hash function in Chapter 02.  
    <http://mng.bz/oRmy>
29. 
30. Following the above end-point hit successfully, OTP would be generated and stored inside the OTP table.
31. 
32. 

11.4 Implementing the Business Logic

1. **Agenda**:
   1. You will learn to implement and use JWTs for authentication and authorization.