

# **Spring Boot Seminar - 0**

**Wafflestudio Rookies seminar 2024**

**Sangmin Kim, 2024**

# 세미나 목표

- Spring Boot + Kotlin + JPA + MySQL 로 기본적인 API 서버를 만들고 배포할 수 있다.
- 일반적인 API 서버에서 필요로 하는 기능들을 구현할 수 있다. (Authentication, Search, Batch Processing, Image Upload 등)
- 앱 서버를 빌드하는 방법 (gradle) 과 컨테이너 기반의 배포 (docker, k8s) 에 대해 이해한다.

# Seminar - 0 목차

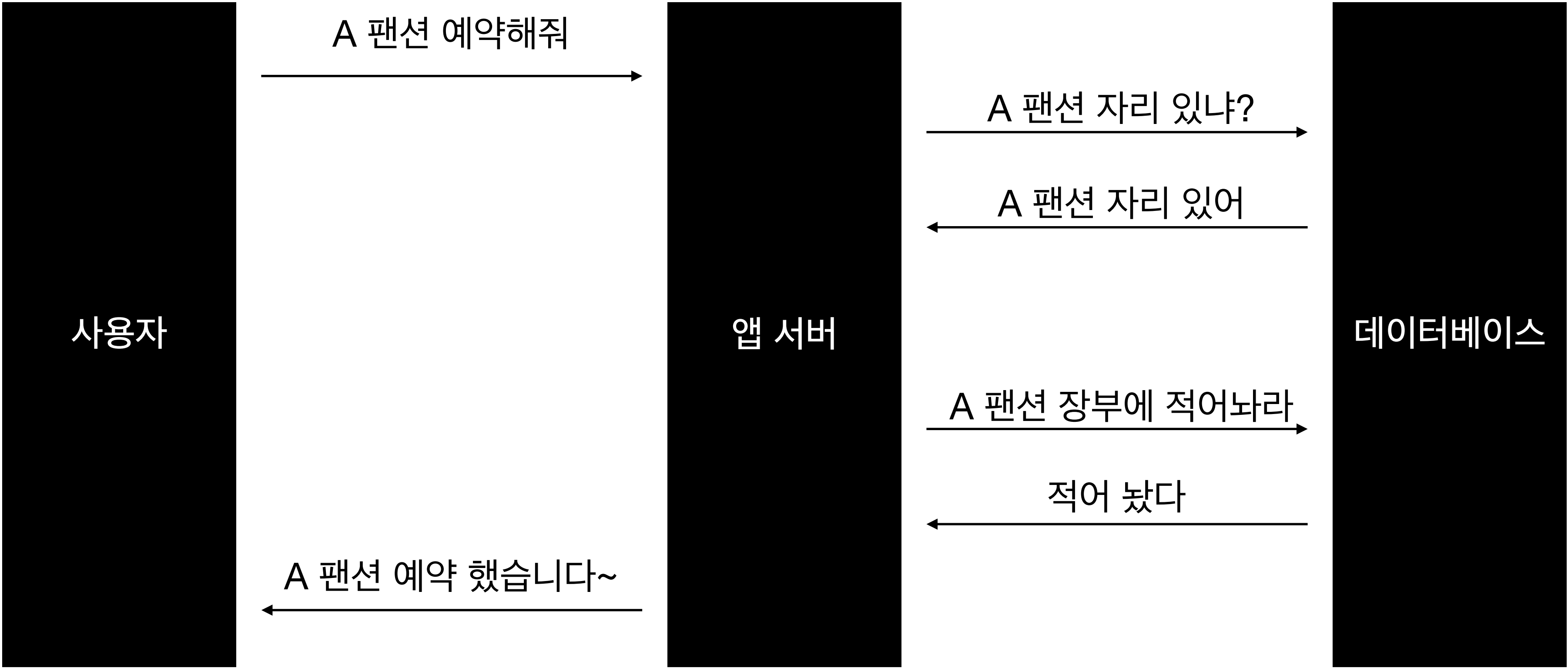
- Orientation
- 백엔드 서버의 데이터 흐름
- 앱 서버 뜯어보기
- Spring Boot Server 띄워보기

# 백엔드 서버의 데이터 흐름



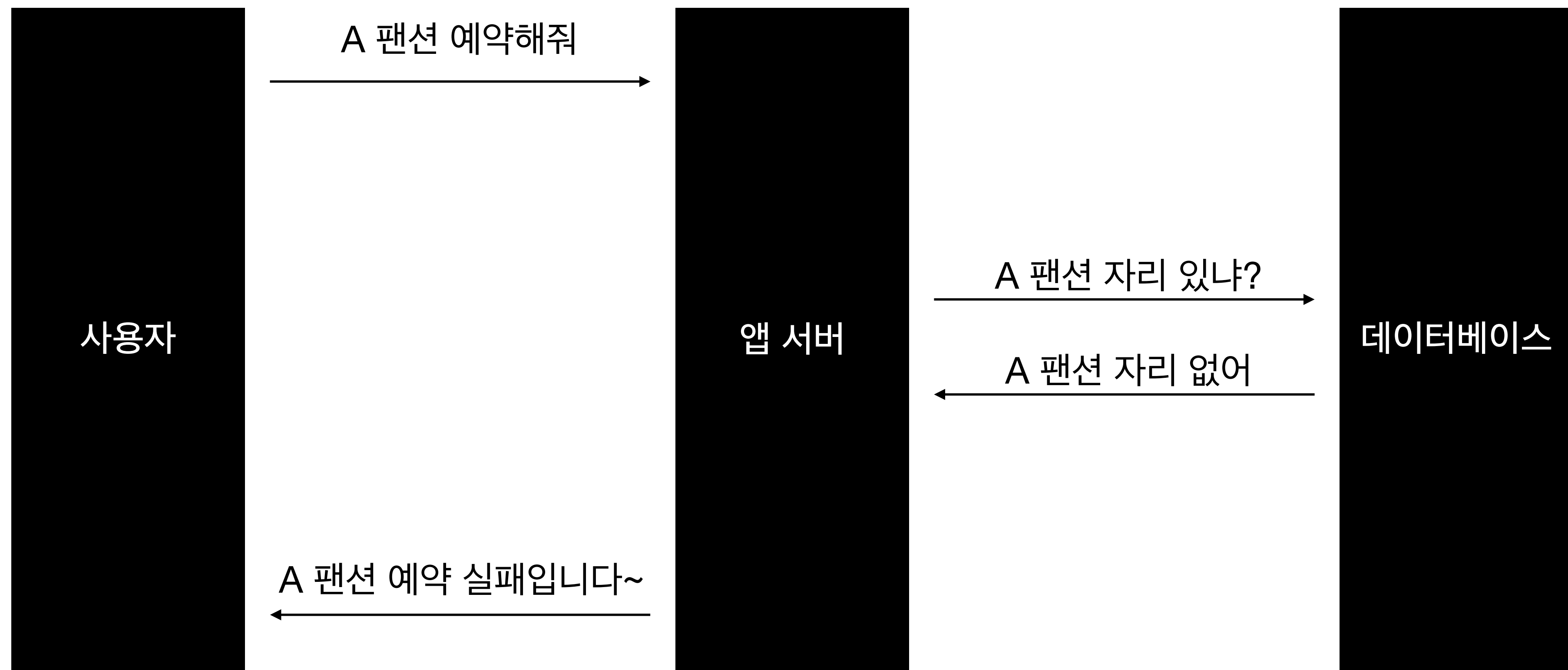
# 백엔드 서버의 데이터 흐름

## 팬션 예약 시나리오



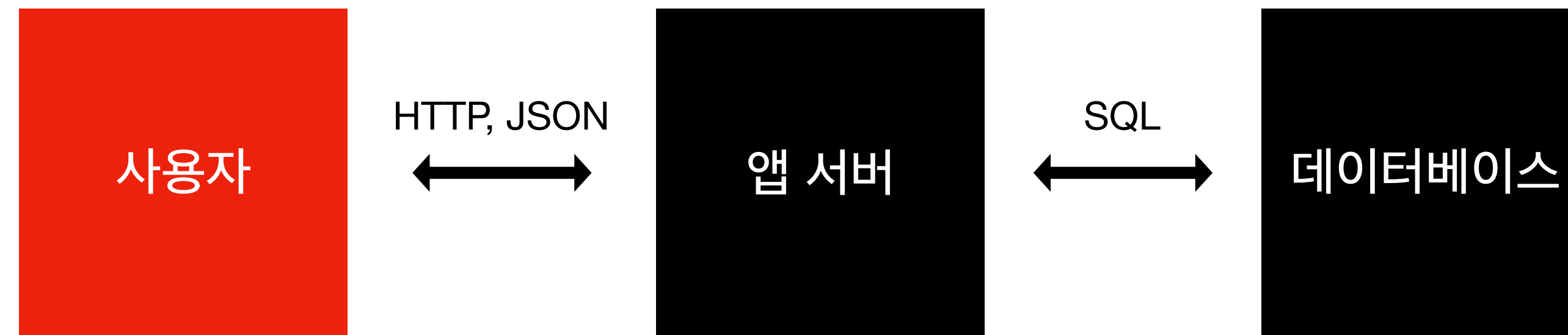
# 백엔드 서버의 데이터 흐름

## 팬션 예약 시나리오



# 백엔드 서버의 데이터 흐름

## 사용자



- 웹 브라우저
- 모바일 앱 (iOS, Android)
- 다른 앱 서버가 사용자가 될 수도..

# 백엔드 서버의 데이터 흐름

## 앱 서버

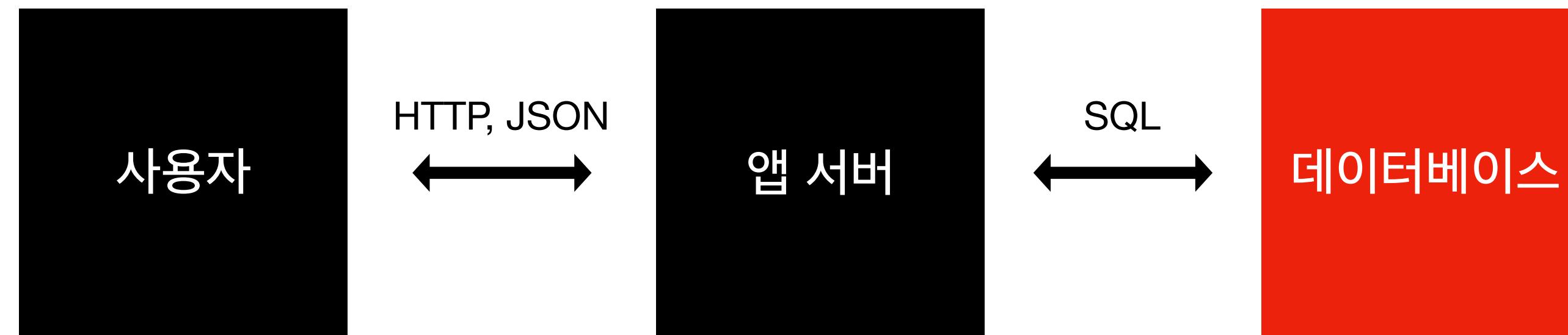


- Spring Boot Server
- FastAPI Server
- .NET Server
- ...



# 백엔드 서버의 데이터 흐름

## 데이터베이스

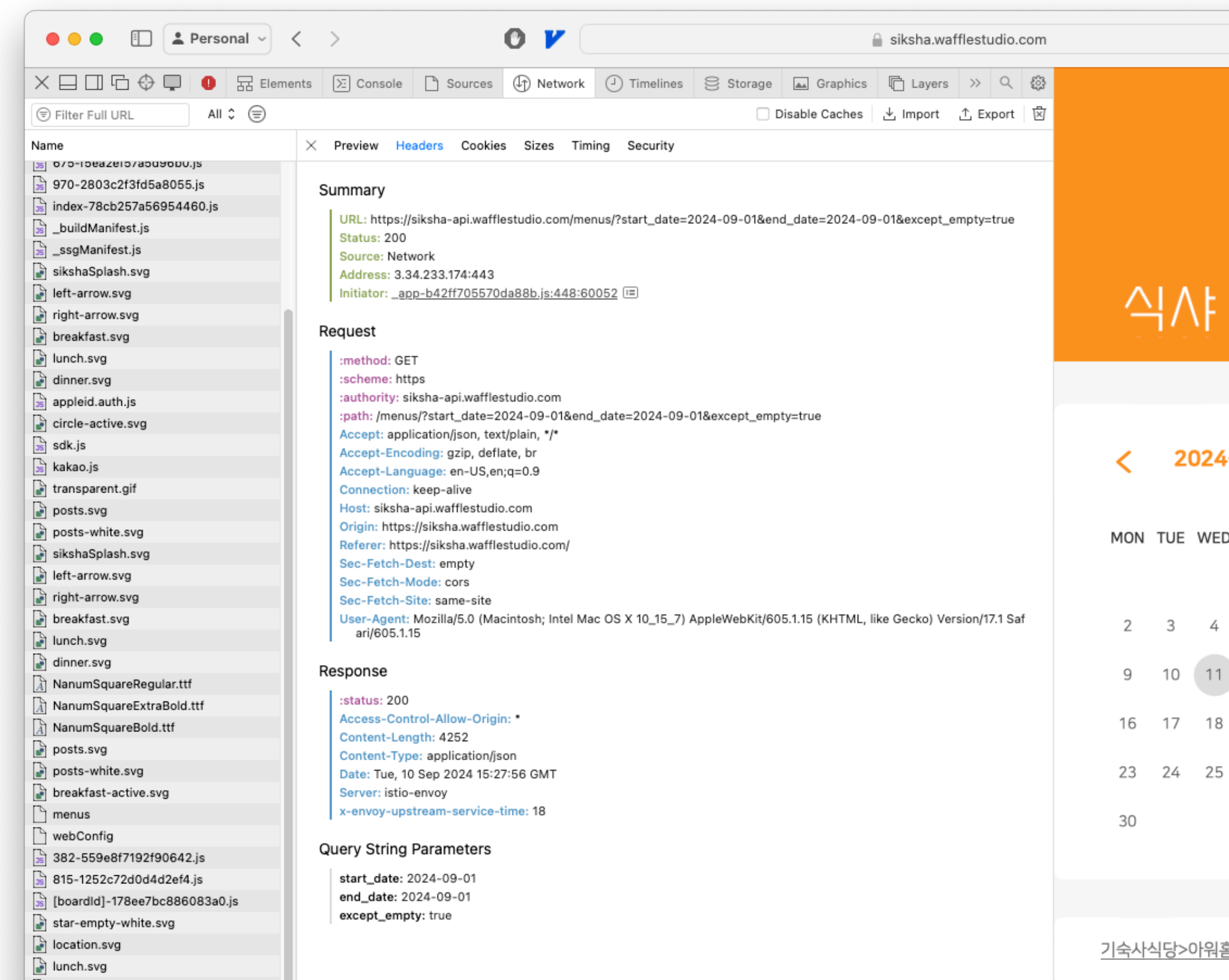
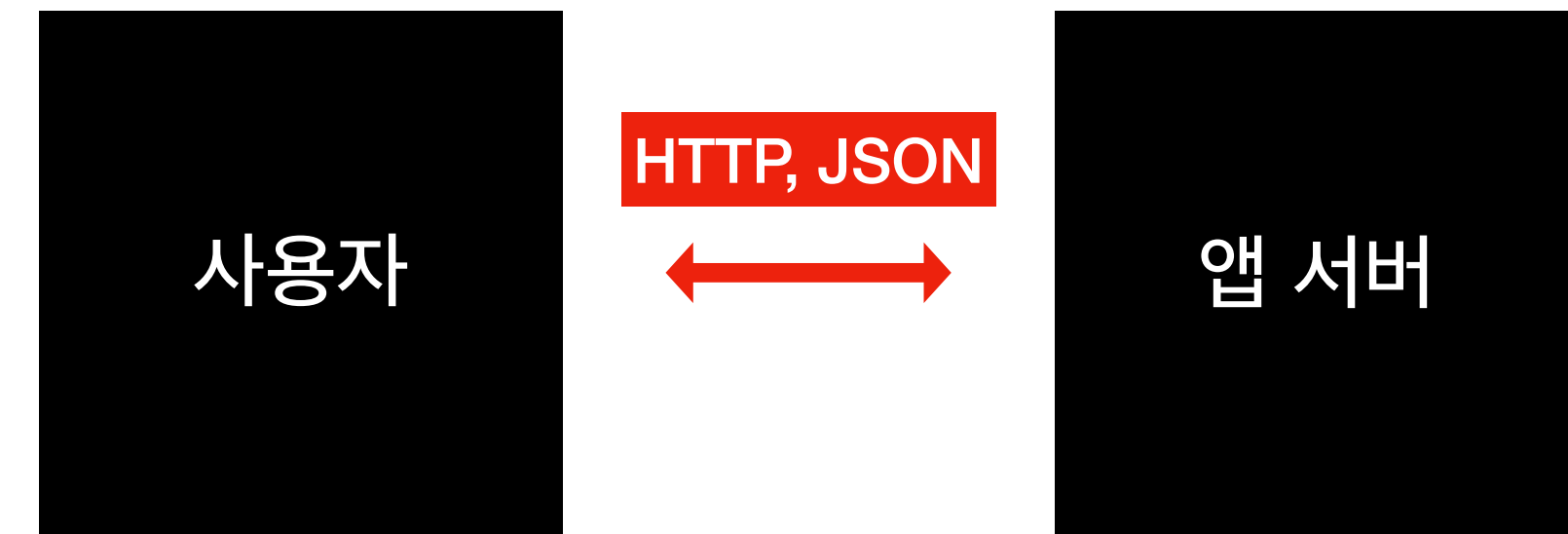


- MySQL
- PostgreSQL
- MongoDB
- ...

# 백엔드 서버의 데이터 흐름

## 사용자와 앱 서버

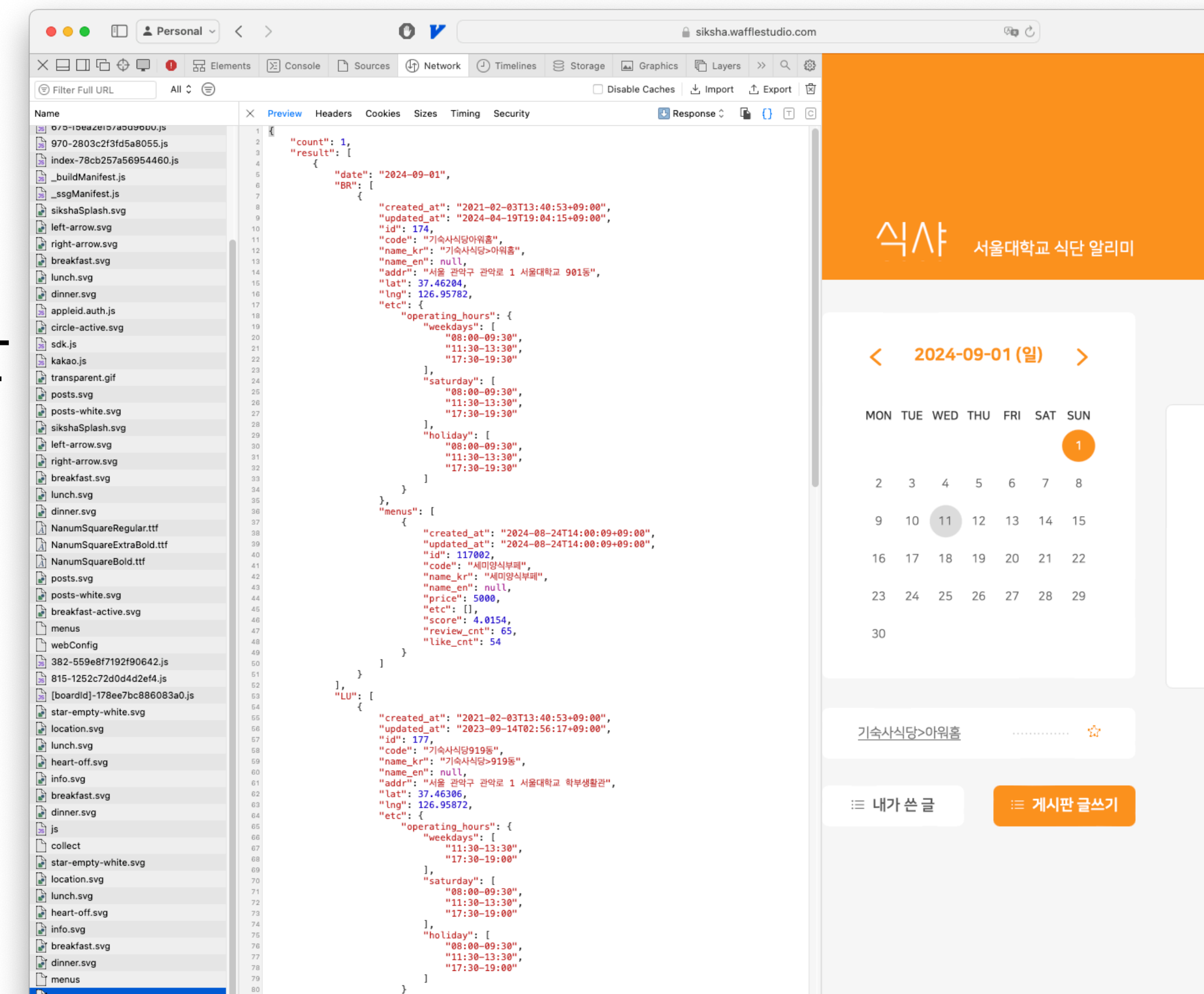
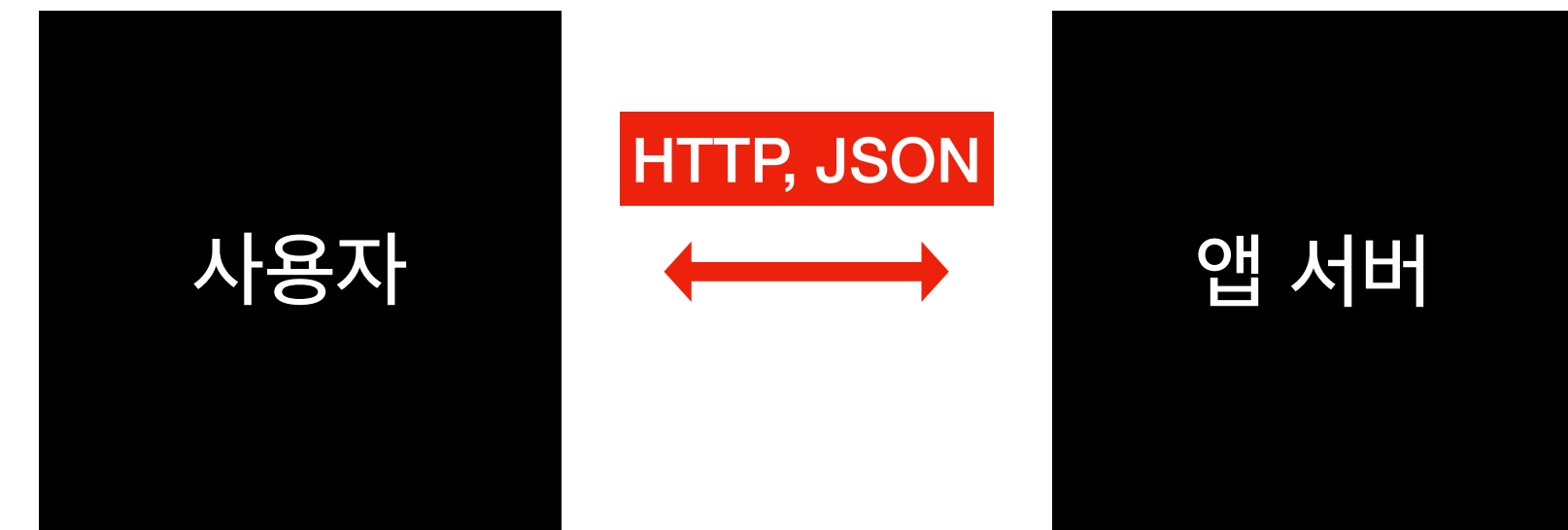
- HTTP / HTTPS
- 사용자와 앱 서버 간의 통신 규약
- Header / Body 의 형태로 데이터를 주고 받자
- URL (<http://www.google.com>)
- Method (GET, POST, PUT, DELETE ...)
- Status Code (200, 404, 500, 503 ...)



# 백엔드 서버의 데이터 흐름

## 사용자와 앱 서버

- JSON Schema
- 데이터를 직렬화 하는 방식
- HTTP 소통 시 Body 를 JSON 으로 주고 받는다.
- -> 사용자(프론트 개발자) 와 JSON 스키마에 대한 규칙을 잘 정의해야 한다



# 백엔드 서버의 데이터 흐름

## 앱 서버와 데이터베이스



- SQL
  - 데이터 베이스의 데이터를 조작하기 위한 소통 수단
  - DML (Data Manipulation Language)
    - 데이터 추가 / 데이터 읽기 / 데이터 삭제 / 데이터 수정 (CRUD)
  - DDL (Data Definition Language)
    - 데이터를 어떻게 저장할지 그 규격을 정의

# 앱 서버 뜯어보기

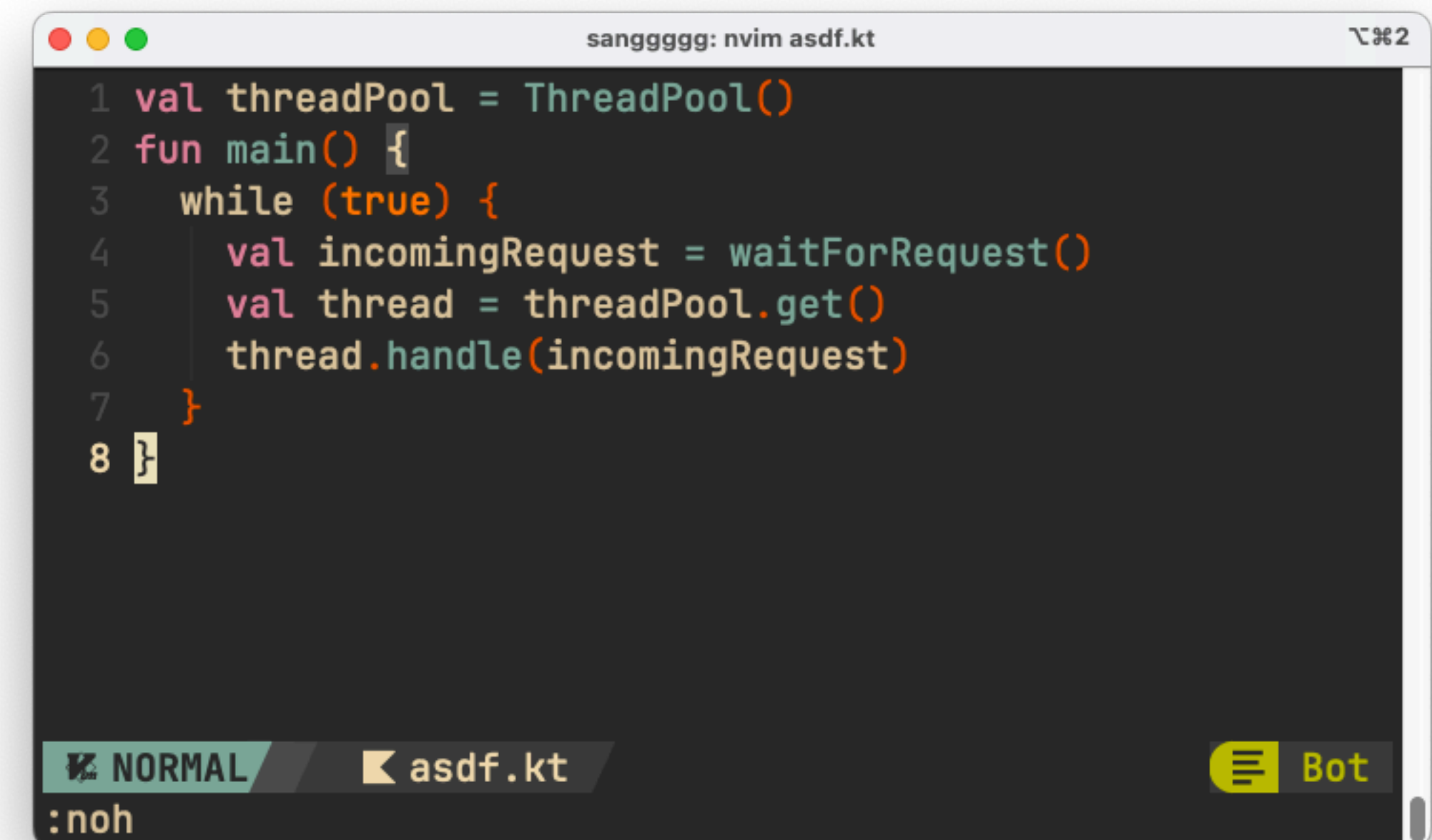
## Spring Boot 가 하는 일

- JVM 위에서 서버 개발에 필요로 하는 대부분의 편의 기능 제공
  - Web Server
  - DI (Dependency Injection)
  - DB Connection
  - ...

# 앱 서버 뜯어보기

## Web Server

- 외부 Network IO 를 기다리고, 새로운 요청에 대한 응답을 처리하는 작업을 진행한다.
- Spring 은 주로 Tomcat 을 쓴다. (Jetty, Netty 등의 다른 Web Server 로 갈아끼울 수도 있다.)



```
sanggggg: nvim asdf.kt
1 val threadPool = ThreadPool()
2 fun main() {
3     while (true) {
4         val incomingRequest = waitForRequest()
5         val thread = threadPool.get()
6         thread.handle(incomingRequest)
7     }
8 }
```

NORMAL asdf.kt Bot :noh



# 앱 서버 뜯어보기

## Dependency Injection

- Component 를 조립하며 개발을 할 때, 코드를 효율적으로 관리하기 위한 방법
- 내가 의존하는 Component 의 생성을 내가 하지 않고, DI 프레임 워크가 진행한 후 만들어진 Component 를 가져다 주면 이를 사용한다.
- Pros
  - 공유되는 Component 를 잘 관리할 수 있다.
  - 직접 생성을 피해서, Interface 에 의존할 수 있다.
- Cons : 어렵다



The screenshot shows a code editor window titled 'sanggggg: nvim asdf.kt'. The code is written in Kotlin and illustrates a simple Dependency Injection setup. It defines a 'Server' class that depends on 'UserService' and 'ProductService' interfaces. The 'Server' class has private val properties for these services and a placeholder for future implementation. Below the 'Server' class, the 'UserService' and 'ProductService' classes are defined as empty classes.

```
1
2 class Server(
3     private val userService: UserService,
4     private val productService: ProductService,
5 ) {
6     // ...
7 }
8
9 class UserService()
10
11 class ProductService()
```

At the bottom of the editor, there is a status bar with 'NORMAL' mode, the filename 'asdf.kt', and a 'Bot' icon.

# 앱 서버 뜯어보기

## DB Connection

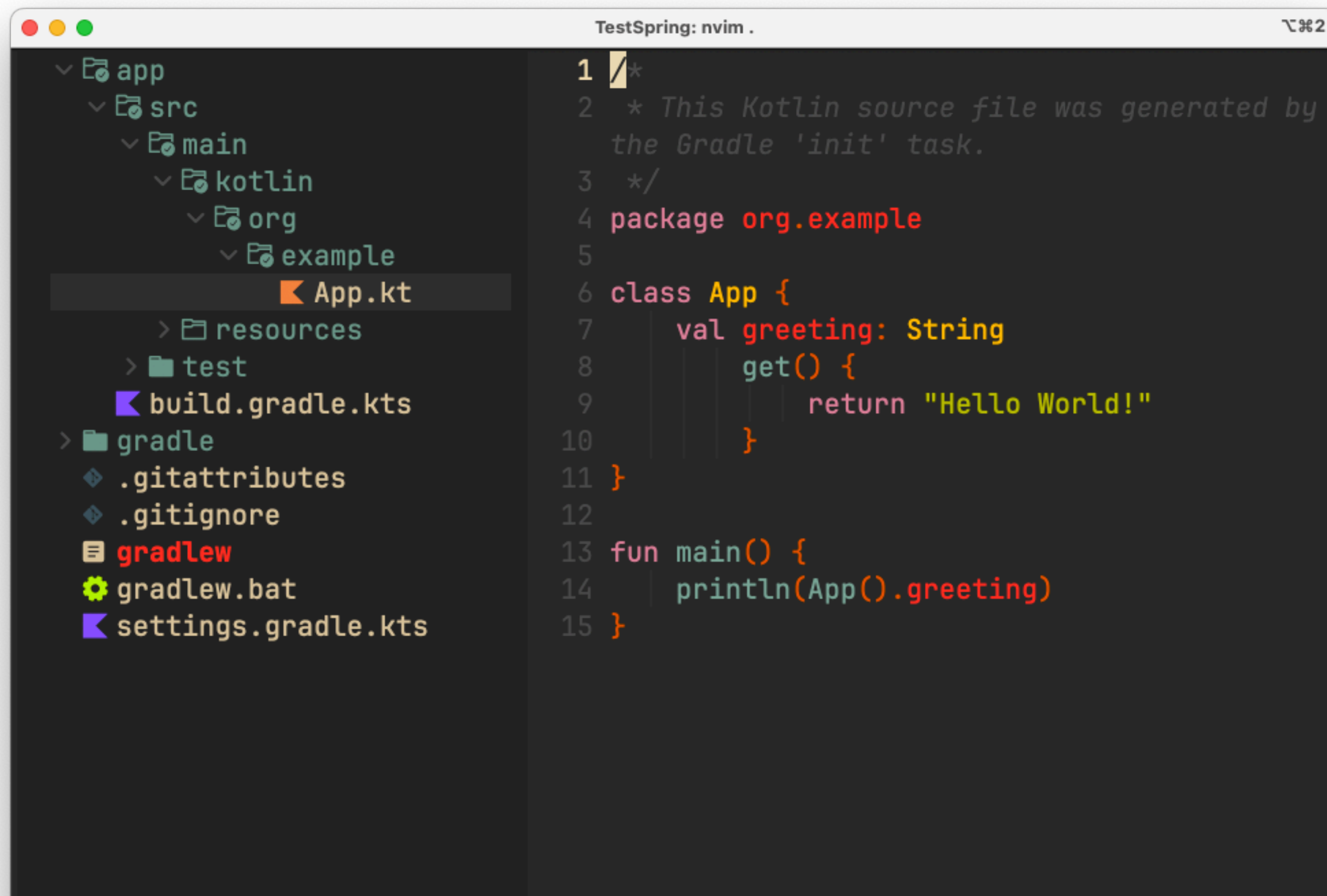
- DB 와 소통하기 위한 툴들
  - DB 에 연결 맺고 소통하고 싶다 (**DB Connector**)
  - DB 와 커넥션을 효율적으로 관리하고 싶다 (**Connection Pool**)
  - 매번 SQL 을 직접 쓰기 싫다 (**Query Builder**)
  - 매번 SQL 결과 (plain text) 를 Kotlin 객체로 매핑하기 귀찮다 (**Object Relation Mapper**)
- Spring 에서 제공하는 [**jdbc, Hikari CP, JPA**] 를 사용할 예정



# Spring Boot 서버 띄워보기

## Gradle Project 생성

- `gradle init` & Select “application”, “kotlin” ...
- Add dependencies for Spring Boot (on assignment template)



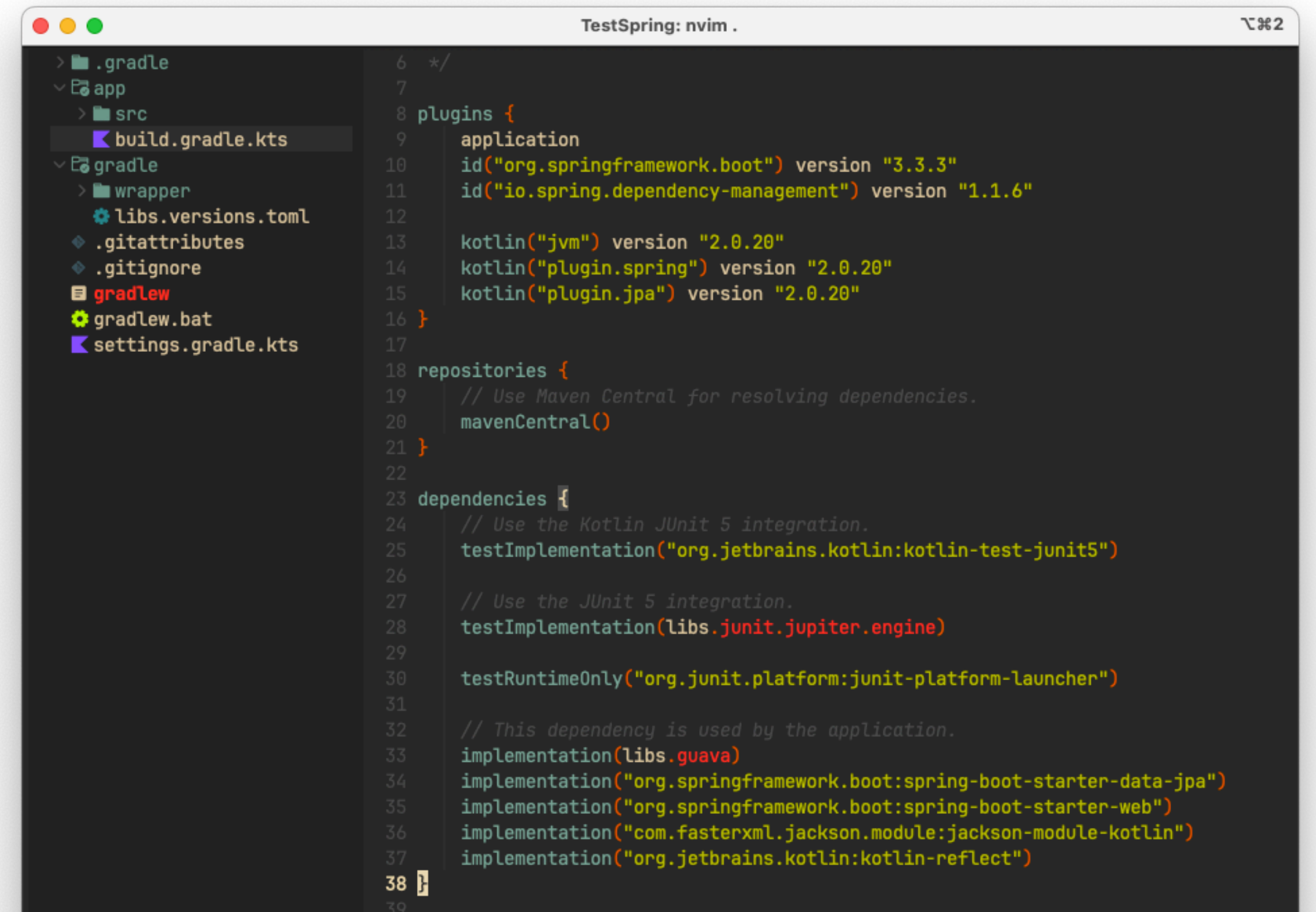
The screenshot shows a code editor window titled 'TestSpring: nvim .'. The left sidebar displays a file tree with the following structure:

- app
  - src
    - main
      - kotlin
        - org
          - example
            - App.kt (selected)- resources
- test
- build.gradle.kts

- gradle
- .gitattributes
- .gitignore
- gradlew
- gradlew.bat
- settings.gradle.kts

The main editor area shows the content of App.kt:

```
1 /*  
2  * This Kotlin source file was generated by  
3  * the Gradle 'init' task.  
4  */  
5  
6 package org.example  
7  
8 class App {  
9     val greeting: String  
10  
11     get() {  
12         return "Hello World!"  
13     }  
14  
15 fun main() {  
16     println(App().greeting)  
17 }  
18
```



The screenshot shows a code editor window titled 'TestSpring: nvim .'. The left sidebar displays a file tree with the following structure:

- .gradle
- app
  - src
    - build.gradle.kts (selected)
- gradle
  - libs.versions.toml
  - wrapper
    - libs.versions.toml
    - .gitattributes
    - .gitignore
    - gradlew
    - gradlew.bat
    - settings.gradle.kts

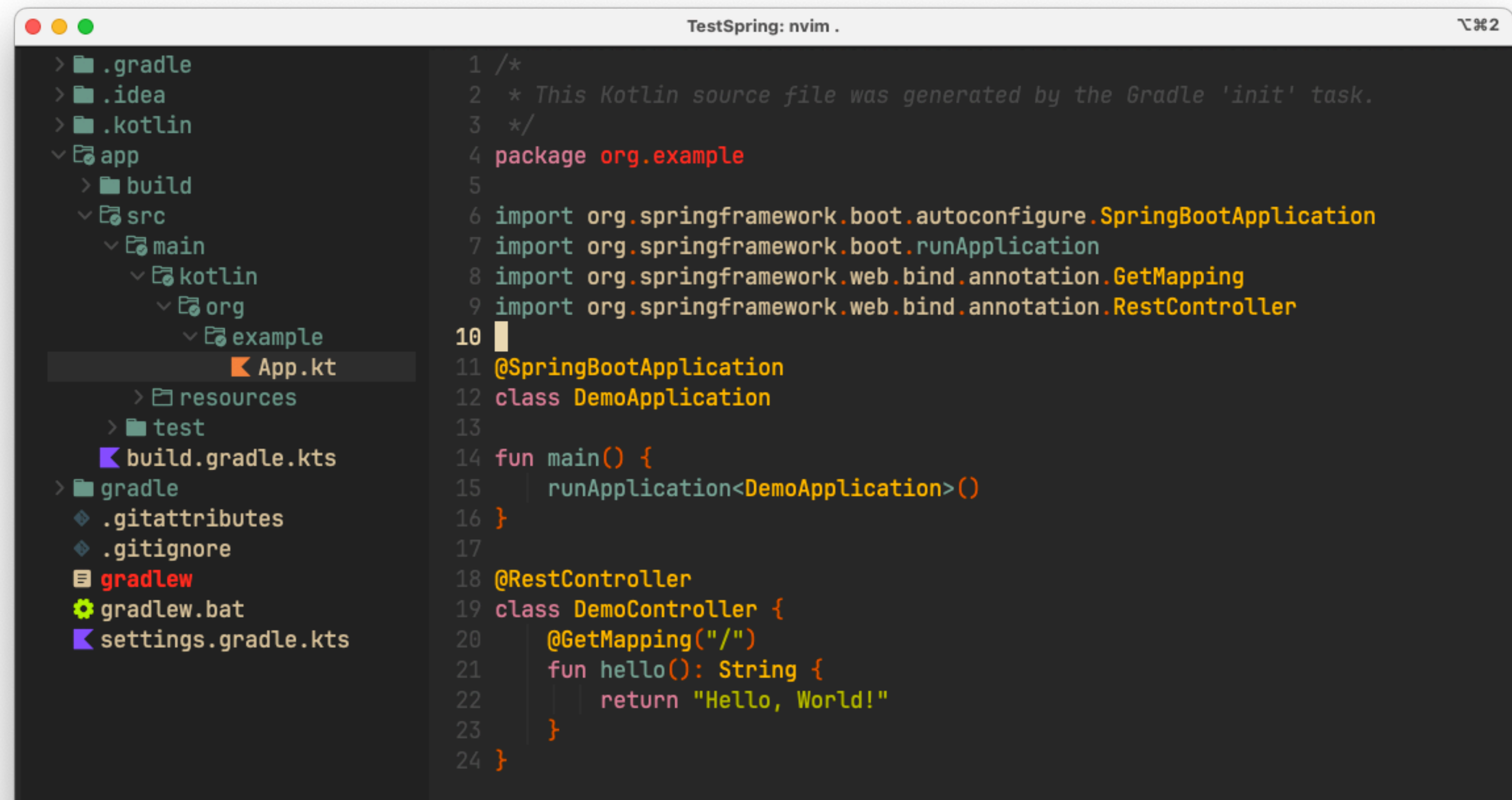
The main editor area shows the content of build.gradle.kts:

```
1  
2 */  
3  
4 plugins {  
5     application  
6     id("org.springframework.boot") version "3.3.3"  
7     id("io.spring.dependency-management") version "1.1.6"  
8  
9     kotlin("jvm") version "2.0.20"  
10    kotlin("plugin.spring") version "2.0.20"  
11    kotlin("plugin.jpa") version "2.0.20"  
12 }  
13  
14 repositories {  
15     // Use Maven Central for resolving dependencies.  
16     mavenCentral()  
17 }  
18  
19 dependencies {  
20     // Use the Kotlin JUnit 5 integration.  
21     testImplementation("org.jetbrains.kotlin:kotlin-test-junit5")  
22  
23     // Use the JUnit 5 integration.  
24     testImplementation(libs.junit.jupiter.engine)  
25  
26     testRuntimeOnly("org.junit.platform:junit-platform-launcher")  
27  
28     // This dependency is used by the application.  
29     implementation(libs.guava)  
30     implementation("org.springframework.boot:spring-boot-starter-data-jpa")  
31     implementation("org.springframework.boot:spring-boot-starter-web")  
32     implementation("com.fasterxml.jackson.module:jackson-module-kotlin")  
33     implementation("org.jetbrains.kotlin:kotlin-reflect")  
34 }  
35
```

# Spring Boot 서버 띄워보기

## Entry point - Spring Application

- `@SpringBootApplication` 어노테이션이 부착된 클래스
- 내 프로젝트의 Component 들을 생성하고 요청을 처리할 수 있도록 한다.



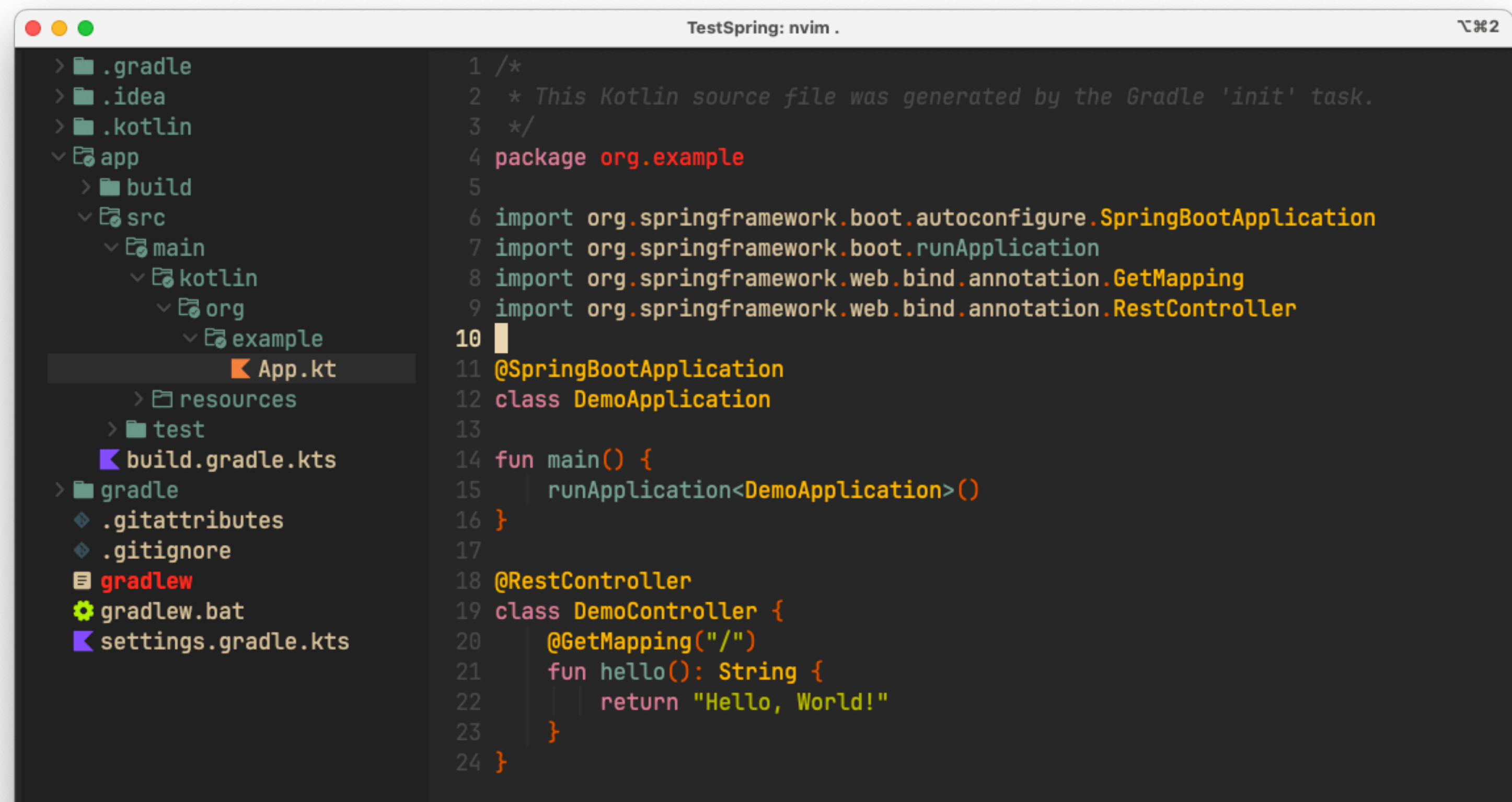
The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with folders like .gradle, .idea, .kotlin, app, build, src, main, kotlin, org, example, resources, test, gradle, .gitattributes, .gitignore, gradlew, gradlew.bat, and settings.gradle.kts. The code editor shows the main class of the application, App.kt, which is a Kotlin file generated by the Gradle 'init' task. The code defines a package org.example, imports Spring Boot and Spring Framework classes, and defines a DemoApplication class with a main method and a DemoController class with a hello method.

```
1 /*
2  * This Kotlin source file was generated by the Gradle 'init' task.
3  */
4 package org.example
5
6 import org.springframework.boot.autoconfigure.SpringBootApplication
7 import org.springframework.boot.runApplication
8 import org.springframework.web.bind.annotation.GetMapping
9 import org.springframework.web.bind.annotation.RestController
10
11 @SpringBootApplication
12 class DemoApplication
13
14 fun main() {
15     runApplication<DemoApplication>()
16 }
17
18 @RestController
19 class DemoController {
20     @GetMapping("/")
21     fun hello(): String {
22         return "Hello, World!"
23     }
24 }
```

# Spring Boot 서버 띄워보기

## API Endpoint - REST Controller

- `@RestController` 어노테이션이 부착된 클래스
- 요청이 들어오는 URL, HTTP Method, URL Parameters, Query Parameters 에 대해 어떻게 처리해야 하는지 명시



The screenshot shows a code editor with a project structure on the left and a Kotlin source file on the right. The project structure includes folders like .gradle, .idea, .kotlin, app, build, src, main, kotlin, org, example, resources, test, build.gradle.kts, gradle, .gitattributes, .gitignore, gradlew, gradlew.bat, and settings.gradle.kts. The source file, App.kt, contains the following code:

```
1 /*
2  * This Kotlin source file was generated by the Gradle 'init' task.
3  */
4 package org.example
5
6 import org.springframework.boot.autoconfigure.SpringBootApplication
7 import org.springframework.boot.runApplication
8 import org.springframework.web.bind.annotation.GetMapping
9 import org.springframework.web.bind.annotation.RestController
10
11 @SpringBootApplication
12 class DemoApplication
13
14 fun main() {
15     runApplication<DemoApplication>()
16 }
17
18 @RestController
19 class DemoController {
20     @GetMapping("/")
21     fun hello(): String {
22         return "Hello, World!"
23     }
24 }
```

# References

For more details, check out below

- Tutorials
  - <https://docs.spring.io/spring-boot/tutorial/first-application/index.html>
  - <https://docs.spring.io/spring-boot/reference/web/servlet.html>
- Previous seminars
  - <https://github.com/wafflestudio/seminar-2023/tree/main/spring> (Thanks to @PFCJeong)
  - <https://github.com/wafflestudio/seminar-2022/tree/main/spring> (Thanks to @Jhvictor4)