

もっと実践!

サーバーサイド Kotlin



もっと実践!サーバーサイド Kotlin

FORTE 著

2020-03-01 版 aozora Project 発行

はじめに

この本を手にとっていただきどうもありがとうございます。著者の FORTE (フォルテ) です。前著である「入門!実践!サーバーサイド Kotlin」に引き続き、今回もサーバーサイド Kotlin 本を書きました。本書も前著と同じく Windows、Mac 両対応です。IDE については特に影響する箇所はないはずなので、前著を読んでいれば問題ないはずです。もし本著の内容で分からない点があれば前著を見るか、ぜひ著者^{*1}までご一報ください。ベストエフォートで回答させていただきます。

本著の内容

今回はもっと実践!ということで、コンシューマー向けの Web サービスでは必ずと言ってよいほど使われるユーザー登録、認証とアプリケーションを公開するの 2 点に関して実践してみた結果を解説しました。

ユーザー登録と認証はその性質上、ユーザーの個人情報を預かる可能性や外部に公開したくない情報を預かる可能性が高くなります。そのためセキュリティや脆弱性に一層気を使う必要がある部分となります。今回は本番運用できるレベルと自信を持っていえるわけではありませんが、実装例と動作するコードを公開しています。ゆくゆくは私が個人サービスとして公開、運用した知見を持って本番運用できるレベルのコード、解説をしていきたいと思っています。ですが、今回はあくまで基礎レベルの解説となります。

アプリケーションを公開するというのには一般にデプロイと呼ばれている作業になります。今回はデプロイ先として heroku を用いました。heroku (ヘロク) はサンプルや小規模なアプリを公開する分には非常に便利なサービスです。今回は heroku を用いることで Spring Boot アプリケーションを動作させるまでの苦労がほぼなくなったといえるくらい簡単にできました。今回はデプロイの手順を Windows、Mac の両方で解説し、アプリケーションをリリースするというもっともモチベーションが上がる行為を解説していきます。

^{*1} Twiter : <https://twitter.com/FORTEGp05>

どんな人向けか

本著は前著「入門!実践!サーバーサイド Kotlin」を読んでいる前提としています。具体的には Kotlin + Spring Boot + JPA + Thymeleaf + データベース (h2 Database など) でデータ作成、検索、更新、削除 (いわゆる CRUD 処理) が分かる、やったことがある、開発環境などもすでにある、あるいは自分で揃えられる程度の方を対象読者としています。

そのため、サーバーサイド Kotlin とは? Spring Boot とは? えっそもそも Kotlin ってなに? 読み方もわからない… という人はぜひ前著である「入門!実践!サーバーサイド Kotlin」をお読みになることをお勧めします。次の QR コード、リンクから購入可能です! (PR)



▲図 1 前著の販売ページ

<https://fortegp05.booth.pm/items/1560389>

この本で得られること

この本は Java など Web 開発の経験がある人向けにサーバーサイド Kotlin でユーザー認証と認可、デプロイを実践してみる本です。この本を読み終わると次のような状態になります。

- Spring Security による認証の使い方が分かる
 - ユーザー登録
 - ユーザー認証

-
- デフォルトユーザーの作成
 - 認証関係のテストの書き方
 - heroku へのデプロイ
 - heroku のユーザー登録
 - データベース接続情報のマスクの仕方

あなたの Kotlin で Web アプリケーションを作りたい、個人サービスを作ってみたいという思いに答えられたらこんなにうれしいことはありません。

この本では解説しないこと

本著では Spring Security や heroku の解説をしています、あくまで使い方のみであり詳細な仕組みや中身のソースコードについては解説していません。

Spring Security については完璧なセキュリティを保証するものではありません。あくまでサンプルソースであり、本番環境での動作実績やなにかの脆弱性診断をパスしているものではありません。

また解説には JPA や Thymeleaf も使用していますが、本著では解説しません。よくわからない!という方はぜひ前著をご覧ください。ことをお勧めします。

この本の使い方

この本は筆者がサーバーサイド Kotlin によるユーザー登録、認証やデプロイを学ぶ中で疑問に思ったことや調べたことを技術書の形でアウトプットしたものです。そのため、ユーザー認証や登録について知りたければ第 1 章「Spring Security による認証とユーザー登録」のページからご覧ください。またとりあえずデプロイをしてみたい!ということであれば第 2 章「デプロイ」からご覧ください。もちろん最初から全部読んでいただいても大丈夫です。

本書はこの本のとおりによれば動くものができる、というところを目指して書かれています。この本を読めば理屈や仕組みがすべて理解できるようには書かれていません。この本を入り口としてそのさらに奥にあることに興味を持っていただけたら幸いです。

特にユーザー認証、認可についてはこれを利用することでさまざまなアプリケーション開発が可能になると思います。ぜひ、これを利用して自分用のツールを作ったりしてみてください。

またこの本のコラムは勉強記録として私が感じたポイントやハマったポイントなどを解説しています。たとえばこまめにビルドすると効率がいいよとか、前著からどれくらい成長できているか?みたいな話をしています。Kotlin と直接関係ない話をしているかもしれませんが、よかったら見てみてください。

読み終わった感想や間違いの指摘、追加要望などは次のハッシュタグを用いて Twitter で呟いていただけると嬉しいです。ぜひ、感想をお待ちしております。

#もっと実践サーバーサイドKotlin

免責事項

本書に記載する内容は筆者の所属する組織の公式見解ではありません。また、本書は可能な限り正確を期すように努めていますが、筆者がその内容を保証するものではありません。そのため、本書の記載内容に基づいた読者の行為、及び読者が被った損害について筆者はなんら責任を負うものではありません。

目次

はじめに	2
本著の内容	2
どんな人向けか	3
この本で得られること	3
この本では解説しないこと	4
この本の使い方	4
免責事項	5
 第 1 章 Spring Security による認証とユーザー登録	 8
1.1 Spring Security とは	8
1.2 開発環境について	8
1.2.1 バージョン一覧（執筆時）	9
1.3 Spring Security でお手軽認証	9
1.4 実用的な認証として管理者画面を実装する	11
1.4.1 認証情報として任意のユーザー名とパスワードを設定する	11
1.4.2 データベースを用いてユーザー情報を管理する	15
1.5 要認証ページとして管理者画面を実装する	23
1.5.1 認証対象ページの設定	23
1.5.2 管理者画面に記事一覧を表示する	25
1.5.3 管理者画面で記事の単数削除	33
1.5.4 管理者画面で記事の複数削除	36
1.6 ユーザー登録を実装する	41
1.6.1 ユーザー登録画面への遷移を実装する	41
1.6.2 ユーザー登録処理の実装	49
1.6.3 ユーザーログインの動作確認	57
1.7 登録したユーザー情報で記事投稿を制御する	65
 第 2 章 デプロイ	 75
2.1 heroku によるデプロイ	75
2.2 デプロイのリスクと対策	75
2.2.1 リスク	75

2.2.2	対策	76
2.3	デプロイの準備	76
2.3.1	認証情報の環境変数化	76
2.3.2	heroku へのユーザー登録	77
2.3.3	heroku cli を Windwos にインストールする	79
2.3.4	heroku cli を Mac にインストールする	80
2.3.5	heroku cli でログインする	81
2.3.6	git のインストール	81
2.3.7	Github の準備	82
2.3.8	psql をインストールする	82
2.3.9	psql を Windows にインストールする	82
2.3.10	psql を Mac にインストールする	88
2.4	heroku にデプロイする	88
2.4.1	heroku アプリを作成する	89
2.4.2	heroku にデータベースとして PostgreSQL をセットする	89
2.4.3	heroku にアプリをプッシュする	89
2.4.4	管理者ユーザーを登録する	91
2.4.5	動作確認	94
2.4.6	非公開にする	94
あとがき		96
	コードを書くのが大変	96
	サーバーサイド Kotlin を広めていく	97
	最後になりましたが…	97
電子版について		98
著者紹介		99
	文章	99
	表紙イラスト担当	99

第 1 章

Spring Security による認証とユーザー登録

1.1 Spring Security とは

Spring Security は認証およびアクセス制御フレームワークです。名前のとおり Spring フレームワークの一部であり、Spring はもちろん、Java アプリケーションに認証と認可の両方を提供することができます。そして強力であり、高度にカスタマイズ可能となっています。

ここでいう認証とはアクセスしてきたユーザーを識別し本当に本人であることを確認することです。具体的にはパスワードによる確認になります。そのため、パスワードは当人しか知らないものであるという前提となっています。なお、ユーザーの ID は認証ではなくユーザーを識別するためのものになります。

次に認可とは、そのユーザーの属性に応じてアクセスできる範囲を確認することです。たとえば管理者画面には管理者しかアクセスできない、ユーザー画面にはそのユーザーしかアクセスできないなどが認可となります。本著では Role という属性でそのユーザーアクセスできる範囲を制御しています。

公式ページ（英語）は次になります。

<https://spring.io/projects/spring-security>

1.2 開発環境について

本著は前著「入門!実践!サーバーサイド Kotlin」に引き継いでいますので、そのプロジェクトを引き継いで解説していきます。もし本著から購入された場合は次の URL より前著の最後の状態のプロジェクトをダウンロードしてください。

https://github.com/fortegp05/server_side_kotlin_bbs_sample/releases/tag/ssk1

もちろん前著をご購入頂いて 1 から始めていただいても大丈夫です（PR）

<https://fortegp05.booth.pm/items/1560389>

なお、本書でご紹介しているソースコードの完成品は次の Github リポジトリにアップロード済みです。

https://github.com/fortegp05/server_side_kotlin_bbs_sample

また、同じ内容を Java で表現したソースコードもご用意してあります。よかったら併せてご利用ください。

https://github.com/fortegp05/java_bbs_sample

1.2.1 バージョン一覧（執筆時）

Windows

- Windows : Windows 10 Version 1903 (OS Build 18362. 592)
- IntelliJ IDEA (Community) : 2019.2.1 192.6262.58
- VS Code : Version 1.38.0

MacOS

- macOS : macOS Catalina 10.15.1
- IntelliJ IDEA (Community) : 2019.1.3 191.7479.19
- VS Code : Version 1.41.1

共通のもの

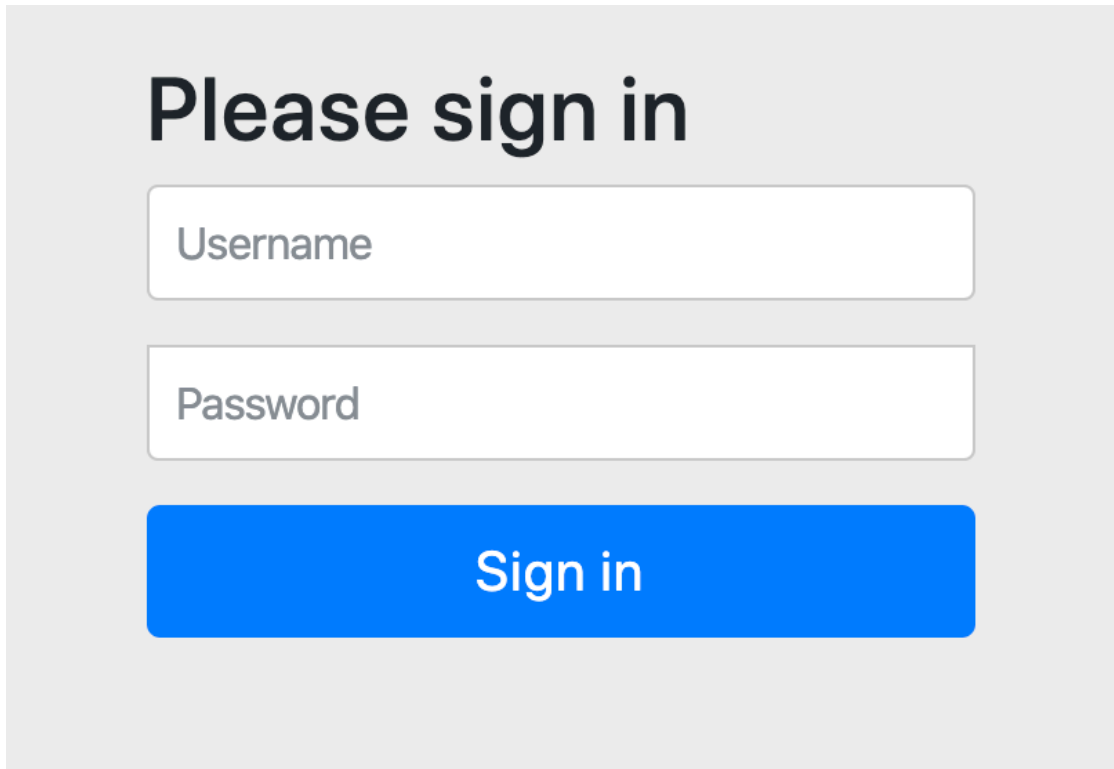
- Spring Boot 2.1.8
- Java : 12.0.2

1.3 Spring Security でお手軽認証

それでは Spring Security を用いて簡単にユーザー認証をしてみます。拍子抜けするほど簡単にできますので、一緒にやってみましょう。まずは spring-boot-starter-security を build.gradle.kts に追加します。

```
implementation("org.springframework.boot:spring-boot-starter-security")
```

BootRun して「http://localhost:8080」にアクセスしてみます。すると、次のような認証画面が出ます。



Please sign in

Username

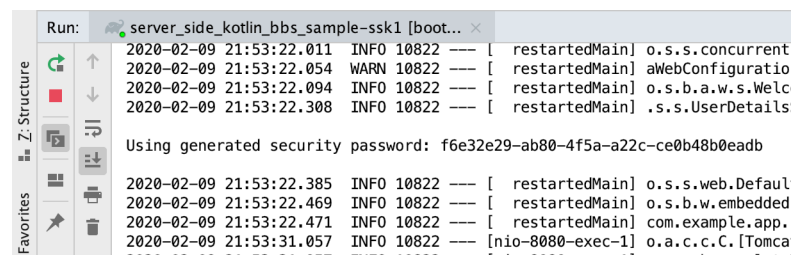
Password

Sign in

▲図 1.1 認証画面

このユーザーの Username は user、パスワードは次の形式で起動時のログに出ます。

Using generated security password: f6e32e29-ab80-4f5a-a22c-ce0b48b0eadb



```
Run: server_side_kotlin_bbs_sample-ssk1 [boot... x
2020-02-09 21:53:22.011 INFO 10822 --- [ restartedMain] o.s.s.concurrent
2020-02-09 21:53:22.054 WARN 10822 --- [ restartedMain] aWebConfiguratio
2020-02-09 21:53:22.094 INFO 10822 --- [ restartedMain] o.s.b.a.w.s.Welc
2020-02-09 21:53:22.308 INFO 10822 --- [ restartedMain] .s.s.UserDetails

Using generated security password: f6e32e29-ab80-4f5a-a22c-ce0b48b0eadb

2020-02-09 21:53:22.385 INFO 10822 --- [ restartedMain] o.s.s.web.Default
2020-02-09 21:53:22.469 INFO 10822 --- [ restartedMain] o.s.b.w.embedded
2020-02-09 21:53:22.471 INFO 10822 --- [ restartedMain] com.example.app.
2020-02-09 21:53:31.057 INFO 10822 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat
```

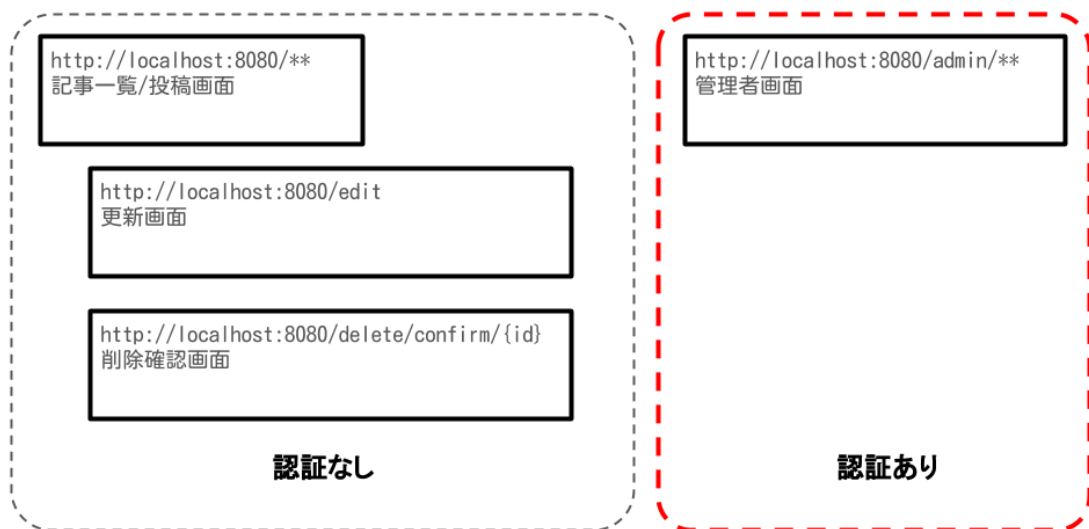
▲図 1.2 認証情報

この情報でログインすると掲示板画面が表示されます。

1.4 実用的な認証として管理者画面を実装する

これで Spring Security による認証を経験できました。しかし、このままではサーバーを起動するたびに毎回パスワードが変わってしまいますし、いちいちログからパスワードを拾ってこななければなりません。ユーザー名も user 固定ですし、パスワードは長すぎて覚えられないでしょう。もっと自分で扱いやすいパスワードにしたいと思っています。

つまりまったく実用的ではないわけです。そこで Spring Security の設定を変更して実用的な仕組みにしてみましょう。認証のイメージとしては管理者しかアクセスできない管理者画面を作成し、そのページに対する認証と認可を設定してみます。



▲図 1.3 管理者画面のイメージ

1.4.1 認証情報として任意のユーザー名とパスワードを設定する

まずは認証情報として任意のユーザー名とパスワードの設定を行っていきます。まずは「/src/main/kotlin/com/example/app/bbs/」の下に「config」というパッケージを作成し、新規ファイル「BbsAdminWebSecurityConfig.kt」を作成します。中身は次のとおりです。なお、import 文は紙面の都合で途中で改行しています。実装時は改行しないようしてください。

▼リスト 1.1 管理者画面認証情報設定

```

1: package com.example.app.bbs.config
2:
3: import org.springframework.beans.factory.annotation.Autowired
4: import org.springframework.context.annotation.Bean
5: import org.springframework.context.annotation.Configuration
6: import org.springframework.security.config.annotation.authentication.
7: builders.AuthenticationManagerBuilder
8: import org.springframework.security.config.annotation.web.configuration.
9: EnableWebSecurity
10: import org.springframework.security.config.annotation.web.configuration.
11: WebSecurityConfigurerAdapter
12: import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder
13: import org.springframework.security.crypto.password.PasswordEncoder
14:
15:
16: @Configuration
17: @EnableWebSecurity
18: class BbsAdminWebSecurityConfig : WebSecurityConfigurerAdapter() {
19:
20:     @Autowired
21:     lateinit var passwordEncoder: PasswordEncoder
22:
23:     @Bean
24:     fun passwordEncoder(): PasswordEncoder {
25:         return BCryptPasswordEncoder()
26:     }
27:
28:     @Override
29:     override fun configure(auth: AuthenticationManagerBuilder) {
30:
31:         auth.inMemoryAuthentication()
32:             .withUser("admin")
33:             // 次は解説用に平文で記載しているので実際にやってはダメ！
34:             .password(passwordEncoder.encode("root"))
35:             .authorities("ROLE_ADMIN")
36:     }
37: }

```

ここで新しく登場した構文について解説していきます。

継承とクラス名のあとの括弧

今回は「BbsAdminWebSecurityConfig : WebSecurityConfigurerAdapter()」となっていますが、この継承元についている括弧はコンストラクタを呼び出す、という指定になります。Kotlin では Java と違って明示的にコンストラクタを定義して継承元のコンストラクタを呼び出すような書き方をしなくとも、継承元の宣言時にコンストラクタと一緒に呼び出すことが可能となっています。

override 句

メソッド宣言の前にある「override」句は明示的にオーバーライドすることを宣言します。Kotlin はデフォルトでメソッドのオーバーライドは禁止されています。そのため、オーバーライドしたいときは明示的にそれを宣言する必要があります。そのための識別子が「override」句になります。

なお、同じような働きをするもので `@Override` というアノテーションがありますが、こちらは Kotlin の言語仕様とは無関係なので、`@Override` だけ付けても Kotlin のコンパイルがとおりません。また両方付ける意味はないので `override` 句を付けたらアノテーションはなくてよいでしょう。

PasswordEncoder

`PasswordEncoder` はパスワードを安全に処理するためにハッシュ化する仕組みとなっています。システムで用いられるパスワードは必ずハッシュ化するなどして、システム上に平文で保存することは止めましょう。万が一データベース情報やファイル、ログ情報などが流出した場合、パスワードも流出してしまいます。そして流出してしまったユーザーが同じパスワードを使い回していたら、他のシステムも不正アクセスの対象となってしまいます。

ハッシュ化したパスワードは、元の文字列に戻すこと（復元）が非常に大変です。そのため、流出したことがわかれば復元されるまえにパスワードを変えたり、アクセス遮断することができます。ハッシュ化すれば完璧というわけではありませんが、二重三重に対策をすることで被害を小さくすることができます。

今回はハッシュ化の仕組みに `BCryptPasswordEncoder` を使用していますが、他にもいくつかあります。ここでは `BCryptPasswordEncoder` で間に合うため解説しませんが、詳しくは公式の JavaDoc に他のクラスが乗っています。

<https://docs.spring.io/spring-security/site/docs/current/api/org.springframework.security.crypto.password.PasswordEncoder.html>

なお、解説用にソースに平文で書いていますが、**運用時や Github などにソースを push ときは絶対やってはダメ**です。実際に Github から認証情報が漏洩する事例などが報告されています。セキュリティの事故はハッキングなどによる外部起因のものよりも、内部起因のものが多いと報告されています^{*1}。これは内部不正のような意図的な行動も含みますが、意図していないミスなども含みます。セキュリティ対策は何もなければ笑い話ですみませんが、何かあってからでは遅いものになります。普段から高い意識を持っておくといざというときに痛い思いをしなくて済む可能性が高いです。

^{*1} 情報セキュリティインシデントに関する調査報告書
<https://www.jnsa.org/result/incident/2018.html>

[「システム上にパスワードは平文で保存するな」みたいなことはどこで学べる?]

この文章を書いている疑問に思ったのですが、「システム上にパスワードは平文で保存するな」というようなことをどこで学んだのでしょうか?私は記憶にはないですが、恐らくネット上でそういった記述を見たのだと思います。少なくとも専門学校でこういった内容を学んだ覚えはありません。一応 IPA には安全なウェブサイトの作り方というページがありそこでは学べるようです。^{*2}

ですが、日常的に IPA のサイトなんて見ないですし、会社でも見た覚えはないのでここで学ぶというのはなかなか難しいですね。

今の所は本著のような誰かのアウトプットにふれる、自分でアウトプットしようとして調べてみるあたりが学べる可能性としては高そうです。座して待っているは何も得られない、ということなのかもしれません。

inMemoryAuthentication

このメソッドは認証をメモリで行うことを示します。メモリですので、サーバーを停止すれば消えてしまいます。次に起動したときに認証情報を保持するために、withUser メソッドや password メソッドで認証情報を設定しています。

解説が長くなりましたが、平文で載せてしまっているパスワードをハッシュ化するものに置き換える作業します。単純な話、平文のパスワードである「root」をハッシュ化したものを実装すればいいだけなので、一度実際にハッシュ化したものをログに出力してそれを取り出して実装します。

次を「BbsAdminWebSecurityConfig.configure」に記載して BootRun してください。

```
val password: String = passwordEncoder.encode("root")
System.out.println(password)
```

するとログにハッシュ化されたパスワードがログに表示されるので、コピーしてパスワードとして指定しましょう。このとき、忘れずに System.out.println は削除しておきます。

▼リスト 1.2 ハッシュ化したパスワードを指定する

^{*2} IPA の安全なウェブサイトの作り方 <https://www.ipa.go.jp/security/vuln/websecurity.html>

```
1: // 変更後
2: auth.inMemoryAuthentication()
3:   .withUser("admin")
4:   .password(
5:     "$2a$10$CPNJ.P1WH8k1aMhC6ytjIuwXyULWKMxTP3H6h.LRnpumtccpvXEGy"
6:   )
7:   .authorities("ROLE_ADMIN")
8:
9: // 次は忘れずに削除しておく
10: val password: String = passwordEncoder.encode("root")
11: System.out.println(password)
```

BootRun しながら、Username は admin、パスワードは root でログインできれば確認 OK です。これでユーザー名とパスワードを任意のものに置き換えることができました。

[定期的にビルド (BootRun) しよう。]

一気に大量に実装して BootRun するとエラーが起きたときの切り分けが面倒になります。エラーログを追いかけて、大量の変更点を一個ずつ見ていくのは面倒ですよ。そして、面倒になるとモチベーションが下がって進めづらくなってしまいます。

そこで、慣れないコードを書いていくときは少しずつ書いてはビルドするとストレスを下げつつ前に進んでいけるとおもいます。たとえば 1 メソッド、1 ファイル実装したらビルドするように、リズムを作っていくとやりやすいでしょう。

1.4.2 データベースを用いてユーザー情報を管理する

任意のユーザーとパスワードをもつ管理者ユーザーを作成できましたが、もし管理者ユーザーを複数増やしたり、名前やパスワードを変えたいときにいちいちソースコードをいじるのは不便ですよ。管理者ユーザーなのでそんなに変更があるわけではないでしょうが、将来的に一般ユーザーも実装するかもしれないことを考えると一般ユーザーと同様にデータベースの情報で認証できたほうが便利そうです。

というわけで、さきほど作成した管理者ユーザーの認証情報をデータベースの情報に変更してみましょう。まずは Entity を作成していきます。

▼リスト 1.3 User の Entity 作成