The LockManager App Case Study: Overview

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

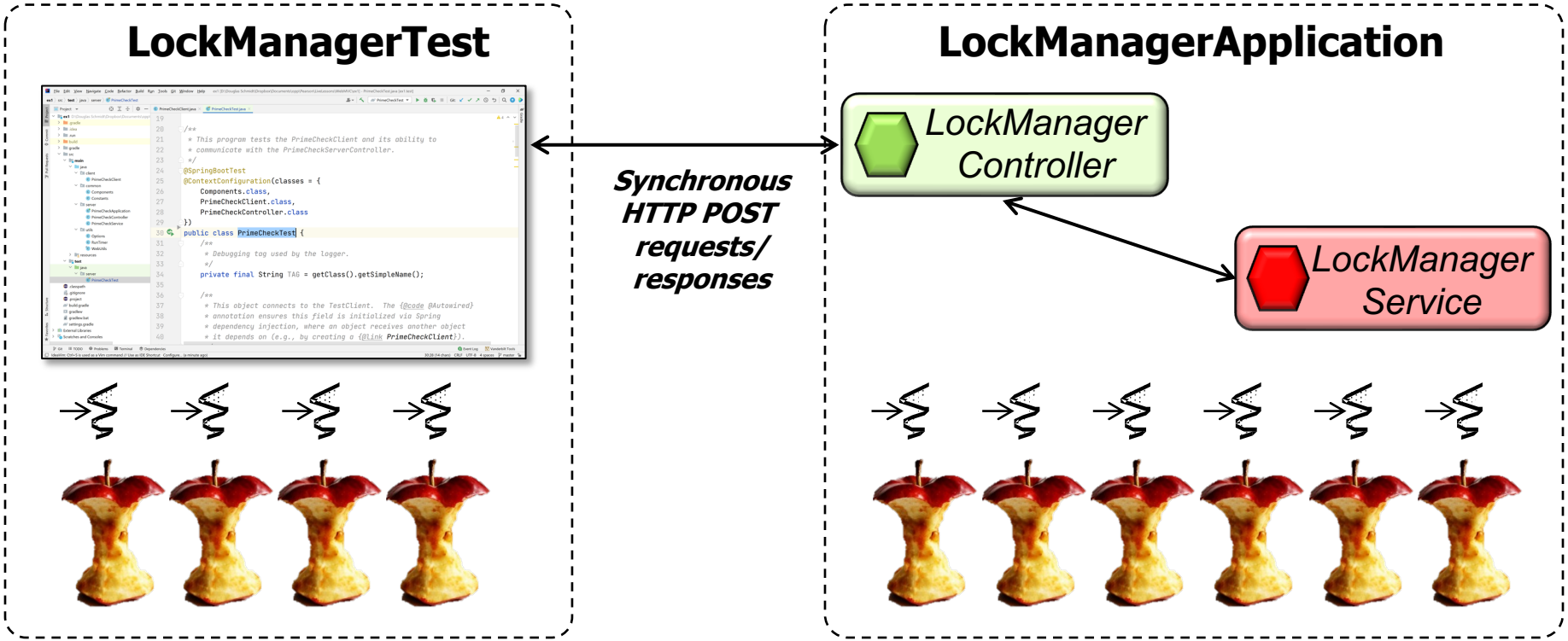
Professor of Computer Science

Institute for Software Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

- Understand how to use Spring WebMVC to send/receive HTTP POST requests synchronously to/from a microservice that provides a distributed semaphore

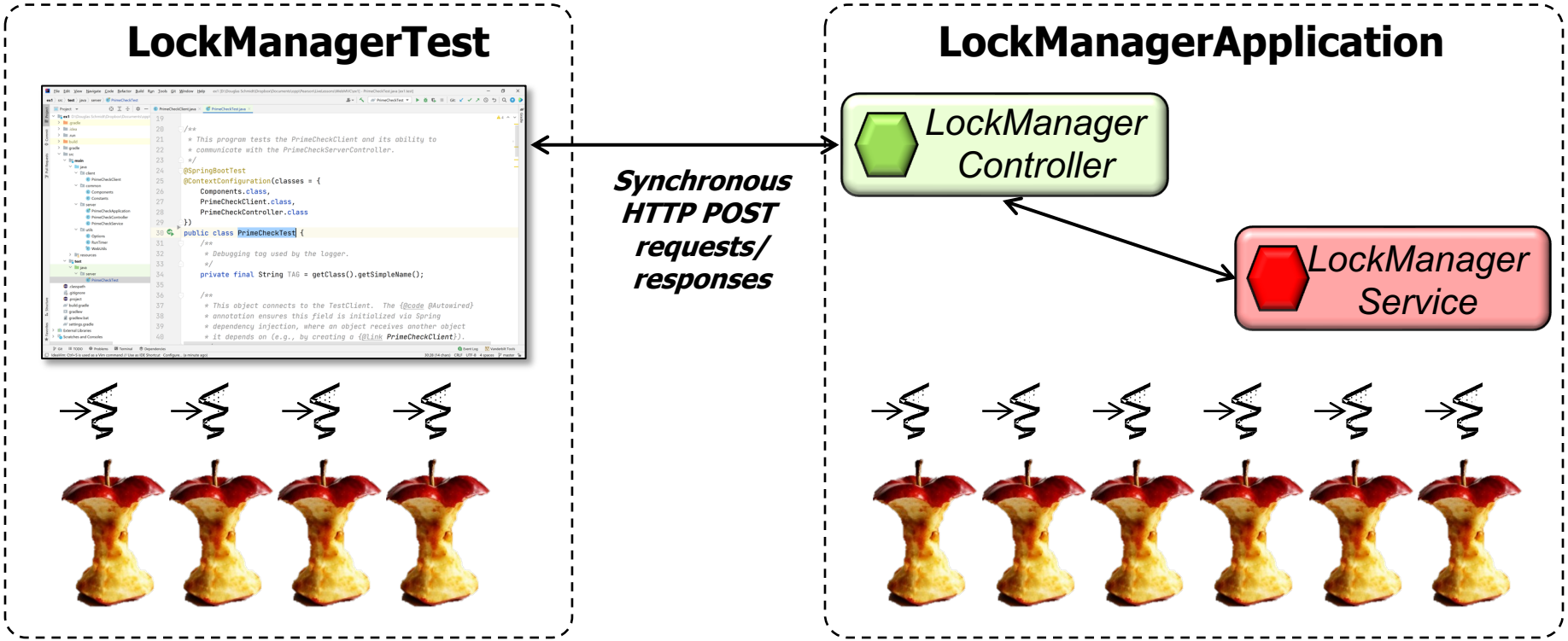


See github.com/douglascraigschmidt/LiveLessons/tree/master/WebMVC/ex5

Overview of the Lock Manager App Case Study

Overview of the LockManager App Case Study

- This case study shows how to use Spring WebMVC to send/receive HTTP POST requests synchronously to/from a LockManager microservice

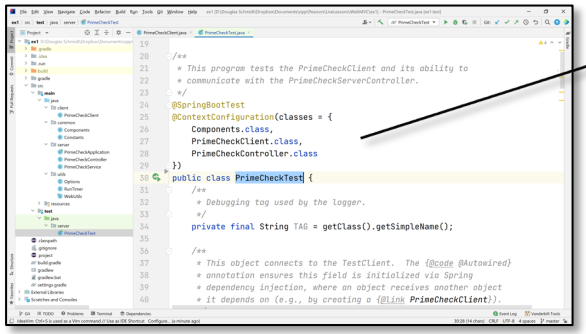


See github.com/douglascraigschmidt/LiveLessons/tree/master/WebMVC/ex5

Overview of the LockManager App Case Study

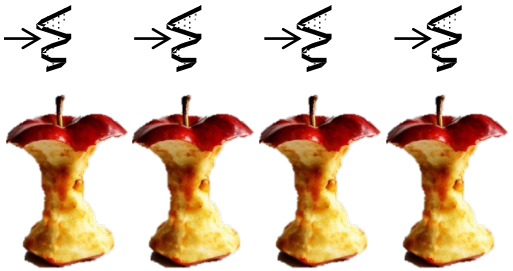
- This case study shows how to use Spring WebMVC to send/receive HTTP POST requests synchronously to/from a LockManager microservice

LockManagerTest

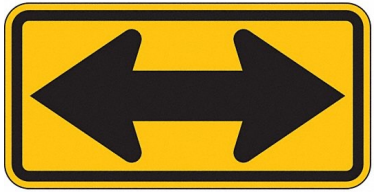


```
/**
 * This program tests the PrimeCheckClient and its ability to
 * communicate with the PrimeCheckServerController.
 */
@SpringBootTest
@ContextConfiguration(classes = {
    Components.class,
    PrimeCheckClient.class,
    PrimeCheckController.class
})
public class PrimeCheckTest {
    /**
     * Debugging tag used by the logger.
     */
    private final String TAG = getClass().getSimpleName();

    /**
     * This object connects to the TestClient.  The {@code @Autowired}
     * annotation ensures this field is initialized via Spring
     * dependency injection, where an object receives another object
     * it depends on (e.g., by creating a {@link PrimeCheckClient}).
```



The client synchronously acquires & releases remotely managed locks individually or in bulk using the declarative LockAPI interface



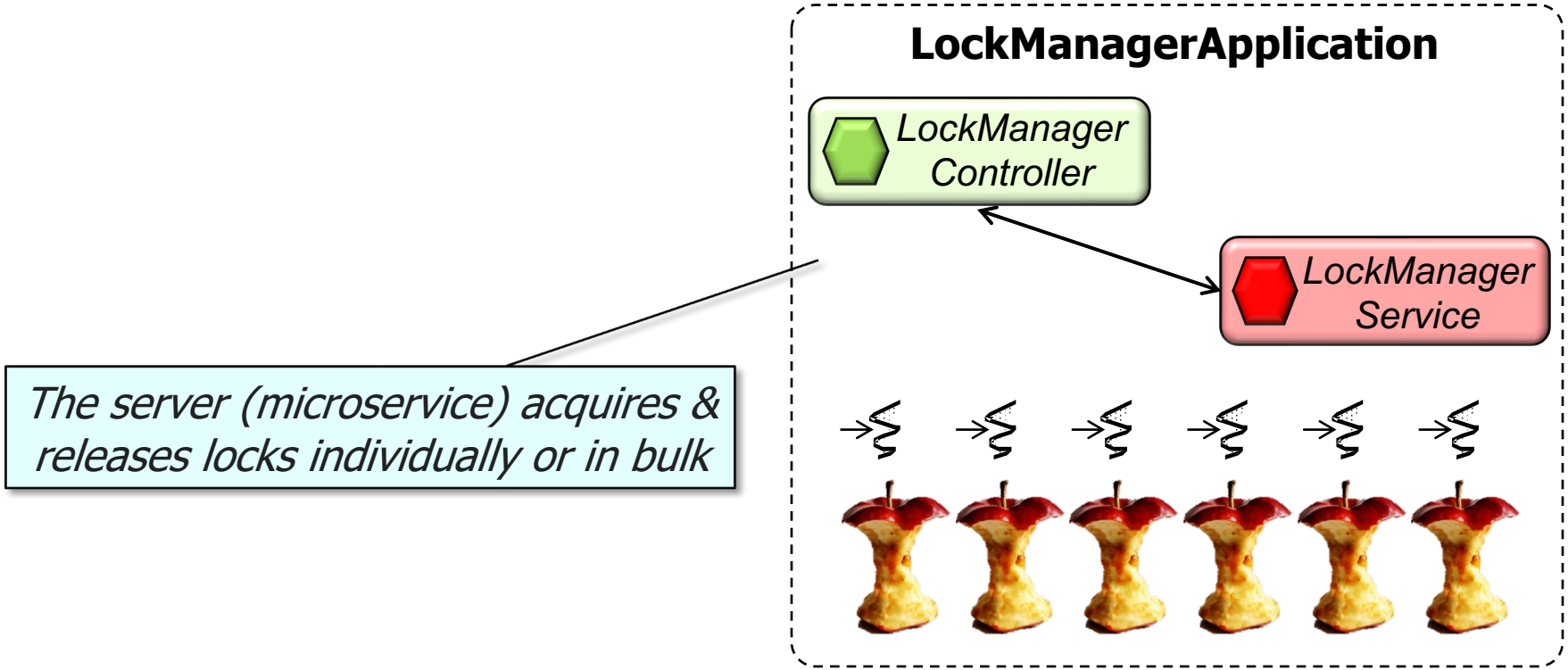
LockAPI

acquire(LockManager)	Lock
acquire(LockManager, Integer)	List<Lock>
create(Integer)	LockManager
release(LockManager, List<Lock>)	Boolean
release(LockManager, Lock)	Boolean

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

Overview of the LockManager App Case Study

- This case study shows how to use Spring WebMVC to send/receive HTTP POST requests synchronously to/from a LockManager microservice

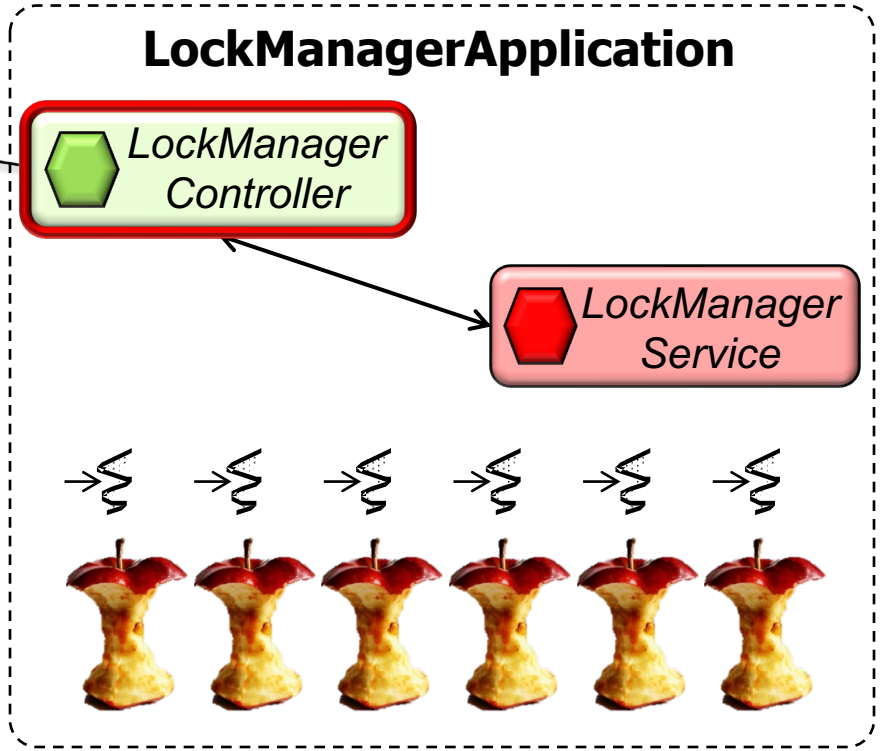


See WebMVC/ex5/src/main/java/edu/vandy/lockmanager/server

Overview of the LockManager App Case Study

- This case study shows how to use Spring WebMVC to send/receive HTTP POST requests synchronously to/from a LockManager microservice

LockManagerController converts all HTTP POST requests into standard or user-defined Java types & forwards them to the LockManagerService



LockManagerController	
mService	LockManagerService
acquire(LockManager)	DeferredResult<Lock>
acquire(LockManager, Integer)	DeferredResult<List<Lock>>
create(Integer)	LockManager
release(LockManager, List<Lock>)	Boolean
release(LockManager, Lock)	Boolean

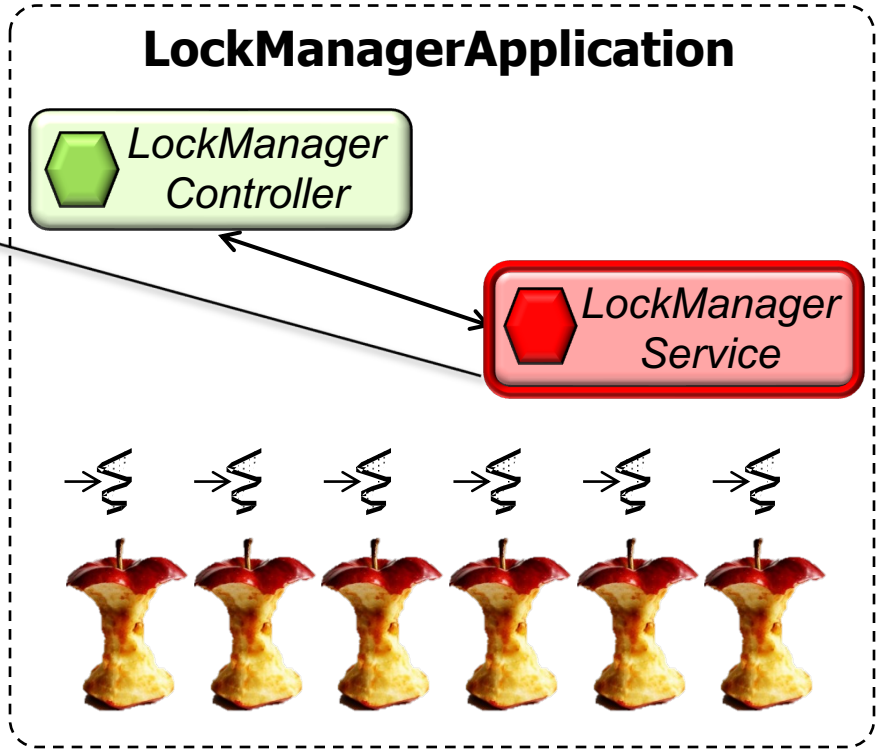
See WebMVC/ex5/src/main/java/edu/vandy/lockmanager/server/LockManagerController.java

Overview of the LockManager App Case Study

- This case study shows how to use Spring WebMVC to send/receive HTTP POST requests synchronously to/from a LockManager microservice

The LockManagerService uses an Array BlockingQueue object & virtual threads to implement a distributed semaphore

LockManagerService	
mExecutor	AsyncTaskExecutor
mLockManagerMap	Map<LockManager, ArrayBlockingQueue<Lock>>
acquire(LockManager, Callback)	void
acquire(LockManager, int)	DeferredResult<List<Lock>>
create(Integer)	LockManager
makeLocks(int)	List<Lock>
release(LockManager, List<Lock>)	Boolean
release(LockManager, Lock)	Boolean
tryAcquireLock(ArrayBlockingQueue<Lock>, List<Lock>)	Integer

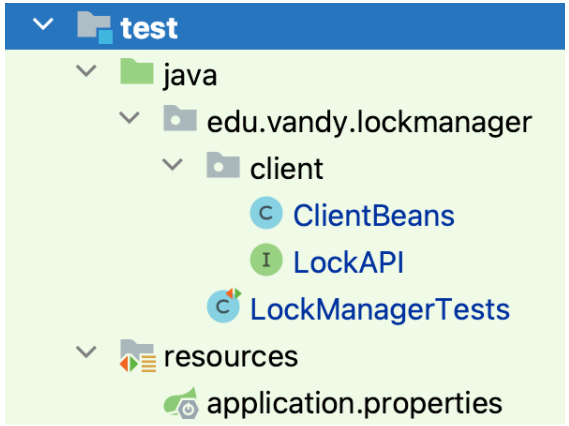


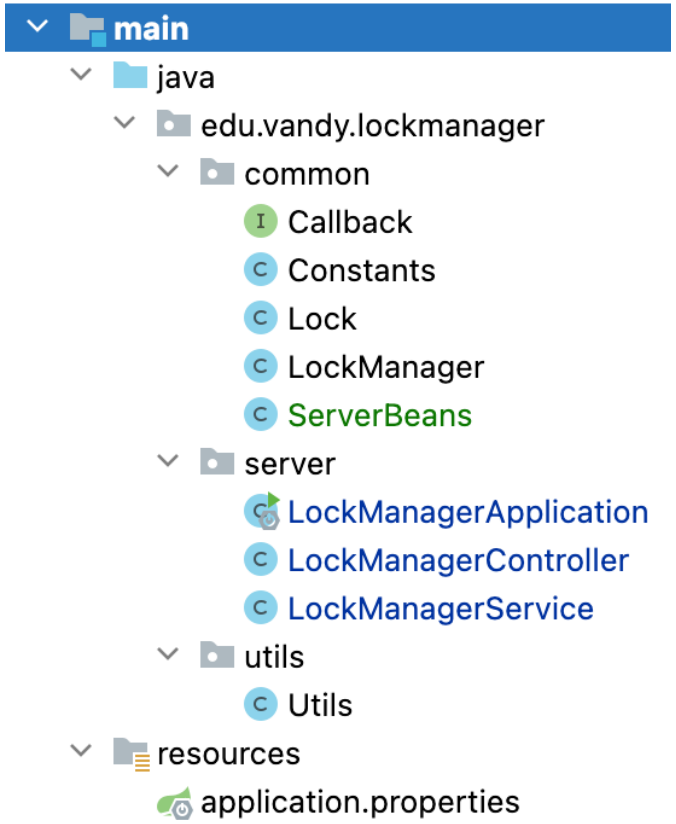
See WebMVC/ex5/src/main/java/edu/vandy/lockmanager/server/LockManagerService.java

Structure of the Lock Manager App Project

Structure of the LockManager App Project

- The LockManager App project source code is organized into several packages

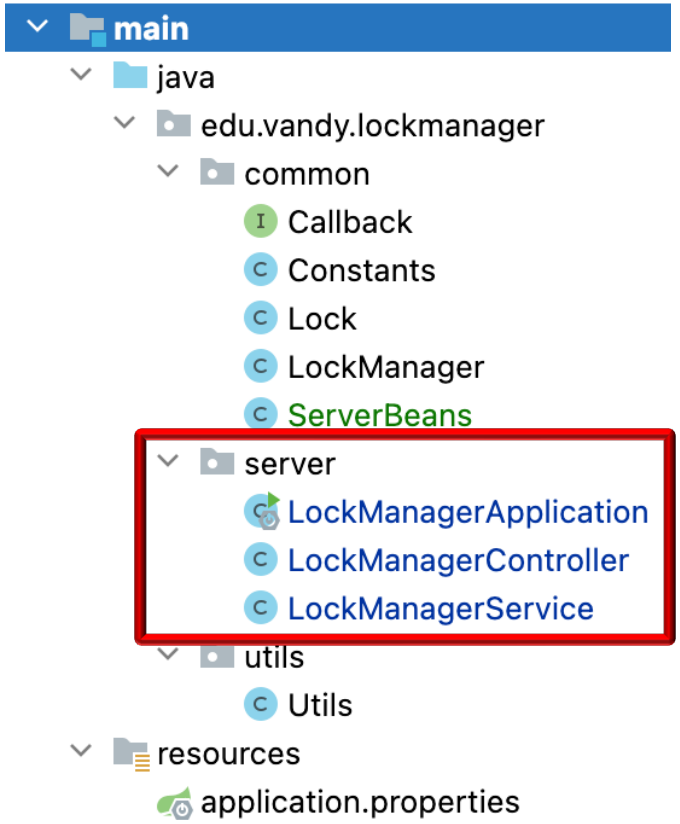




See github.com/douglascraigschmidt/LiveLessons/tree/master/WebMVC/ex5

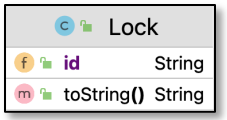
Structure of the LockManager App Project

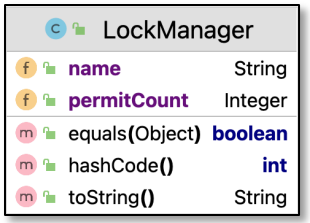
- The LockManager App project source code is organized into several packages
  - main
    - server
      - Contains the "app" entry point, the controller, & the service

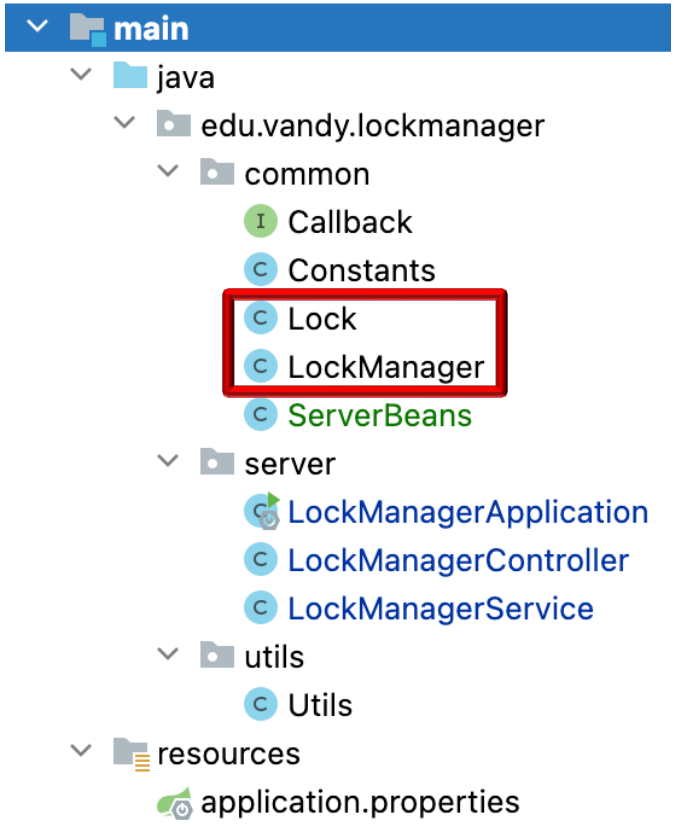


Structure of the LockManager App Project

- The LockManager App project source code is organized into several packages
  - main
    - server
      - Contains the "app" entry point, the controller, & the service
        - This implementation sends/receives a range of standard & user-defined Java types

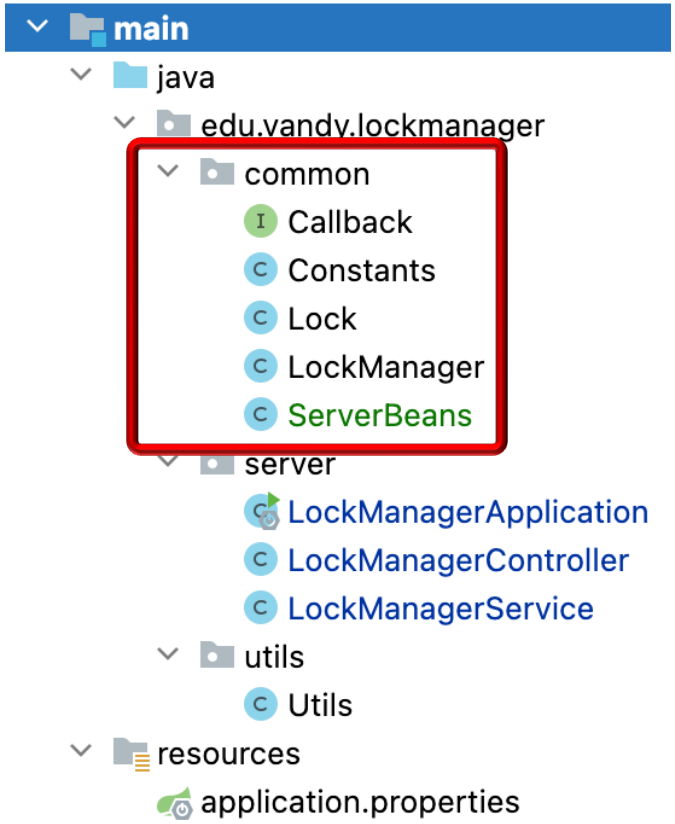






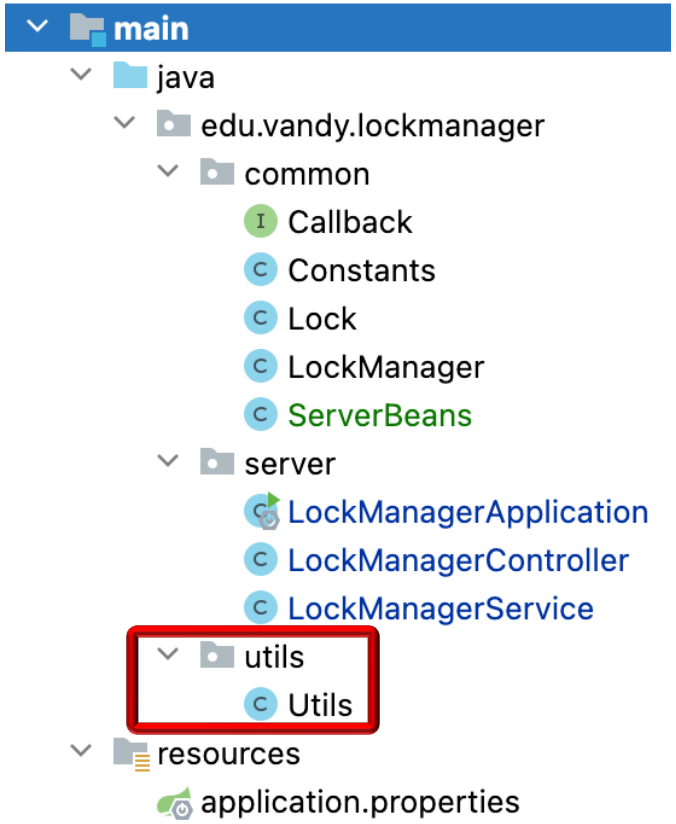
Structure of the LockManager App Project

- The LockManager App project source code is organized into several packages
  - main
    - server
    - common
      - Consolidates various project-specific helper classes
        - e.g., user-defined types like Lock & LockManager passed between client & server



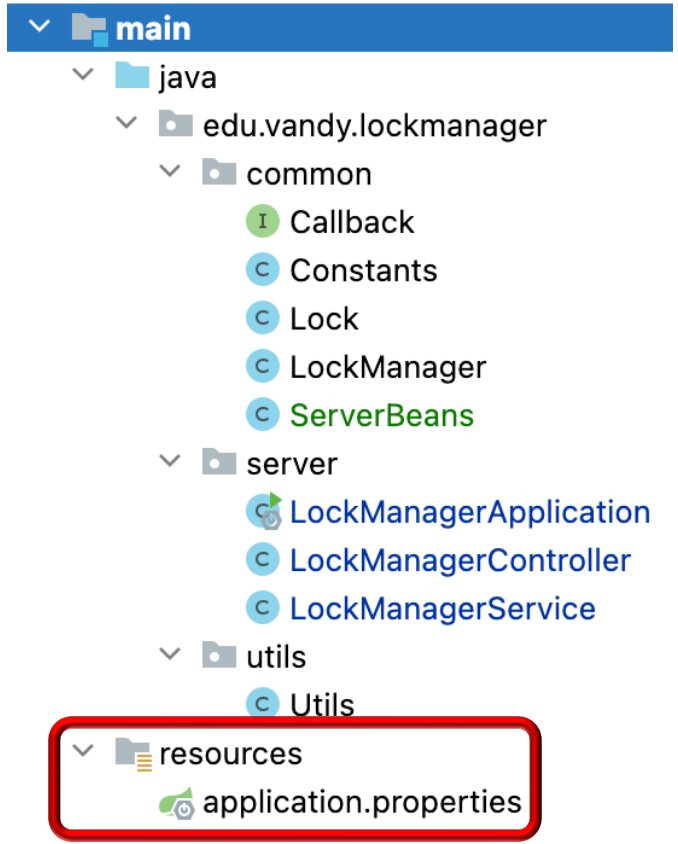
Structure of the LockManager App Project

- The LockManager App project source code is organized into several packages
  - main
    - server
    - common
    - utils
      - General-purpose utilities
        - e.g., log output & generate unique ids



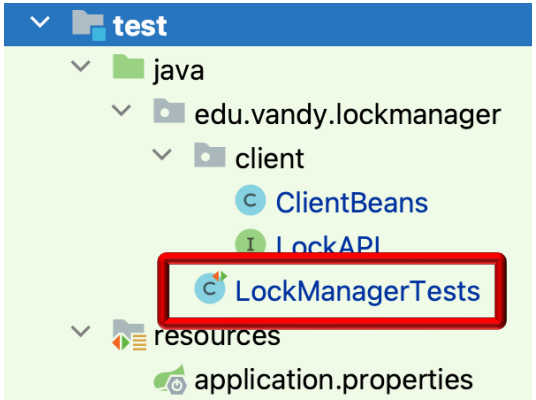
Structure of the LockManager App Project

- The LockManager App project source code is organized into several packages
  - main
    - server
    - common
    - utils
    - resources
      - Defines various application properties
        - e.g., name & port number



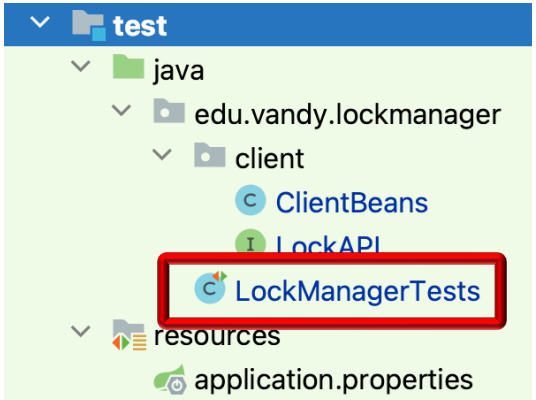
Structure of the LockManager App Project

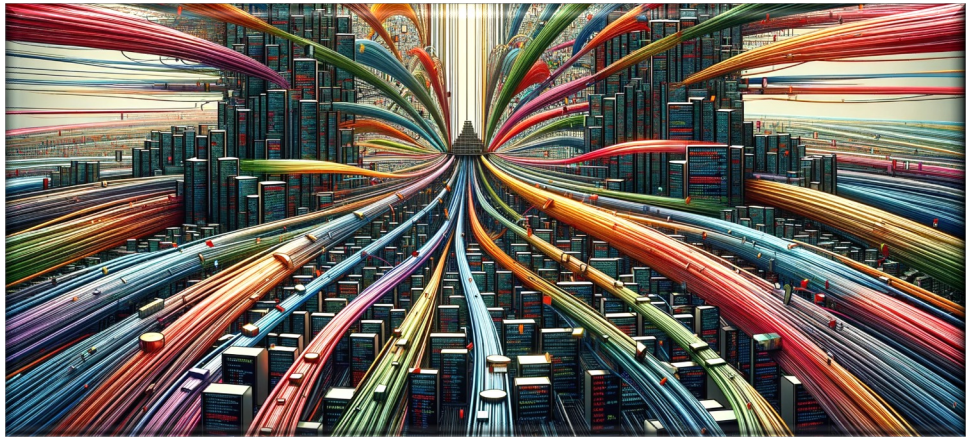
- The LockManager App project source code is organized into several packages
  - test
    - LockManagerTest
      - This test driver initiates synchronous calls to the LockManager microservice



Structure of the LockManager App Project

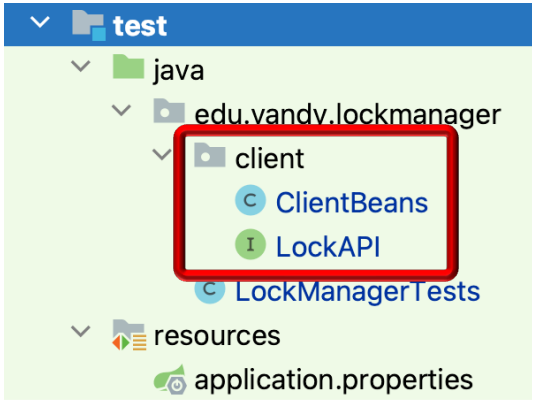
- The LockManager App project source code is organized into several packages
  - test
    - LockManagerTest
      - This test driver initiates synchronous calls to the LockManager microservice
        - Java parallel streams are used to emulate multiple concurrent clients

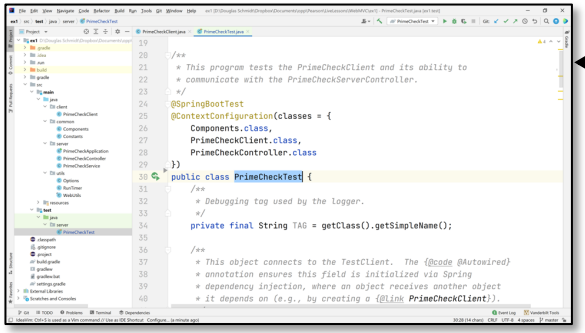


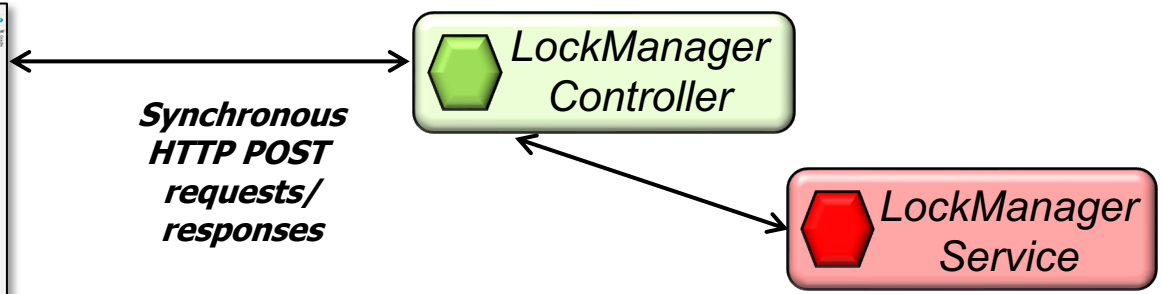


Structure of the LockManager App Project

- The LockManager App project source code is organized into several packages
  - test
    - LockManagerTest
    - client
      - Sends/receives HTTP POST requests to LockManager microservice synchronously

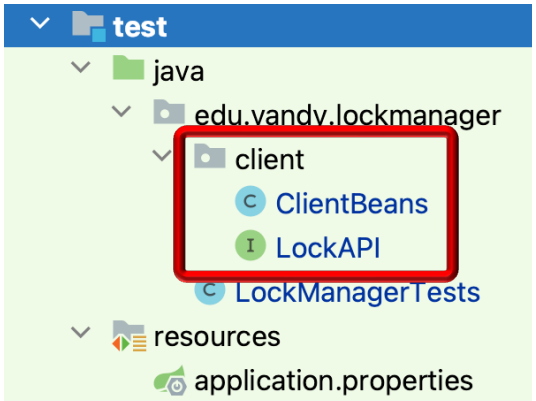


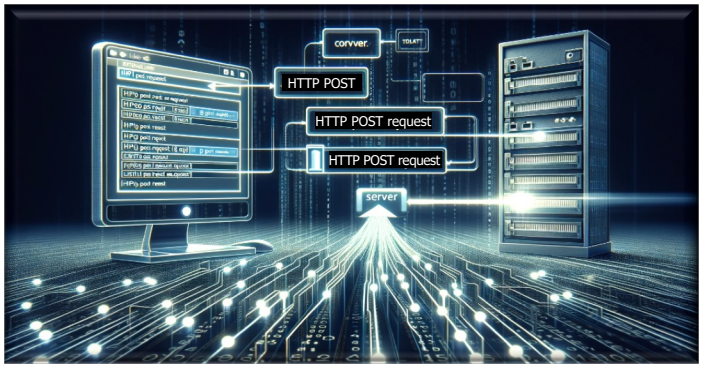




Structure of the LockManager App Project

- The LockManager App project source code is organized into several packages
  - test
    - LockManagerTest
    - client
      - Sends/receives HTTP POST requests to LockManager microservice synchronously
        - HTTP POST requests are used since the server's state is modified

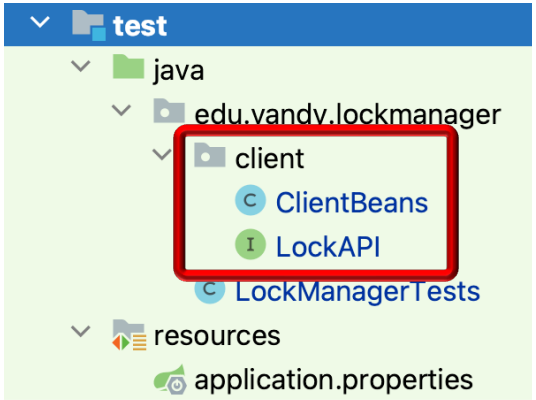




See www.baeldung.com/cs/http-get-vs-post

Structure of the LockManager App Project

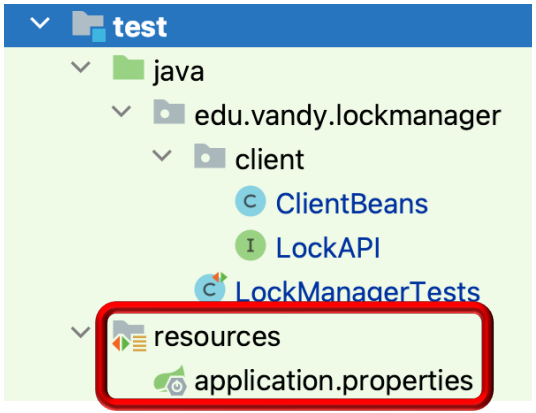
- The LockManager App project source code is organized into several packages
  - test
    - LockManagerTest
    - client
      - Sends/receives HTTP POST requests to LockManager microservice synchronously
        - HTTP POST requests are used since the server's state is modified
      - Declarative HTTP interface features in Spring 6 are also used to automate proxy generation



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

Structure of the LockManager App Project

- The LockManager App project source code is organized into several packages
  - test
    - LockManagerTest
    - client
    - resources
      - Enables/disables verbose Spring logging

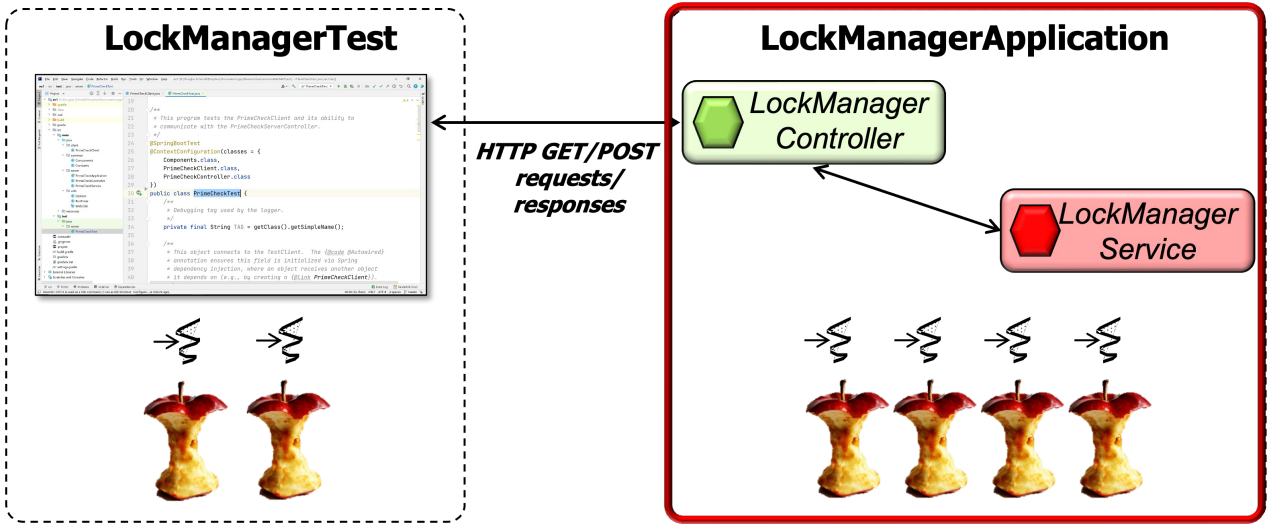



```
logging.level.root=OFF
logging.level.org.springframework.web=OFF
logging.level.org.hibernate=OFF
```

Pros & Cons of the LockManager App

Pros & Cons of the LockManager App

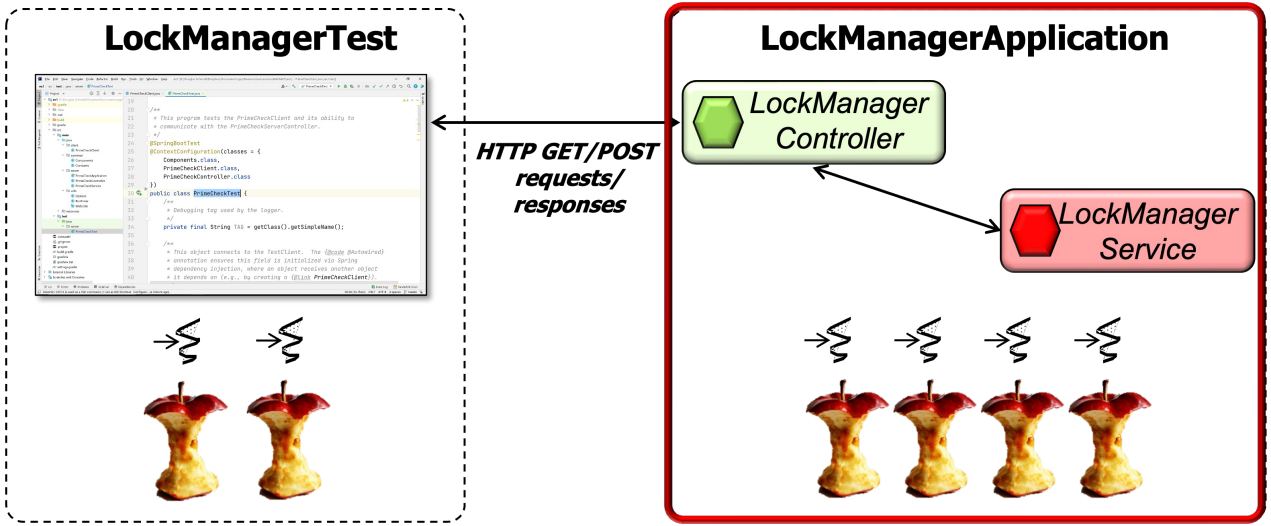
- Pros
  - Spring's DeferredRequest mechanism avoids blocking the servlet thread



Can improve system scalability in traditional (i.e., pre-JDK 19) Java execution environments

Pros & Cons of the LockManager App

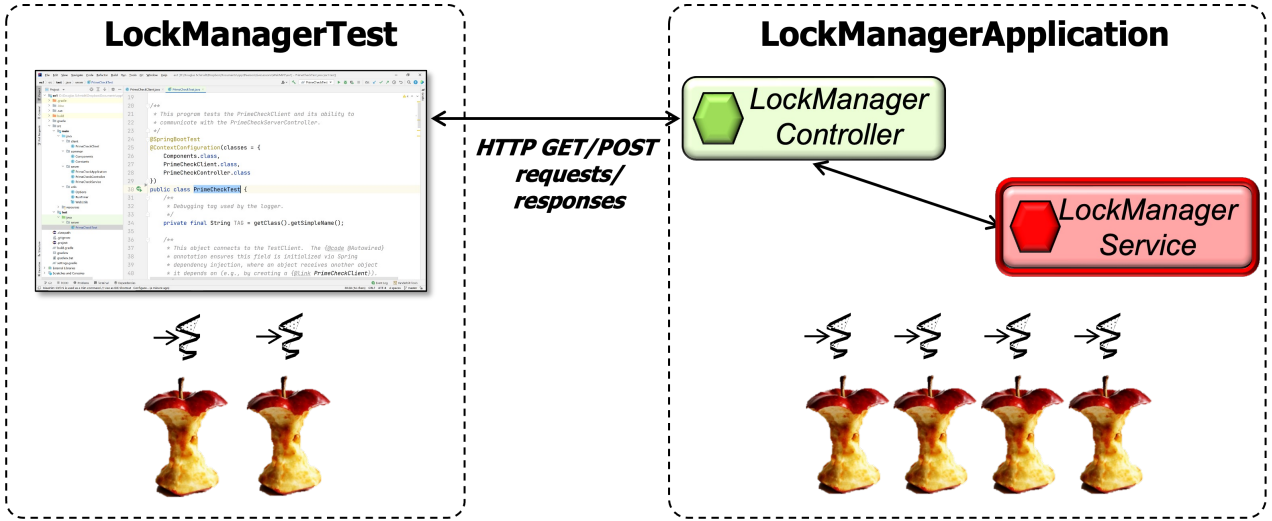
- Pros
  - Spring's DeferredRequest mechanism avoids blocking the servlet thread
    - The "servlet thread" is commonly known as an "HTTP worker thread"



See www.stefankreidel.io/blog/spring-webmvc-servlet-threading

Pros & Cons of the LockManager App

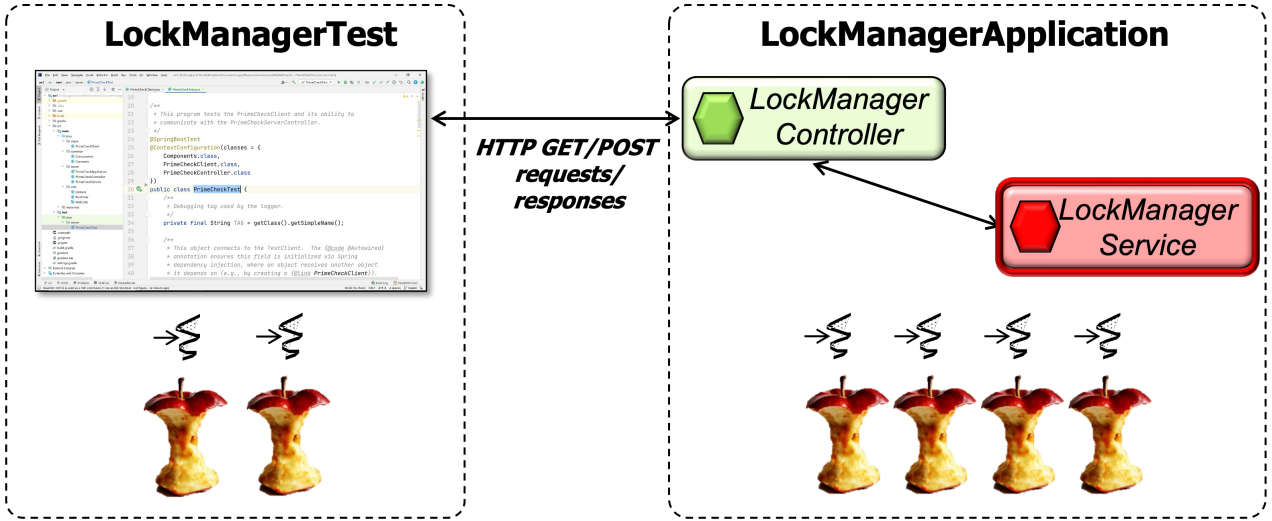
- Pros
  - Spring's DeferredRequest mechanism avoids blocking the servlet thread
  - Clever (largely) lock-free semaphore algorithm avoids knowing synchronizers



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

Pros & Cons of the LockManager App

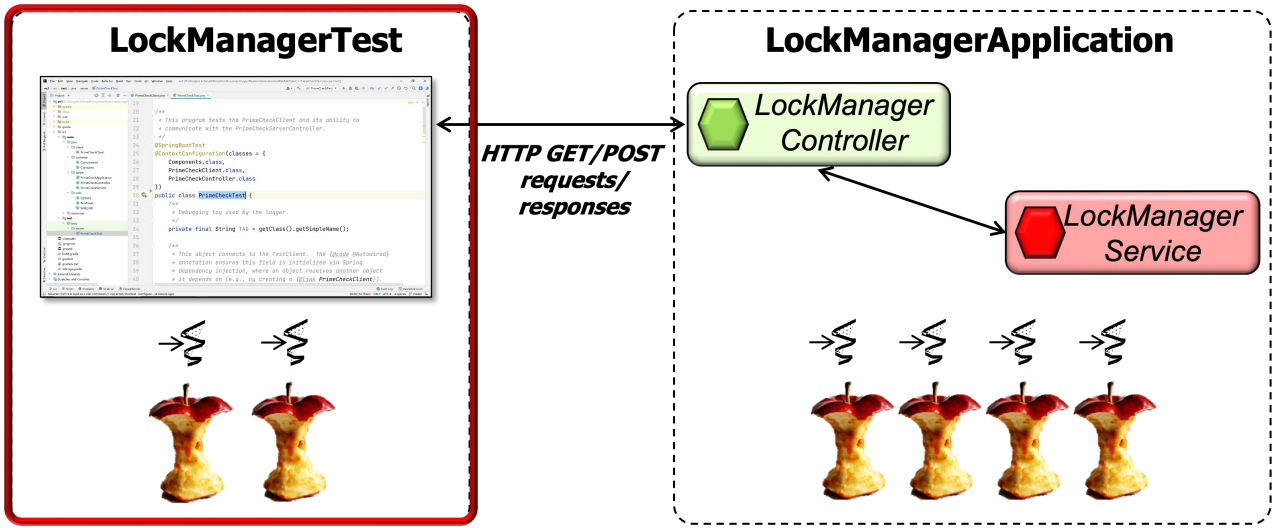
- Pros
  - Spring's DeferredRequest mechanism avoids blocking the servlet thread
  - Clever (largely) lock-free semaphore algorithm avoids knowing synchronizers
    - However, my videos describing Java synchronizers are available online



See www.youtube.com/playlist?list=PLZ9NgFYEMxp59VZ369_XQBD0MRiifJedm

Pros & Cons of the LockManager App

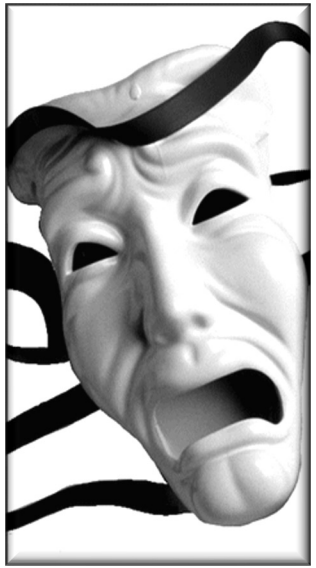
- Pros
  - Spring's DeferredRequest mechanism avoids blocking the servlet thread
  - Clever (largely) lock-free semaphore algorithm avoids knowing synchronizers
  - The client uses declarative Spring 6 HTTP interface synchronous proxies

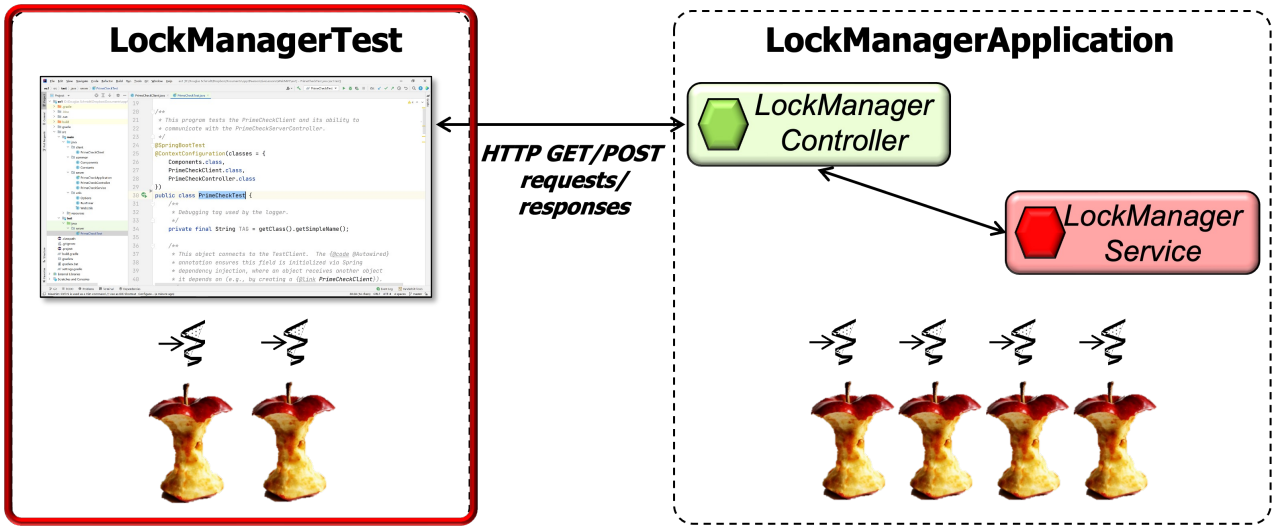


See www.baeldung.com/spring-6-http-interface

Pros & Cons of the LockManager App

- Cons
  - The client isn't actually asynchronous, only the server

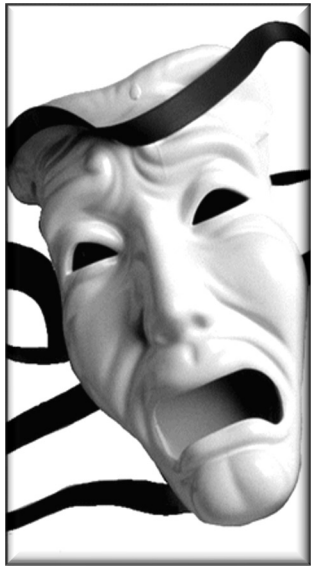


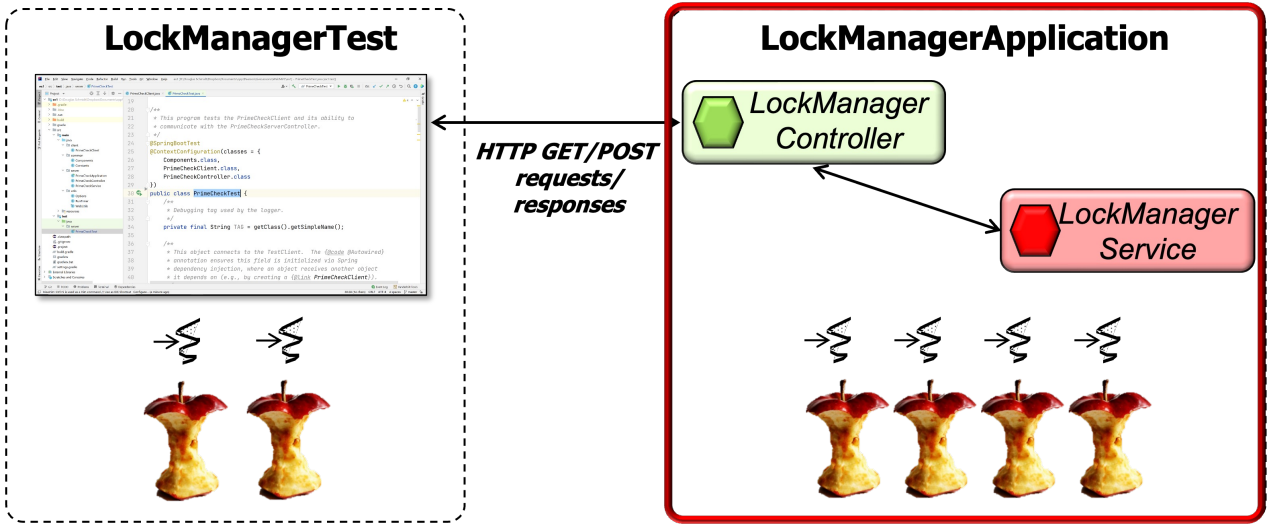


As a result, client threads may block, which can cause timeout problems

Pros & Cons of the LockManager App

- Cons
  - The client isn't actually asynchronous, only the server
  - The server uses the Spring WebMVC thread pool model

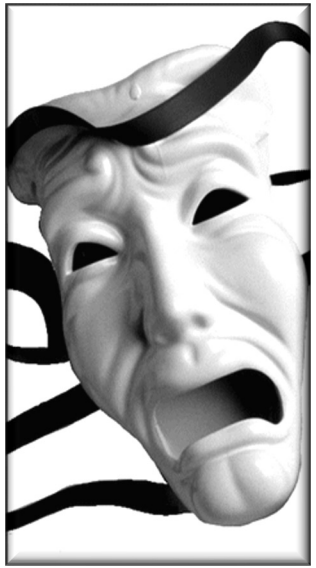


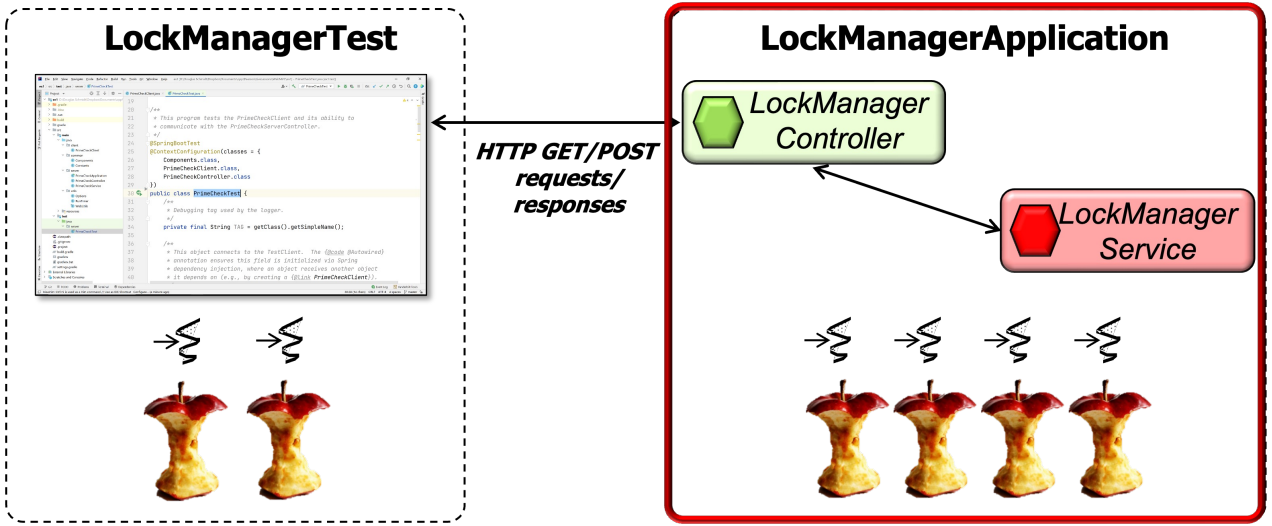


This pool defaults to a fixed number of traditional Java threads

Pros & Cons of the LockManager App

- Cons
  - The client isn't actually asynchronous, only the server
  - The server uses the Spring WebMVC thread pool model
    - This design doesn't take full advantage of Java virtual threads

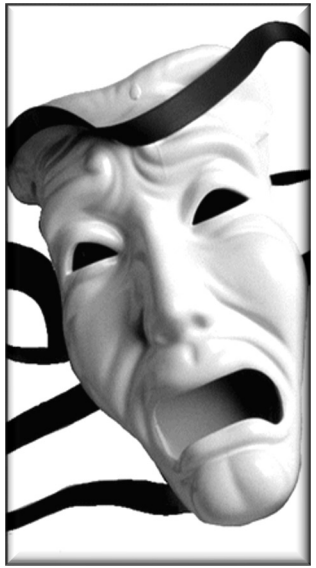


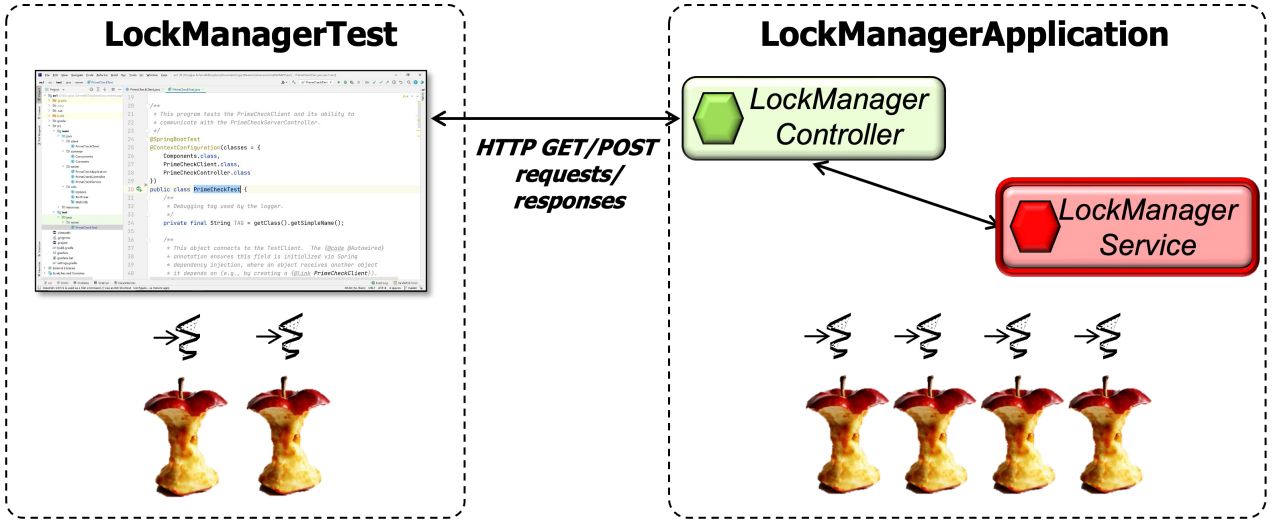


We address this issue in upcoming Spring case studies

Pros & Cons of the LockManager App

- Cons
  - The client isn't actually asynchronous, only the server
  - The server uses the Spring WebMVC thread pool model
  - The ArrayBlockingQueue implementation is not optimal

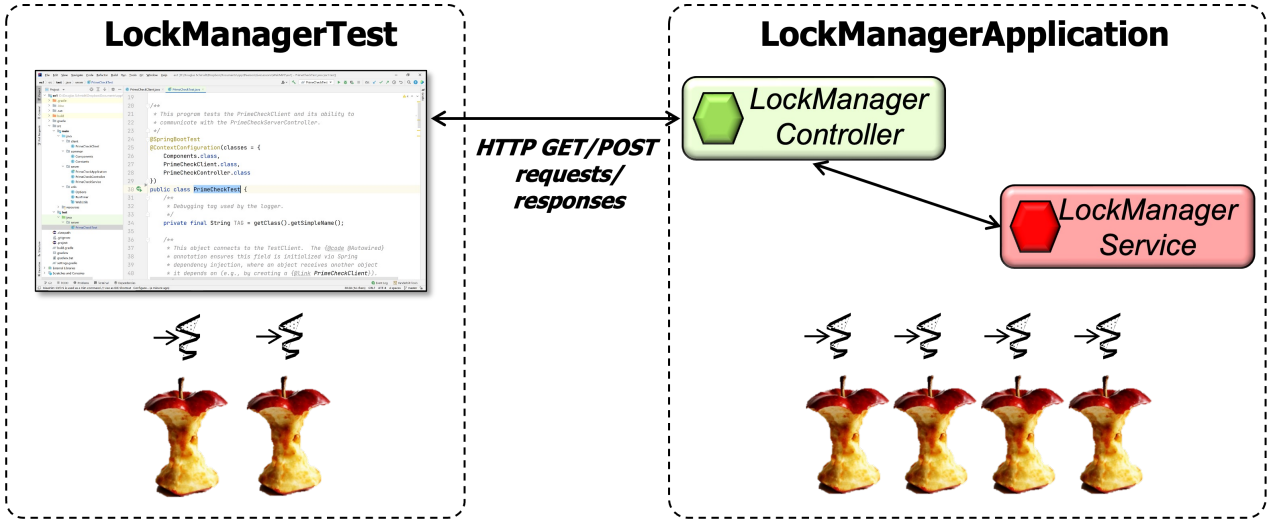




There are far more optimal ways of implementing a distributed semaphore!!

Pros & Cons of the LockManager App

- Cons
  - The client isn't actually asynchronous, only the server
  - The server uses the Spring WebMVC thread pool mode
  - The ArrayBlockingQueue implementation is not optimal



We'll address some limitations later by using WebFlux & Java virtual threads

End of the LockManager App Case Study: Overview