Overview of the RSocket ZippyQuotes App

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

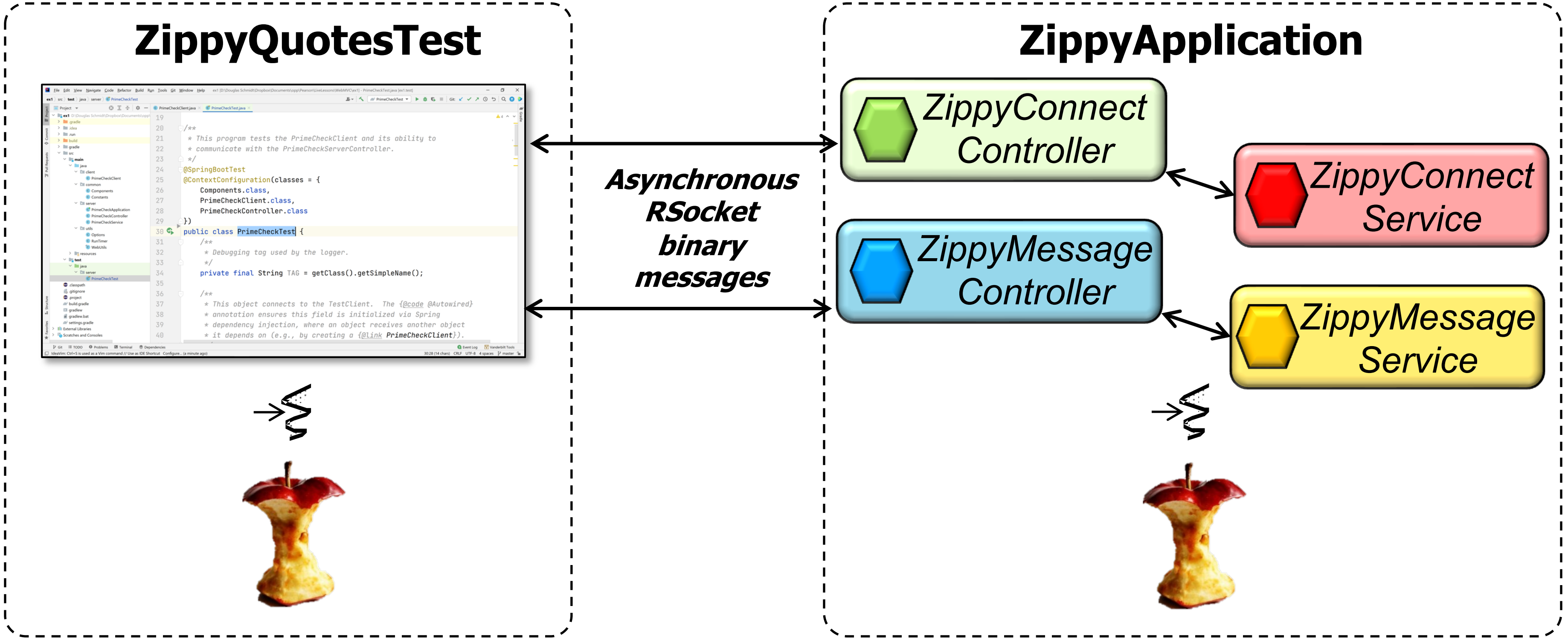
Professor of Computer Science

Institute for Software
Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Lesson

- Understand how RSocket can exchange binary messages asynchronously to/from a Spring microservice using Project Reactor reactive types

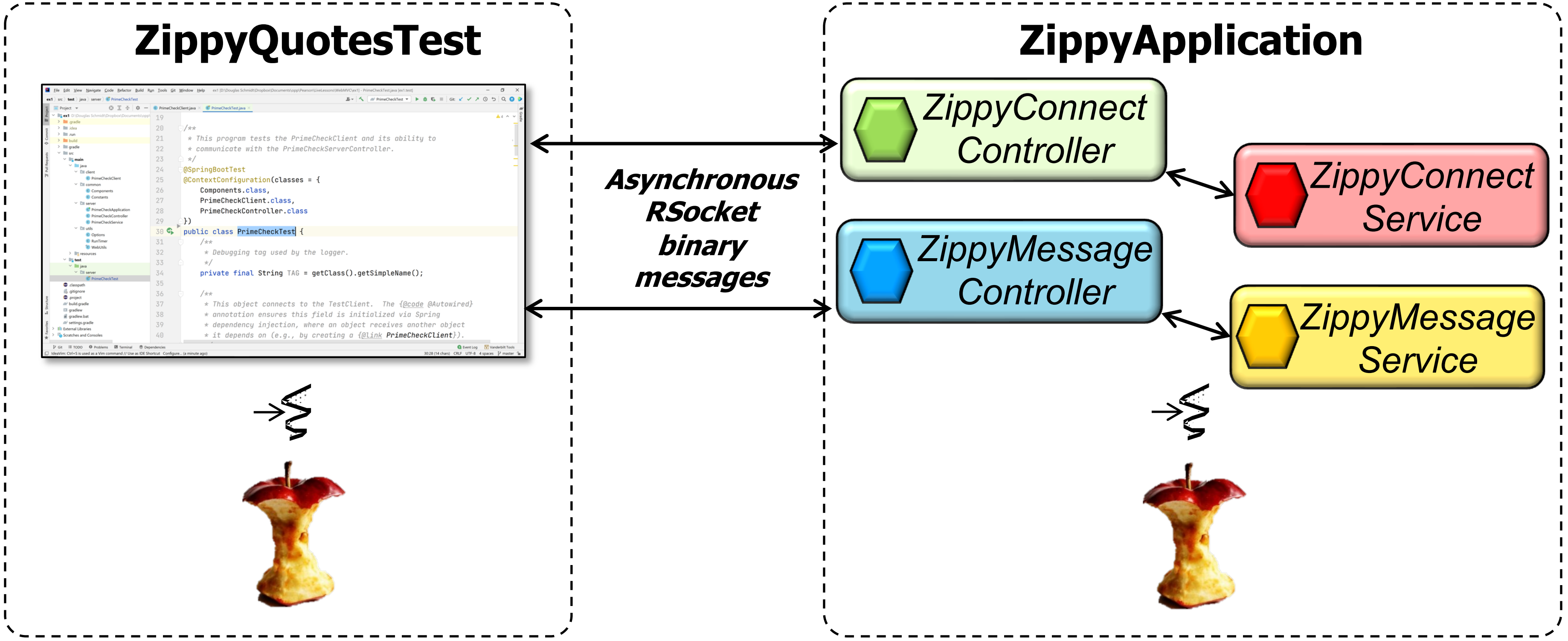


```
/**
 * This program tests the PrimeCheckClient and its ability to
 * communicate with the PrimeCheckServerController.
 */
@SpringBootTest
@ContextConfiguration(classes = {
    Components.class,
    PrimeCheckClient.class,
    PrimeCheckController.class
})
public class PrimeCheckTest {
    /**
     * 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}).
```

See github.com/douglascraigschmidt/LiveLessons/tree/master/RSocket/ex1

Learning Objectives in this Lesson

- Understand how RSocket can exchange binary messages asynchronously to/from a Spring microservice using Project Reactor reactive types



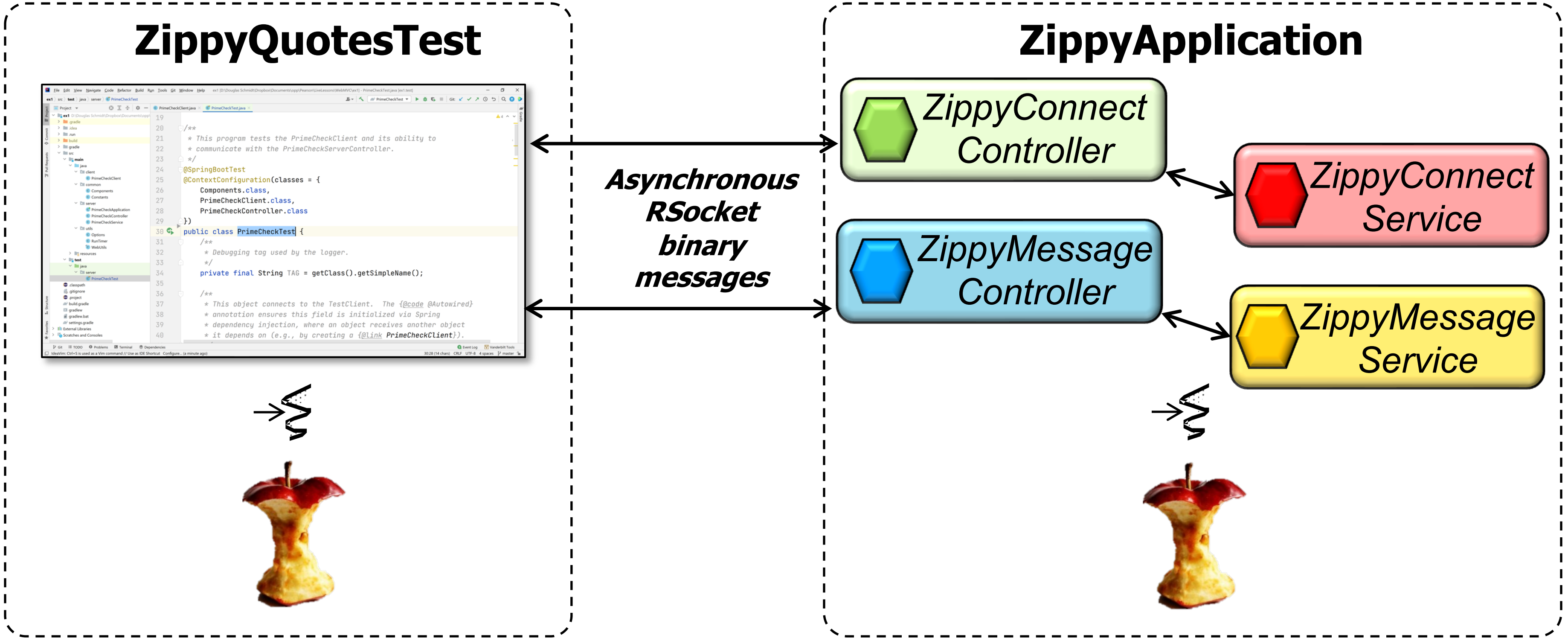
```java
/**
 * This program tests the PrimeCheckClient and its ability to
 * communicate with the PrimeCheckServerController.
 */
@SpringBootTest
@ContextConfiguration(classes = {
    Components.class,
    PrimeCheckClient.class,
    PrimeCheckController.class
})
public class PrimeCheckTest {
    /**
     * 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}).
```

This example also shows how to use RSocket security & authentication features

Overview of the RSocket ZippyQuotes App

Overview of the RSocket ZippyQuotes App

- This case study shows how an RSocket client can exchange binary messages asynchronously with the ZippyApplication microservice



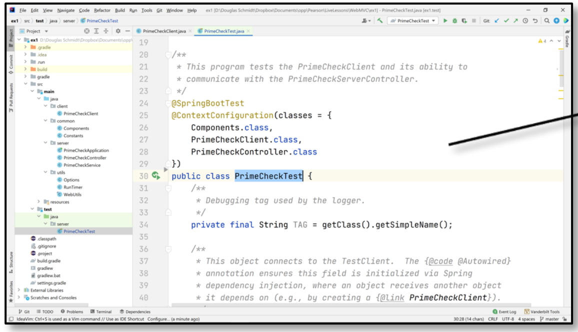
```
/**
 * This program tests the PrimeCheckClient and its ability to
 * communicate with the PrimeCheckServerController.
 */
@SpringBootTest
@ContextConfiguration(classes = {
    Components.class,
    PrimeCheckClient.class,
    PrimeCheckController.class
})
public class PrimeCheckTest {
    /**
     * 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}).
```

See github.com/douglascraigschmidt/LiveLessons/tree/master/RSocket/ex1

Overview of the RSocket ZippyQuotes App

- This case study shows how an RSocket client can exchange binary messages asynchronously with the ZippyApplication microservice

ZippyQuotesTest



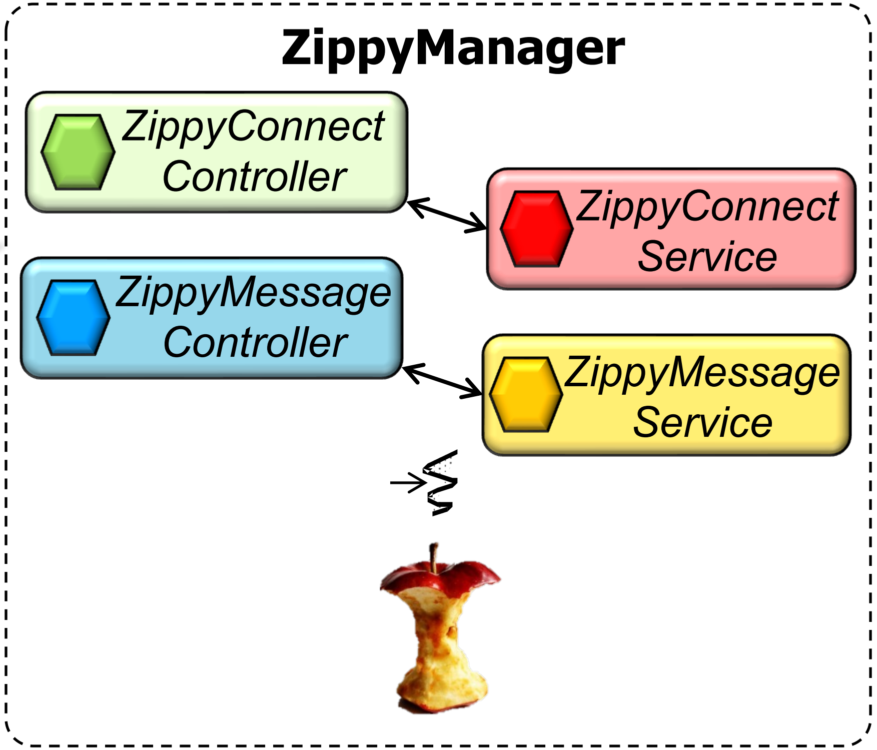
```java
/**
 * This program tests the PrimeCheckClient and its ability to
 * communicate with the PrimeCheckServerController.
 */
@SpringBootTest
@ContextConfiguration(classes = {
    Components.class,
    PrimeCheckClient.class,
    PrimeCheckController.class
})
public class PrimeCheckTest {
    /**
     * 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 can asynchronously connect to the server & send/receive binary messages via the Spring RSocket APIs

The client also authenticates itself to the server

Overview of the RSocket ZippyQuotes App

- This case study shows how an RSocket client can exchange binary messages asynchronously with the ZippyApplication microservice



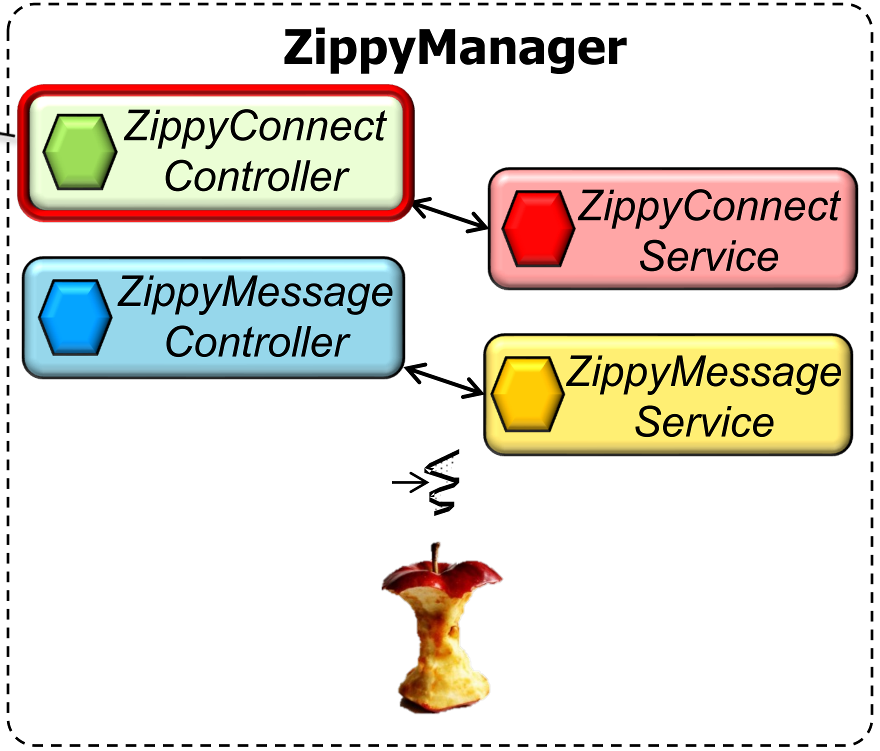
The server (microservice) can receive & reply to connections & binary messages asynchronously

The server also requires the client to authentic itself before finalizing the connection

Overview of the RSocket ZippyQuotes App

- This case study shows how an RSocket client can exchange binary messages asynchronously with the ZippyApplication microservice

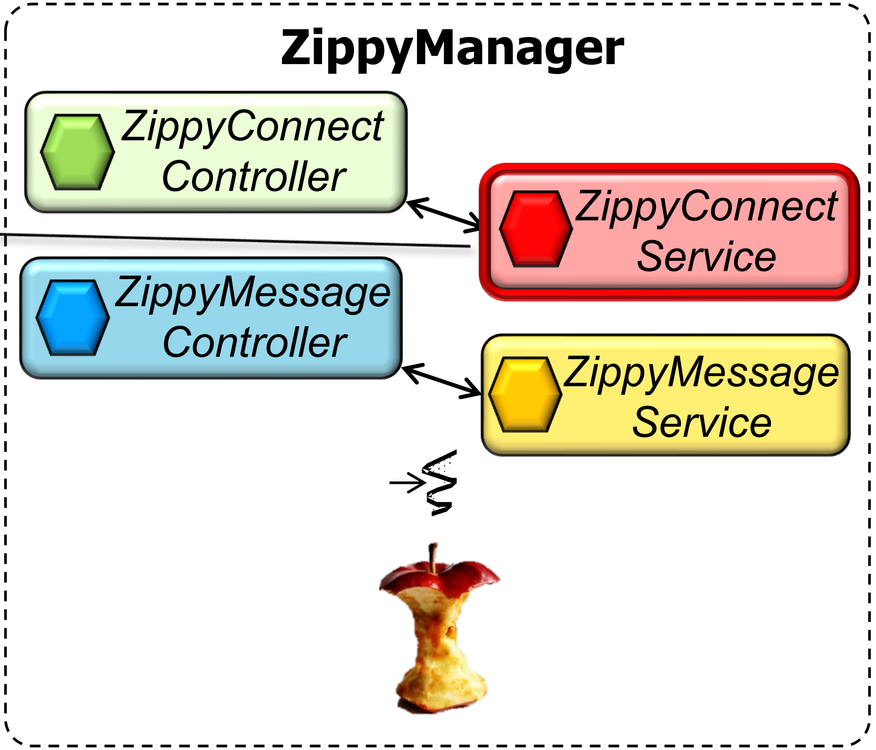
ZippyConnectController receives client connection requests & forwards them to the ZippyConnectService



Overview of the RSocket ZippyQuotes App

- This case study shows how an RSocket client can exchange binary messages asynchronously with the ZippyApplication microservice

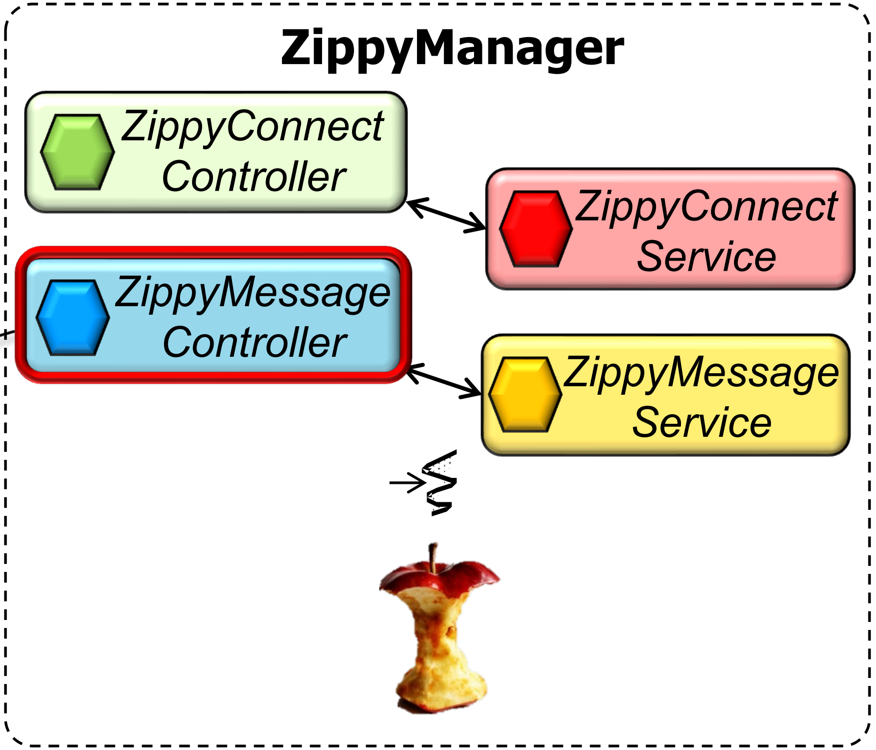
ZippyConnectService performs authentication of client credentials



Overview of the RSocket ZippyQuotes App

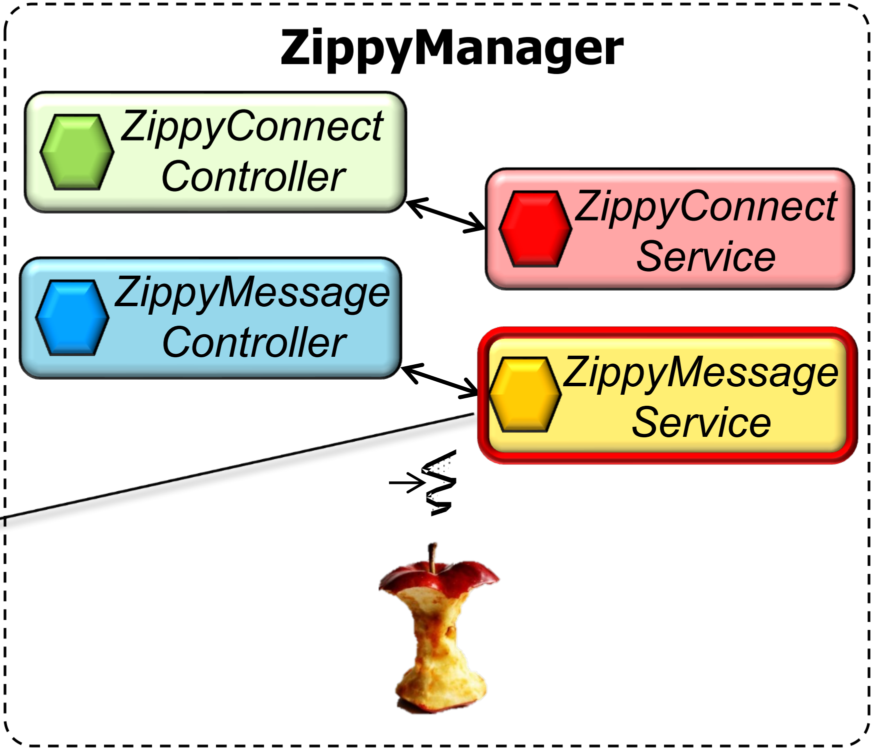
- This case study shows how an RSocket client can exchange binary messages asynchronously with the ZippyApplication microservice

ZippyMessageController converts CBOR-encoded messages into Java types & forwards them to ZippyMessageService



Overview of the RSocket ZippyQuotes App

- This case study shows how an RSocket client can exchange binary messages asynchronously with the ZippyApplication microservice

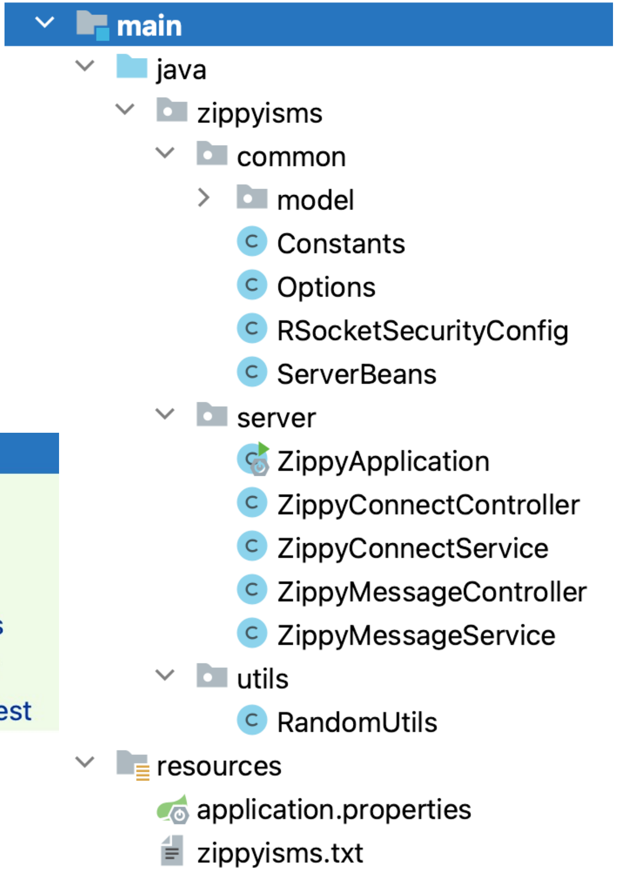


ZippyMessageService implements the methods that process client message requests & return the associated subscriptions & Zippy quotes

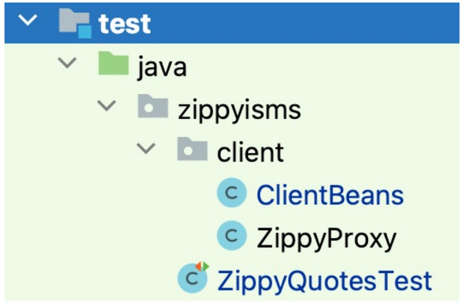
Structure & Functionality of the RSocket ZippyQuotes App

Structure & Functionality of the RSocket ZippyQuotes App

- The RSocket ZippyQuotes App project source code is organized into several packages



```
main
  java
    zippyisms
      common
        model
        Constants
        Options
        RSocketSecurityConfig
        ServerBeans
      server
        ZippyApplication
        ZippyConnectController
        ZippyConnectService
        ZippyMessageController
        ZippyMessageService
      utils
        RandomUtils
  resources
    application.properties
    zippyisms.txt
```

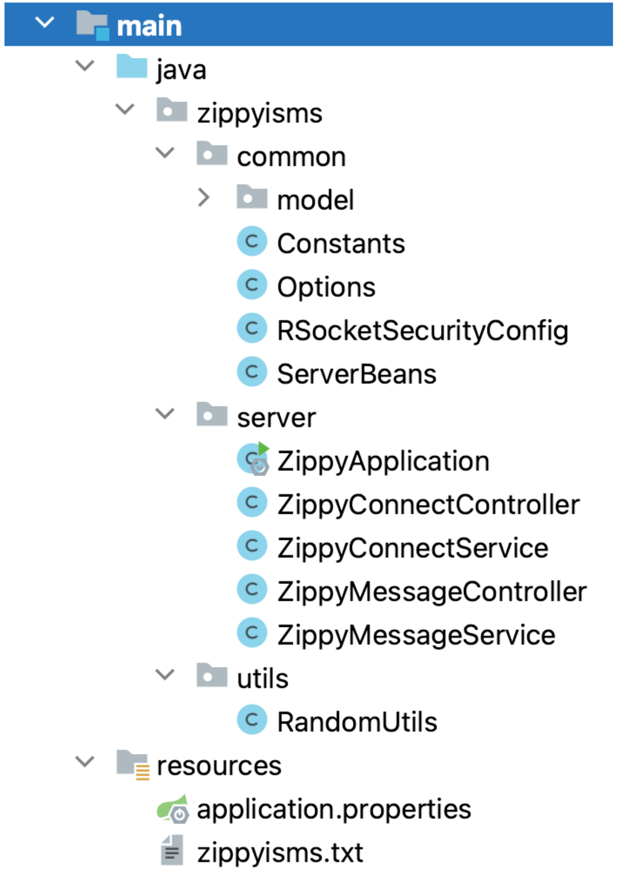


```
test
  java
    zippyisms
      client
        ClientBeans
        ZippyProxy
      ZippyQuotesTest
```

See github.com/douglascraigschmidt/LiveLessons/tree/master/RSocket/ex1

Structure & Functionality of the RSocket ZippyQuotes App

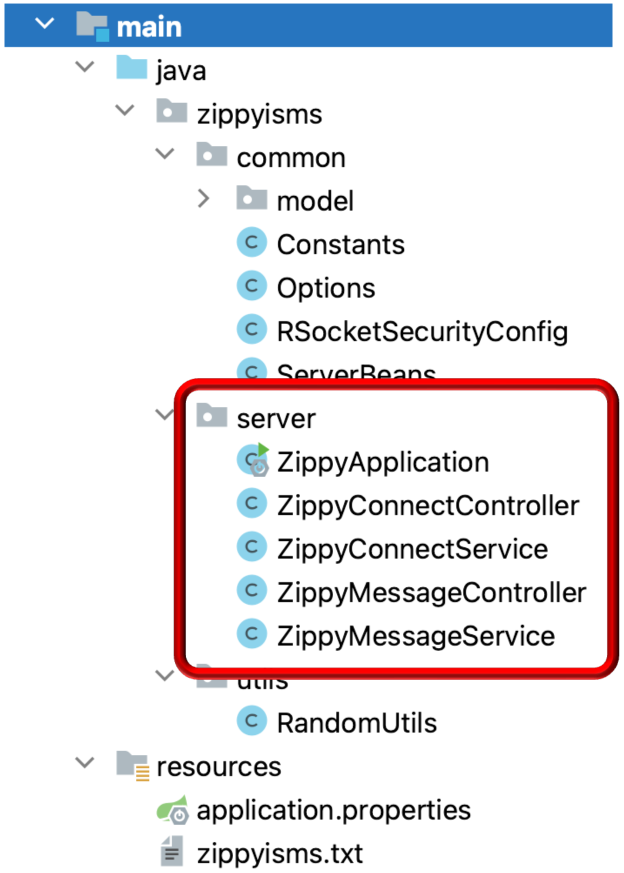
- The RSocket ZippyQuotes App project source code is organized into several packages
  - The src folder contains the server-side classes



```
main
  java
    zippyisms
      common
        model
        Constants
        Options
        RSocketSecurityConfig
        ServerBeans
      server
        ZippyApplication
        ZippyConnectController
        ZippyConnectService
        ZippyMessageController
        ZippyMessageService
      utils
        RandomUtils
  resources
    application.properties
    zippyisms.txt
```

Structure & Functionality of the RSocket ZippyQuotes App

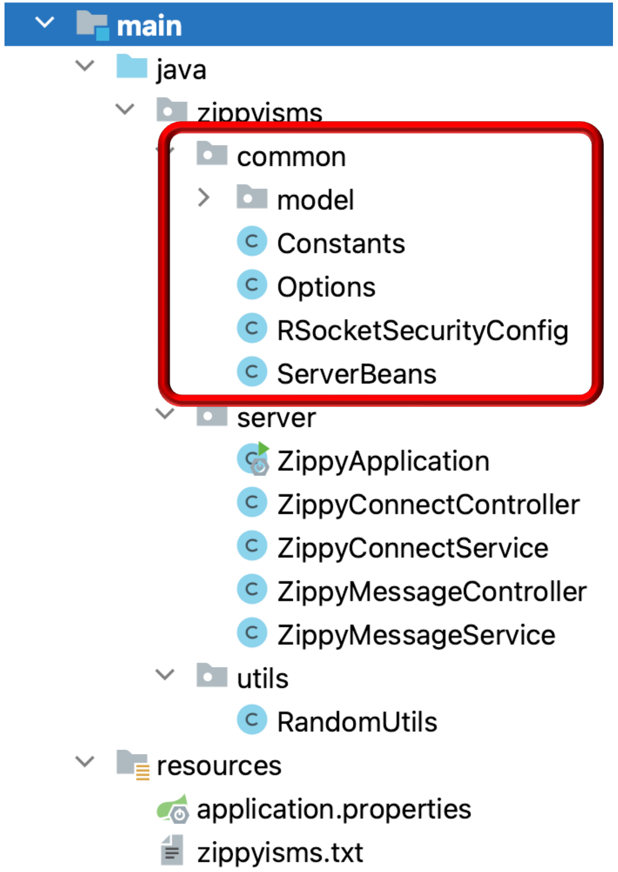
- The RSocket ZippyQuotes App project source code is organized into several packages
  - The src folder contains the server-side classes
    - server
      - The Application, Controller, & Service classes that enable message reception & responses



```
main
  java
    zippyisms
      common
        model
        Constants
        Options
        RSocketSecurityConfig
        ServerBeans
      server
        ZippyApplication
        ZippyConnectController
        ZippyConnectService
        ZippyMessageController
        ZippyMessageService
      utils
        RandomUtils
  resources
    application.properties
    zippyisms.txt
```

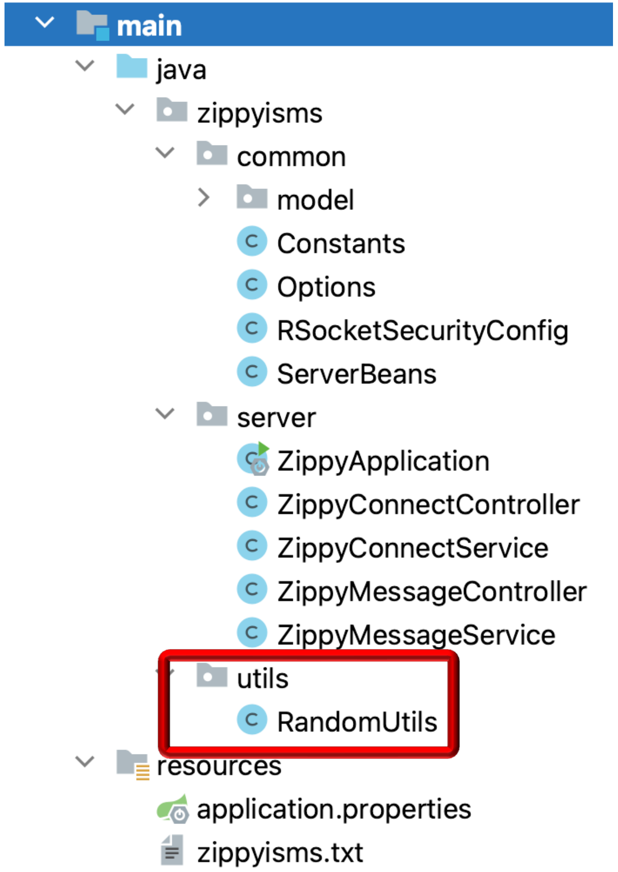
Structure & Functionality of the RSocket ZippyQuotes App

- The RSocket ZippyQuotes App project source code is organized into several packages
  - The src folder contains the server-side classes
    - server
    - common
      - The project-specific reusable classes, including a security module



Structure & Functionality of the RSocket ZippyQuotes App

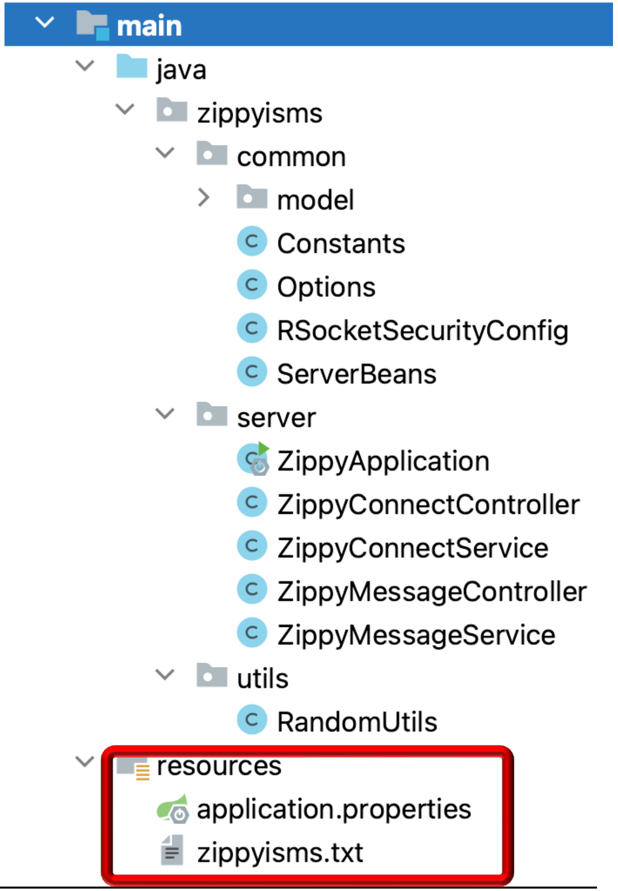
- The RSocket ZippyQuotes App project source code is organized into several packages
  - The src folder contains the server-side classes
    - server
    - common
    - utils
      - The project-independent reusable classes



```
main
  java
    zippyisms
      common
        model
        Constants
        Options
        RSocketSecurityConfig
        ServerBeans
      server
        ZippyApplication
        ZippyConnectController
        ZippyConnectService
        ZippyMessageController
        ZippyMessageService
      utils
        RandomUtils
  resources
    application.properties
    zippyisms.txt
```

Structure & Functionality of the RSocket ZippyQuotes App

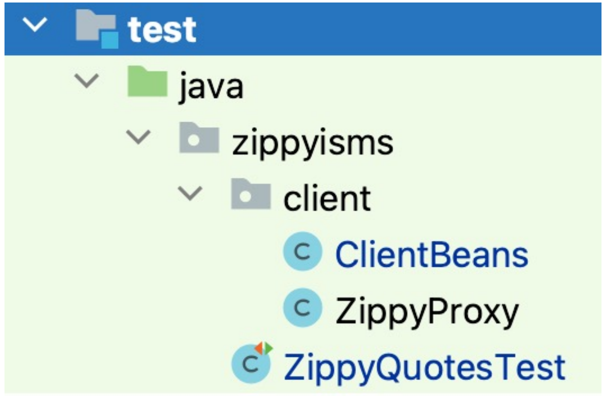
- The RSocket ZippyQuotes App project source code is organized into several packages
  - The src folder contains the server-side classes
    - server
    - common
    - utils
    - resources
      - The server name, port number, & Zippy quotes



```
main
  java
    zippyisms
      common
        model
        Constants
        Options
        RSocketSecurityConfig
        ServerBeans
      server
        ZippyApplication
        ZippyConnectController
        ZippyConnectService
        ZippyMessageController
        ZippyMessageService
      utils
        RandomUtils
  resources
    application.properties
    zippyisms.txt
```

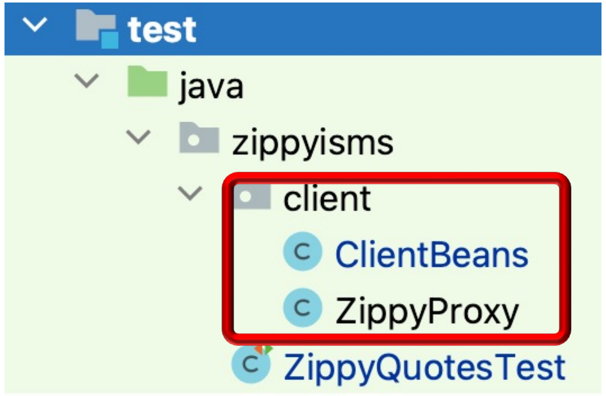
Structure & Functionality of the RSocket ZippyQuotes App

- The RSocket ZippyQuotes App project source code is organized into several packages
  - The src folder contains the server-side classes
  - The test folder contains the client-side classes



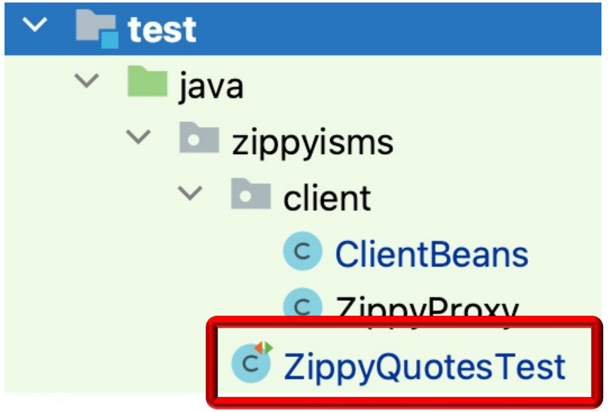
Structure & Functionality of the RSocket ZippyQuotes App

- The RSocket ZippyQuotes App project source code is organized into several packages
  - The src folder contains the server-side classes
  - The test folder contains the client-side classes
    - client
      - The proxies & client bean that establishes a secure connection with the server



Structure & Functionality of the RSocket ZippyQuotes App

- The RSocket ZippyQuotes App project source code is organized into several packages
  - The src folder contains the server-side classes
  - The test folder contains the client-side classes
    - client
      - The proxies & client beans
      - The test driver that connects with the server & exchanges messages



End of Overview of the RSocket ZippyQuotes App