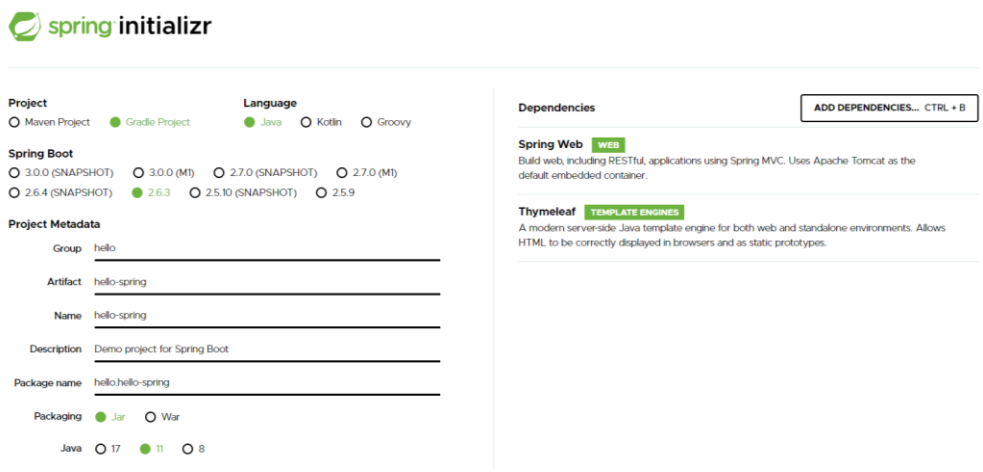
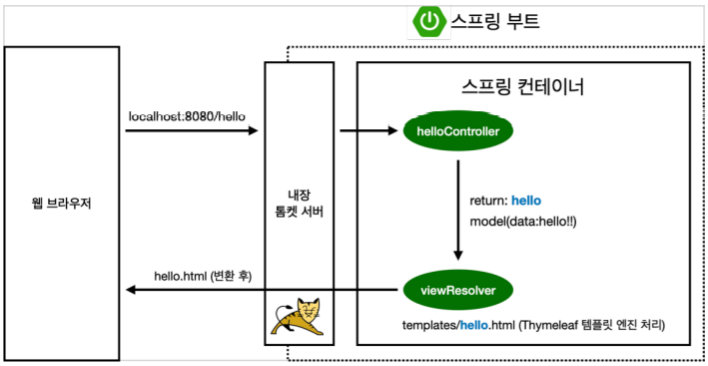


[start.spring.io 페이지 접속 후 초기 프로젝트 파일 생성]

- **스프링 부트 라이브러리 (참고만)**
  - spring-boot-starter-web
    - spring-boot-starter-tomcat: 톰캣 (웹 서버)
    - spring-webmvc: 스프링 웹 MVC
  - spring-boot-starter-thymeleaf: 타임리프 템플릿 엔진 (View)
  - spring-boot-starter(공통): 스프링 부트 + 스프링 코어 + 로깅
    - spring-boot
      - spring-core
    - spring-boot-starter-logging
      - logback, slf4j

- **테스트 라이브러리**
  - spring-boot-starter-test
    - junit: 테스트 프레임워크
    - mockito: 목 라이브러리
    - assert: 테스트 코드를 더 편하게 작성하도록 도와주는 라이브러리
    - spring-test: 스프링 통합 테스트 지원

- **thymeleaf 템플릿 엔진 동작 환경**



- 컨트롤러에서 리턴 값으로 문자를 반환하면 viewResolver가 화면을 찾아서 처리
  - 스프링 부트 템플릿 엔진 기본 viewName 매핑
  - Resources:templates/ + {ViewName} + .html

- **정적 컨텐츠**
  - 스프링 부트 정적 컨텐츠 기능
    - resources/static/hello-static.html
  - 웹 브라우저에서 localhost:8080/hello-static.html을 요청하면 내장 톰캣 서버에서 요청을 받고, 스프링 컨테이너에 넘김. 스프링은 hello-static 관련 컨트롤러를 찾음. 해당 컨트롤러가 없기 때문에 내부 html 파일을 찾아 바로 브라우저에 반환

- **@GetMapping("/{id}")**
  - localhost:8080/{id}와 같이 url이 매핑
  - @RequestMapping(method = RequestMethod.GET …) 방식과 동일
  - 파라미터를 입력 받아 API를 만드는 경우: @RequestParam 사용

- **MVC와 템플릿 엔진**
  - Model, View, Controller
  - Controller: 내부 실행과 관련
    ↳ @Controller로 해당 파일(?)이 컨트롤러임을 명시
  - View: 보여지는 환경 관련

- **API**
  - @ResponseBody: http에서 body 부분에 데이터를 직접 넣어줄 것을 명시
    ↳ [사용 원리]
    - HTTP의 BODY에 문자 내용을 직접 반환
    - 'viewResolver' 대신 'HttpMessageConverter'가 동작
    - 기본 문자 처리: 'StringHttpMessageConverter'
    - 기본 객체 처리: 'MappingJackson2HttpMessageConverter'
    - byte 등등 여러 HttpMessageConverter가 기본으로 등록되어 있음
  - 뷰, 모델 등이 없다는 점에서 템플릿 엔진과 차이가 있음

- **Long과 long**
  - Long: Wrapper Class
  - Long: 순수한 자바 자료형
  - 둘 다 8byte

- **컴포넌트 스캔과 의존 관계 직접 주입**
  - @Component: 컴포넌트 스캔에서 사용
  - @Controller: 스프링 MVC 컨트롤러에서 사용
  - @Service: 스프링 비즈니스 로직에서 사용
  - @Repository: 스프링 데이터 접근 계층에서 사용
  - @Configuration: 스프링 설정 정보에서 사용
  - @Autowired: 자동 의존 관계를 주입

- **스프링 빈 등록**
  - (@Controller 제외) @Service, @Repository, @Autowired 등의 애노테이션을 제거하고 진행
  - DI에는 필드 주입, setter 주입, 생성자 주입의 3가지 방법이 있는데, 의존 관계가 실행 중 동적으로 변하는 경우는 거의 없으므로 생성자 주입 권장
  - 실무에서는 정형화된 컨트롤러, 서비스, 리포지토리와 같은 코드는 컴포넌트 스캔 사용. 정형화되지 않거나 상황에 따라 구현 클래스를 변경해야 한다면 설정을 통해 스프링 빈으로 등록
  - ** @Autowired를 통한 DI는 스프링이 관리하는 객체에서만 동작

- **데이터베이스 연결 환경 설정**
  - Build.gradle 파일에 jdbc, h2 데이터베이스 관련 라이브러리 추가
    - Implementation ‘org.springframework.boot:spring-boot-starter-jdbc’
    - runtimeOnly ‘com.h2databse:h2’
  - 스프링 부트 데이터베이스 연결 설정 추가 (resources/application.properties)
    - spring.datasource.url=jdbc:h2:tcp://localhost/~/test
    - spring.datasource.driver-class-name=org.h2.Driver
    - spring.datasource.username=sa

- **@Transactional**
  - 해당 애노테이션을 테스트 케이스에 달면, 테스트를 실행할 때 트랜잭션을 실행하며 DB에 데이터를 모두 insert. 테스트가 끝나면 롤백 => DB 데이터가 지워짐
  - DB에 데이터가 남지 않으므로 다음 테스트에 영향을 주지 않음

- **개방·폐쇄 원칙(OCP; Open-Closed Principle)**
  - 확장에는 열려 있고, 수정/변경에는 닫혀 있음
  - 스프링의 DI를 사용하면 기존 코드를 전혀 변경하지 않고 설정만으로 구현 클래스 변경 가능