The LockManager App Case Study: Server Structure & Functionality

Douglas C. Schmidt

d.schmidt@vanderbilt.edu

www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science

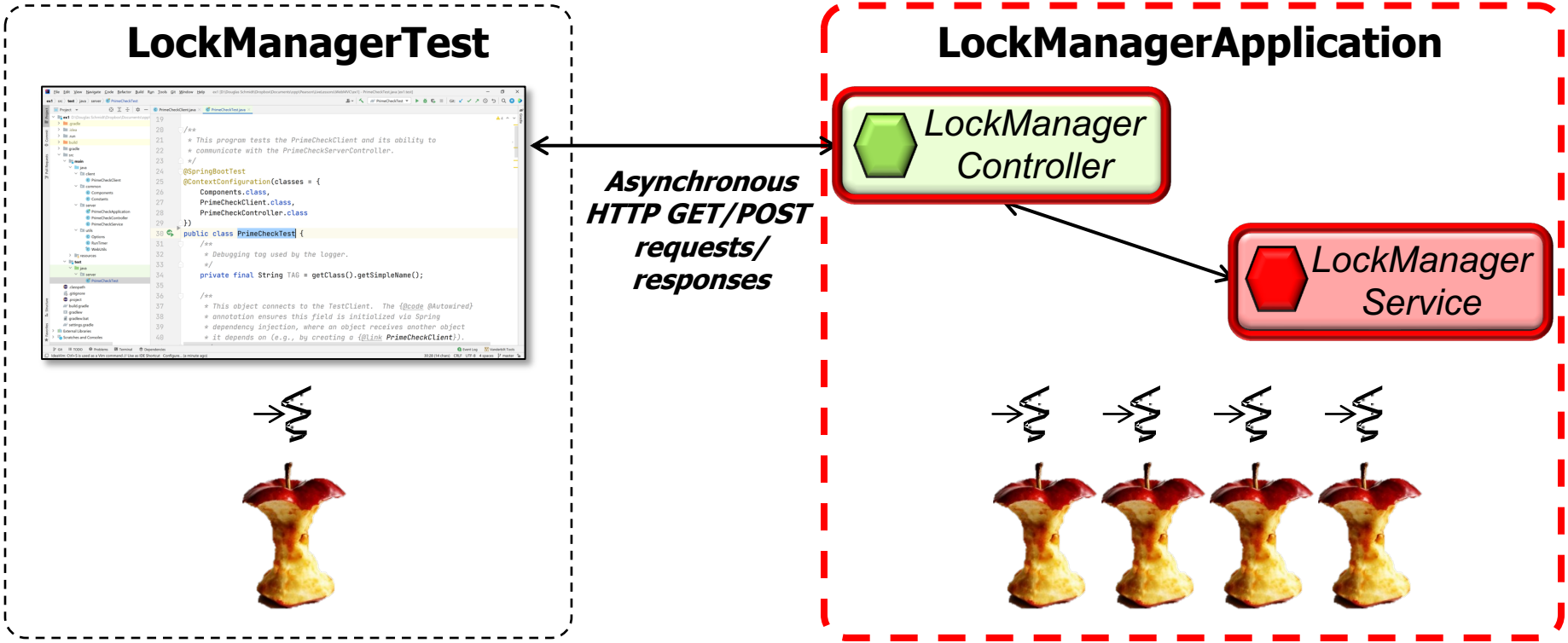
Institute for Software Integrated Systems

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



See WebFlux/ex1/src/main/java/edu/vandy/lockmanager/server

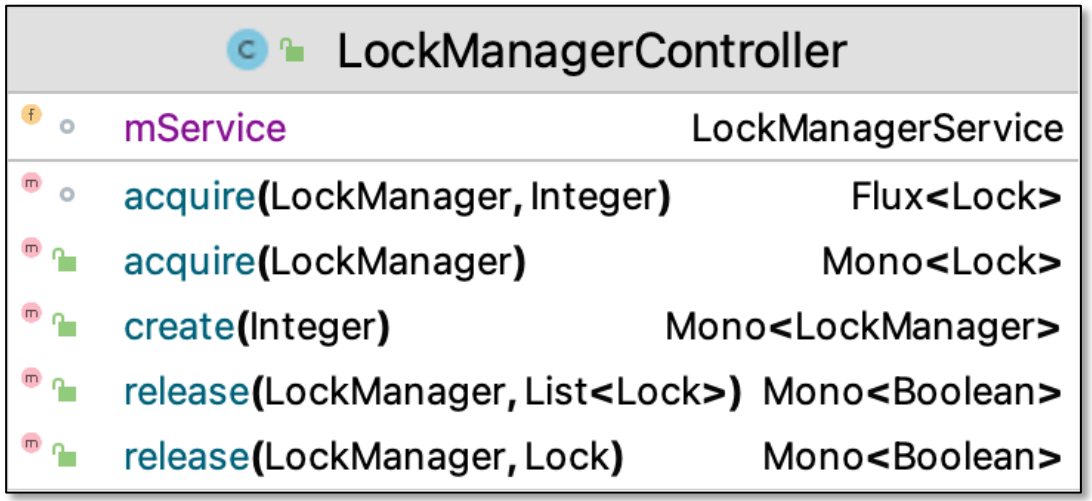
Structure & Functionality of the LockManagerController

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

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

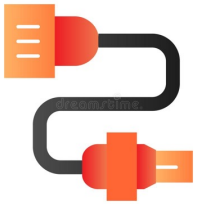
See www.baeldung.com/spring-controller-vs-restcontroller

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

See www.baeldung.com/spring-autowire

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(@RequestB
   @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

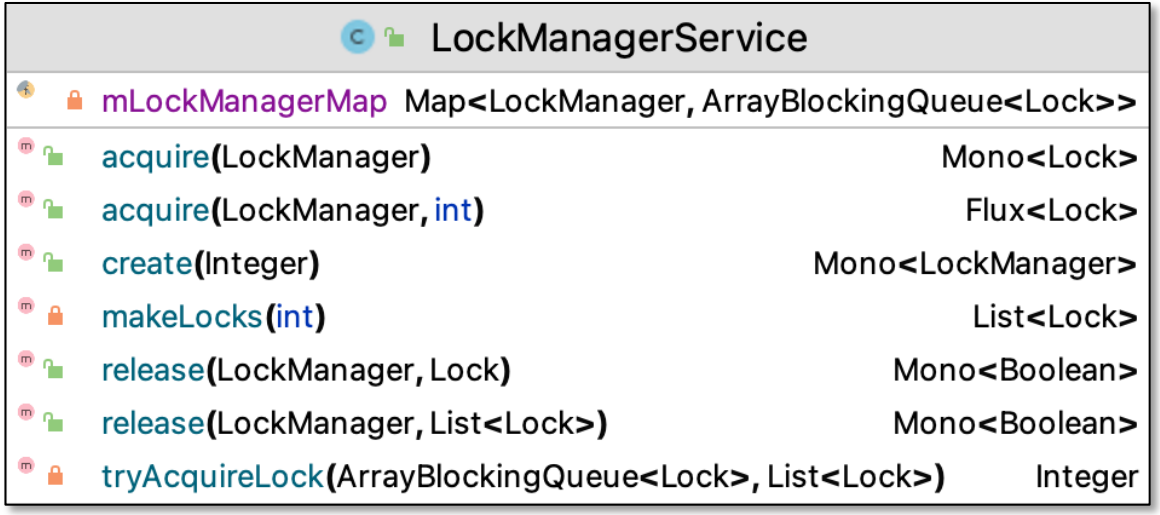
See www.baeldung.com/spring-request-response-body

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

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ArrayBlockingQueue.html

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