The LockManager App Case Study: Client Structure & Functionality

Douglas C. Schmidt

d.schmidt@vanderbilt.edu

www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science

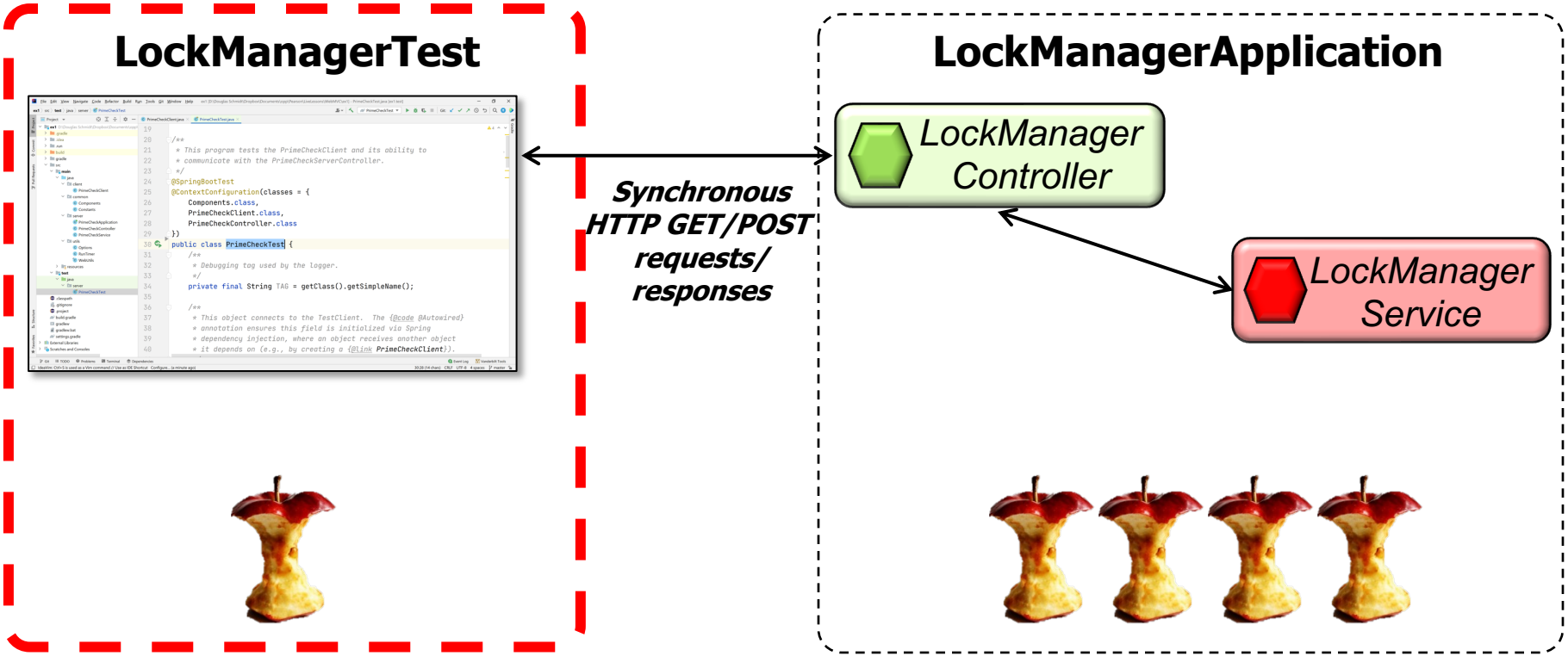
Institute for Software Integrated Systems

Vanderbilt University

Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

- Understand the structure & functionality of client components that send/receive HTTP GET/POST requests/responses to/from the microservice synchronously



See github.com/douglascraigschmidt/LiveLessons/tree/master/WebMVC/ex5

The Structure & Functionality of LockAPI Interface

The Structure & Functionality of the LockAPI Interface

- The LockAPI interface hides details of remote method invocations via HTTP

```
public interface LockAPI {
  @PostExchange(CREATE)
  Boolean create(@RequestBody Integer maxLocks);

  @GetExchange(ACQUIRE_LOCK)
  Lock acquire();

  @GetExchange(ACQUIRE_LOCKS)
  List<Lock> acquire(@RequestParam Integer permits);

  @PostExchange(RELEASE_LOCK)
  Boolean release(@RequestBody Lock lock);

  @PostExchange(RELEASE_LOCKS)
  Boolean release(@RequestBody List<Lock> locks);
}
```

This design uses the new Spring 6 declarative HTTP interface features

See www.baeldung.com/spring-6-http-interface

The Structure & Functionality of the LockAPI Interface

- The LockAPI interface hides details of remote method invocations via HTTP

```
public interface LockAPI {
  @PostExchange(CREATE)
  Boolean create(@RequestBody Integer maxLocks);

  @GetExchange(ACQUIRE_LOCK)
  Lock acquire();

  @GetExchange(ACQUIRE_LOCKS)
  List<Lock> acquire(@RequestParam Integer permits);

  @PostExchange(RELEASE_LOCK)
  Boolean release(@RequestBody Lock lock);

  @PostExchange(RELEASE_LOCKS)
  Boolean release(@RequestBody List<Lock> locks);
}
```

These proxy methods shield clients from low-level details of HTTP programming

The Spring 6 HTTP interface features are much cleaner than using Retrofit!

The Structure & Functionality of the LockAPI Interface

- The LockAPI interface hides details of remote method invocations via HTTP

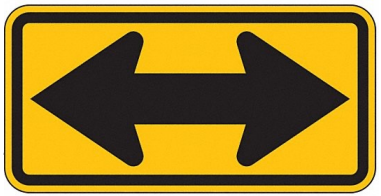
```
public interface LockAPI {
   @PostExchange(CREATE)
   Boolean create(@RequestBody Integer maxLocks)

   @GetExchange(ACQUIRE_LOCK)
   Lock acquire();

   @GetExchange(ACQUIRE_LOCKS)
   List<Lock> acquire(@RequestParam Integer permits);

   @PostExchange(RELEASE_LOCK)
   Boolean release(@RequestBody Lock lock);

   @PostExchange(RELEASE_LOCKS)
   Boolean release(@RequestBody List<Lock> locks);
}
```



These calls are all synchronous & return conventional Java types

The Structure & Functionality of the LockAPI Interface

- The LockAPI interface hides details of remote method invocations via HTTP

```
public interface LockAPI {
  @PostExchange(CREATE)
  Boolean create(@RequestBody Integer maxLocks);

  @GetExchange(ACQUIRE_LOCK)
  Lock acquire();

  @GetExchange(ACQUIRE_LOCKS)
  List<Lock> acquire(@RequestParam Integer permits);

  @PostExchange(RELEASE_LOCK)
  Boolean release(@RequestBody Lock lock);

  @PostExchange(RELEASE_LOCKS)
  Boolean release(@RequestBody List<Lock> locks);
}
```

These annotations mark a method as an HTTP endpoint

See http-declarative-http-client-httpexchange/#3-creating-an-http-service-interface

The Structure & Functionality of the LockAPI Interface

- The LockAPI interface hides details of remote method invocations via HTTP

```
public interface LockAPI {
   @PostExchange(CREATE)
   Boolean create(@RequestBody Integer maxLocks);

   @GetExchange(ACQUIRE_LOCK)
   Lock acquire();

   @GetExchange(ACQUIRE_LOCKS)
   List<Lock> acquire(@RequestParam Integer permits);

   @PostExchange(RELEASE_LOCK)
   Boolean release(@RequestBody Lock lock);

   @PostExchange(RELEASE_LOCKS)
   Boolean release(@RequestBody List<Lock> locks);
}
```

These paths identify a specific HTTP endpoint

The Structure & Functionality of the LockAPI Interface

- The LockAPI interface hides details of remote method invocations via HTTP

```
public interface LockAPI {
  @PostExchange(CREATE)
  Boolean create(@RequestBody Integer maxLocks);

  @GetExchange(ACQUIRE_LOCK)
  Lock acquire();

  @GetExchange(ACQUIRE_LOCKS)
  List<Lock> acquire(@RequestParam Integer permits);

  @PostExchange(RELEASE_LOCK)
  Boolean release(@RequestBody Lock lock);

  @PostExchange(RELEASE_LOCKS)
  Boolean release(@RequestBody List<Lock> locks);
}
```

These annotations are the same ones used by a Spring controller

See www.java67.com/2023/02/requestparam-vs-requestbody-in-spring.html

Creating an Instance of the LockAPI Interface

Creating an Instance of the LockAPI Interface

- The ClientBeans class contains a factory method bean that creates the LockAPI proxy that uses the Spring 6 HTTP interface features

```
@Component
public class ClientBeans {
  @Bean
  public LockAPI getLockAPI() {
    var webClient = WebClient.builder()
        .baseUrl(LOCK_MANAGER_SERVER_BASE_URL).build();

    return HttpServiceProxyFactory
        .builder(WebClientAdapter
                 .forClient(webClient))
        .build()
        .createClient(LockAPI.class);
  ...
```

See WebMVC/ex5/src/test/java/edu/vandy/lockmanager/ClientBeans.java

Creating an Instance of the LockAPI Interface

- The ClientBeans class contains a factory method bean that creates the LockAPI proxy that uses the Spring 6 HTTP interface features

```
@Component
public class ClientBeans {
  @Bean
  public LockAPI getLockAPI() {
    var webClient = WebClient.builder()
        .baseUrl(LOCK_MANAGER_SERVER_BASE_URL).build();

    return HttpServiceProxyFactory
        .builder(WebClientAdapter
                 .forClient(webClient))
        .build()
        .createClient(LockAPI.class);
  ...
```

This @Bean annotation can be injected into classes using Spring's @Autowired annotation

Creating an Instance of the LockAPI Interface

- The ClientBeans class contains a factory method bean that creates the LockAPI proxy that uses the Spring 6 HTTP interface features

```
@Component
public class ClientBeans {
  @Bean
  public LockAPI getLockAPI() {
    var webClient = WebClient.builder()
        .baseUrl(LOCK_MANAGER_SERVER_BASE_URL).build();

    return HttpServiceProxyFactory
        .builder(WebClientAdapter
                 .forClient(webClient))
        .build()
        .createClient(LockAPI.class);
  ...
```

Create the main entry point for performing web requests (for both sync & async calls)

See www.baeldung.com/spring-5-webclient

Creating an Instance of the LockAPI Interface

- The ClientBeans class contains a factory method bean that creates the LockAPI proxy that uses the Spring 6 HTTP interface features

```
@Component
public class ClientBeans {
  @Bean
  public LockAPI getLockAPI() {
    var webClient = WebClient.builder()
        .baseUrl(LOCK_MANAGER_SERVER_BASE_URL).build();

    return HttpServiceProxyFactory
        .builder(WebClientAdapter
                 .forClient(webClient))
        .build()
        .createClient(LockAPI.class);
  ...
```

Adapt the WebClient to provide a synchronous proxy

See www.baeldung.com/spring-6-http-interface

End of the LockManager App Case Study: Client Structure & Functionality