

The LockManager App Case Study: Server Structure & Functionality

Douglas C. Schmidt

d.schmidt@vanderbilt.edu

www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science

**Institute for Software
Integrated Systems**

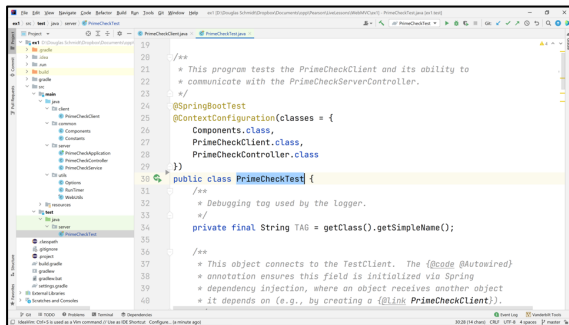
**Vanderbilt University
Nashville, Tennessee, USA**



Learning Objectives in this Part of the Lesson

- This case study shows how Spring WebFlux can be used to send/receive HTTP GET/POST requests asynchronously to/from a LockManager microservice

LockManagerTest



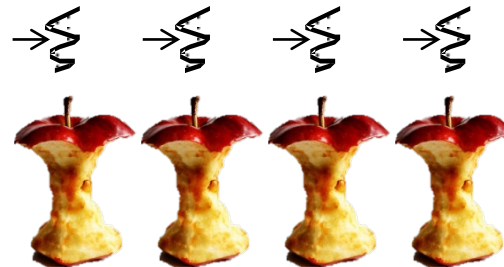
```
19  /**
20   * This program tests the PrimeCheckClient and its ability to
21   * communicate with the PrimeCheckServerController.
22   */
23   //
24   @SpringBootTest
25   @ContextConfiguration(classes = {
26       Components.class,
27       PrimeCheckClient.class,
28       PrimeCheckController.class
29   })
30   public class PrimeCheckTest {
31       /**
32        * Debugging tag used by the logger.
33        */
34       private final String TAG = getClass().getSimpleName();
35
36       /**
37        * This object connects to the TestClient. The @Autowired
38        * annotation ensures this field is initialized via Spring
39        * dependency injection, where an object receives another object
40        * it depends on (e.g., by creating a @Link PrimeCheckClient).
41        */
```



LockManagerApplication



*Asynchronous
HTTP GET/POST
requests/
responses*



See [WebFlux/ex1/src/main/java/edu/vandy/lockmanager/server](https://github.com/vandy-lockmanager/server)

Structure & Functionality of the LockManagerController

Structure & Functionality of the LockManagerController

- LockManagerController maps HTTP GET/POST requests to endpoint handlers

`@RestController`

```
public class LockManagerController {
```

```
    @Autowired
```

```
    LockManagerService mService;
```

```
    ...
```

See [WebFlux/ex1/src/main/java/edu/vandy/lockmanager/server/LockManagerController.java](https://github.com/vandy-lockmanager/server/blob/master/src/main/java/edu/vandy/lockmanager/server/LockManagerController.java)

Structure & Functionality of the LockManagerController

- LockManagerController maps HTTP GET/POST requests to endpoint handlers

@RestController

```
public class LockManagerController {  
    @Autowired  
    LockManagerService mService;  
  
    ...  
}
```

This annotation ensures request handling methods in the controller class automatically serialize return objects into HttpResponse objects

Structure & Functionality of the LockManagerController

- LockManagerController maps HTTP GET/POST requests to endpoint handlers

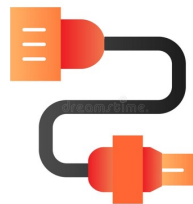
`@RestController`

```
public class LockManagerController {
```

```
    @Autowired
```

```
    LockManagerService mService;
```

```
    ...
```



This field is auto-wired by Spring's dependency injection framework

Structure & Functionality of the LockManagerController

- LockManagerController maps HTTP GET/POST requests to endpoint handlers

@RestController

```
public class LockManagerController {
```

```
...
```

```
@PostMapping(CREATE)
```

```
public Mono<Boolean> create(@RequestBody Integer permitCount)
```

```
@GetMapping(ACQUIRE_LOCK)
```

```
public Mono<Lock> acquire()
```

```
@GetMapping(ACQUIRE_LOCKS)
```

```
public Flux<Lock> acquire(Integer permits)
```

```
@PostMapping(RELEASE_LOCK)
```

```
public Mono<Boolean> release(@RequestBody Lock lock)
```

```
@PostMapping(RELEASE_LOCKS)
```

```
public Mono<Boolean> release(@RequestBody List<Lock> locks)
```

```
...
```

These endpoint handler methods forward to the LockManagerService methods that fulfill the requests

Structure & Functionality of the LockManagerController

- LockManagerController maps HTTP GET/POST requests to endpoint handlers

@RestController

```
public class LockManagerController {
```

```
...
```

```
@PostMapping(CREATE)
```

```
public Mono<Boolean> create(@RequestBody Integer permitCount)
```

```
@GetMapping(ACQUIRE_LOCK)
```

```
public Mono<Lock> acquire()
```

```
@GetMapping(ACQUIRE_LOCKS)
```

```
public Flux<Lock> acquire(Integer permits)
```

```
@PostMapping(RELEASE_LOCK)
```

```
public Mono<Boolean> release(@RequestBody Lock lock)
```

```
@PostMapping(RELEASE_LOCKS)
```

```
public Mono<Boolean> release(@RequestBody List<Lock> locks)
```

```
...
```

*These reactive types enable
async client & server behavior*

See spring.io/blog/2016/04/19/understanding-reactive-types

Structure & Functionality of the LockManagerController

- LockManagerController maps HTTP GET/POST requests to endpoint handlers

@RestController

```
public class LockManagerController {
```

```
...
```

```
@PostMapping(CREATE)
```

```
public Mono<Boolean> create(@RequestBody Integer permitCount)
```

```
@GetMapping(ACQUIRE_LOCK)
```

```
public Mono<Lock> acquire()
```

```
@GetMapping(ACQUIRE_LOCKS)
```

```
public Flux<Lock> acquire(Integer permits)
```

```
@PostMapping(RELEASE_LOCK)
```

```
public Mono<Boolean> release(@RequestBody Lock lock)
```

```
@PostMapping(RELEASE_LOCKS)
```

```
public Mono<Boolean> release(@RequestBody List<Lock> locks)
```

```
...
```

*These annotations map
HTTP GET/POST requests to
endpoint handler methods*

See www.baeldung.com/spring-new-requestmapping-shortcuts

Structure & Functionality of the LockManagerController

- LockManagerController maps HTTP GET/POST requests to endpoint handlers

@RestController

```
public class LockManagerController {  
    ...  
    @PostMapping(CREATE)  
    public Mono<Boolean> create(@RequestBody ...  
    @GetMapping(ACQUIRE_LOCK)  
    public Mono<Lock> acquire()  
    @GetMapping(ACQUIRE_LOCKS)  
    public Flux<Lock> acquire(Integer permits)  
    @PostMapping(RELEASE_LOCK)  
    public Mono<Boolean> release(@RequestBody Lock lock)  
    @PostMapping(RELEASE_LOCKS)  
    public Mono<Boolean> release(@RequestBody List<Lock> locks)  
    ...  
}
```

These strings automatically identify & route to endpoint handler methods from incoming HTTP GET/POST requests

See www.baeldung.com/spring-new-requestmapping-shortcuts

Structure & Functionality of the LockManagerController

- LockManagerController maps HTTP GET/POST requests to endpoint handlers

@RestController

```
public class LockManagerController {
```

```
...
```

```
@PostMapping(CREATE)
```

```
public Mono<Boolean> create(@RequestBody Integer permitCount)
```

```
@GetMapping(ACQUIRE_LOCK)
```

```
public Mono<Lock> acquire()
```

```
@GetMapping(ACQUIRE_LOCKS)
```

```
public Flux<Lock> acquire(Integer permits)
```

```
@PostMapping(RELEASE_LOCK)
```

```
public Mono<Boolean> release(@RequestBody Lock lock)
```

```
@PostMapping(RELEASE_LOCKS)
```

```
public Mono<Boolean> release(@RequestBody List<Lock> locks)
```

```
...
```

*This annotation maps
the HttpRequest body
to a Java object*

See www.baeldung.com/spring-request-response-body

Structure & Functionality of the LockManagerService

Structure & Functionality of the LockManagerService

- LockManagerService defines methods called by LockManagerController, which implements a distributed semaphore using a Java ArrayBlockingQueue

@Service

```
public class LockManagerService {  
    private ArrayBlockingQueue<Lock>  
        mAvailableLocks;  
    ...  
}
```

See <WebFlux/ex1/src/main/java/edu/vandy/lockmanager/server/LockManagerService.java>

Structure & Functionality of the LockManagerService

- LockManagerService defines methods called by LockManagerController, which implements a distributed semaphore using a Java ArrayBlockingQueue

@Service

```
public class LockManagerService {  
    private ArrayBlockingQueue<Lock>  
        mAvailableLocks;  
    ...  
}
```

This annotation indicates the class implements "business logic" & enables auto-detection & wiring of dependent classes via classpath scanning

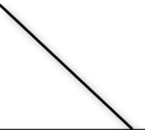
See www.baeldung.com/spring-component-repository-service

Structure & Functionality of the LockManagerService

- LockManagerService defines methods called by LockManagerController, which implements a distributed semaphore using a Java ArrayBlockingQueue

@Service

```
public class LockManagerService {  
    private ArrayBlockingQueue<Lock>  
        mAvailableLocks;  
    ...  
}
```



*Limits concurrent access to the
fixed number of available locks
managed by the LockManagerService*

Structure & Functionality of the LockManagerService

- LockManagerService defines methods called by LockManagerController, which implements a distributed semaphore using a Java ArrayBlockingQueue

@Service

```
public class LockManagerService {  
    ...  
    public Mono<Boolean> create(Integer permitCount) {...}  
    public Mono<Lock> acquire() {...}  
    public Flux<Lock> acquire(Integer permits) {...}  
    public Mono<Boolean> release(Lock lock) {...}  
    public Mono<Boolean> release(List<Lock> locks) {...}  
    ...  
}
```

These methods use the Java ArrayBlockingQueue & reactive types & reactive programming to implement asynchronous distributed semaphore semantics

See next part of the lesson on *"Implementing the Server Components"*

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