

Leveraging Dependency Injection



Billy Korando

SENIOR SOFTWARE CONSULTANT - KEYHOLE SOFTWARE

@korandoBilly

Agenda

Take a closer look at unit testing

Review the different kinds of mocks

Brief overview of the demo project

Learn about Test Driven Development

Demo using TDD and Mocks

What is a Unit Test?

What is a Unit Test?

```
public Order createOrder(List<Items> items, String cusId, String ccNum){
    validateItems(items);

    RestTemplate rest = new RestTemplate();
    Customer customer = rest.get(customerUrl + cusId);
    Payment payment = rest.get(paymentUrl + ccNum);

    Order order = new Order();
    order.setItems(items);
    order.setCustomer(customer);
    order.setPayment(payment);

    String sql =
        "INSERT INTO ORDER (ORDER_ID, CUST_ID, PAYMENT_ID) VALUES (?, ?, ?)";

    jdbcTemplate = new JdbcTemplate(dataSource);
    jdbcTemplate.update(sql, new Object[] { order.getOrderId(), cust.getId(),
        payment.getId() });

    return order;
}
```

What is a Unit Test?

```
@Test
public void throwsExceptionWhenCustomerNotFound() {

    OrderService service ....

    try{
        service.createOrder(null, "BAD_CID", null);
        fail("An Exception should have been thrown");
    } catch (AServiceException e){
    }

}
```

Not a unit test

Reason: not ran in isolation

What is a Unit Test?

```
public class Order{  
    String orderNumber;  
  
    public void setOrderNumber(String orderNumber){  
        this.orderNumber = orderNumber;  
    }  
  
    public String getOrderNumber(){  
        return orderNumber;  
    }  
}
```

What is a Unit Test?

```
@Test  
public void testSetOrderNumber()
```

```
    Order order = new Order();  
    order.setOrderNumber("1234");
```

```
    assertEquals("234", order.getOrderNumber());
```

```
}
```

Not a unit test
Reason: not verifying
business behavior

What is a Unit Test?

```
public Order createOrder(List<Items> items, String cusId, String ccNum){  
  
    itemService.validateItems(items);  
    Customer customer = customerService.findCustomer(cusId);  
    Payment payment = paymentService.createPayment(ccNum);  
  
    Order order = new Order();  
    order.setItems(items);  
    order.setCustomer(customer);  
    order.setPayment(payment);  
  
    orderDao.insertOrder(order);  
  
    return order;  
}
```


What is a Unit Test?

```
@Test
public void testCreateOrder(){

    OrderService service = new OrderService(itemServiceDummy, customerMock,
    paymentMock, orderMock);

    try{
        service.createOrder(null, "BAD_CUS_ID", null);
        fail("An Exception should had been thrown");
    } catch (ServiceException e){
        assertEquals("Customer id: BAD_CUS_ID not found", e.getMessage());
    }

    Order order = service.createOrder(testItemList(), "1234", "1234");
    assertNotNull(order);
}
```

Not a unit test


Reason: verifying multiple scenarios

What is a Unit Test?

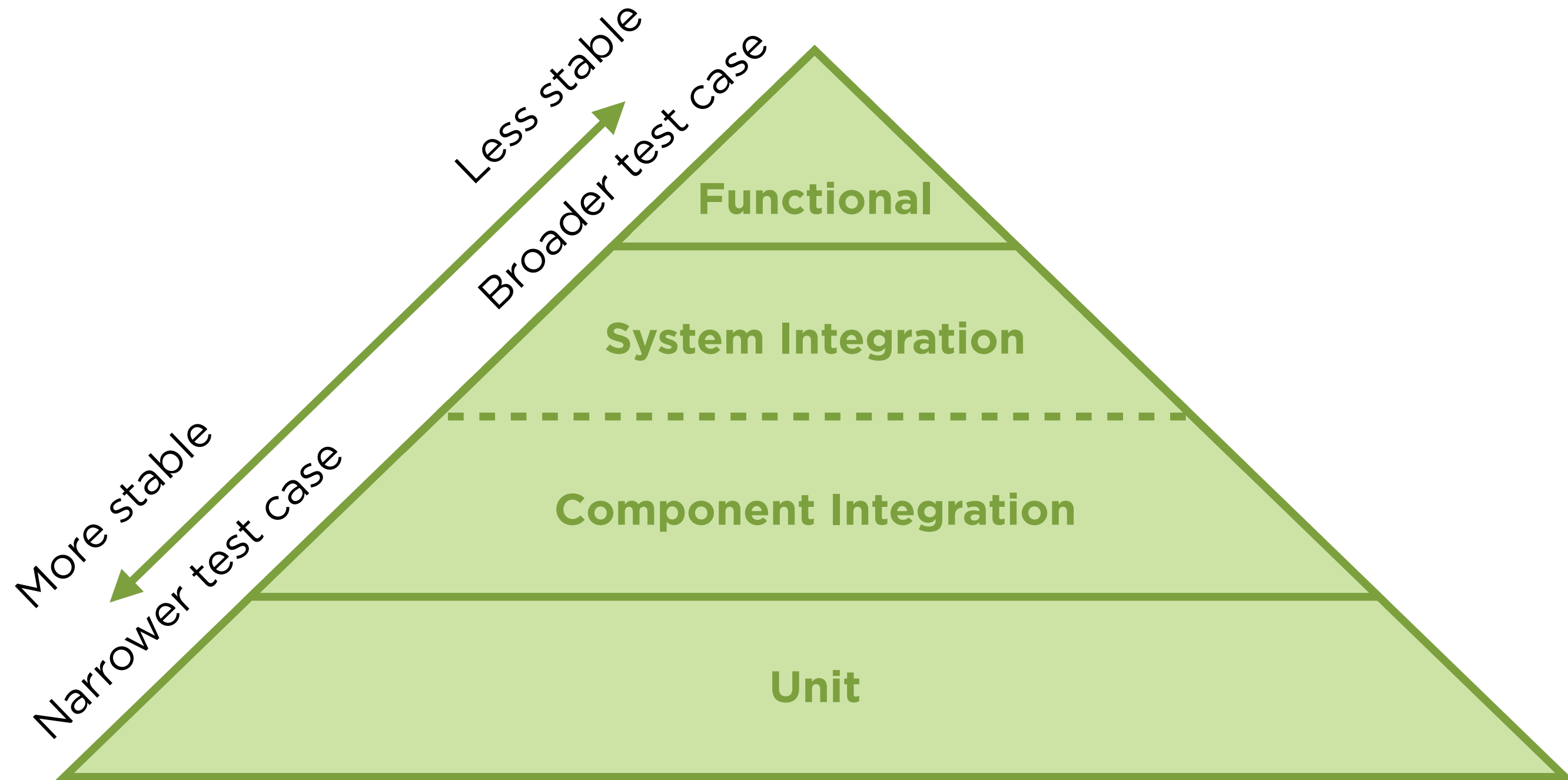
```
@Test
public void testCreateOrder(){

    OrderService service = new OrderService(it, my, customerMock,
    paymentMock, orderDaoMock);

    try{
        service.createOrder(1, 1);
        fail("An Exception should have been thrown");
    } catch (AServiceException e){
        assertEquals("Customer Id: 1 not found!", e.getMessage());
    }
}
```



Pyramid of Testing



Key Characteristics of Unit Tests



Ran in isolation

Verify business behavior

Verify a single scenario

A Deeper Look at Mocks

Types of Mocks

Dummy

Used when a dependency's behavior is not checked.

Stub

Used when a dependency needs to fulfill a condition.

Spy

Used to verify a dependency has been called.

Mock

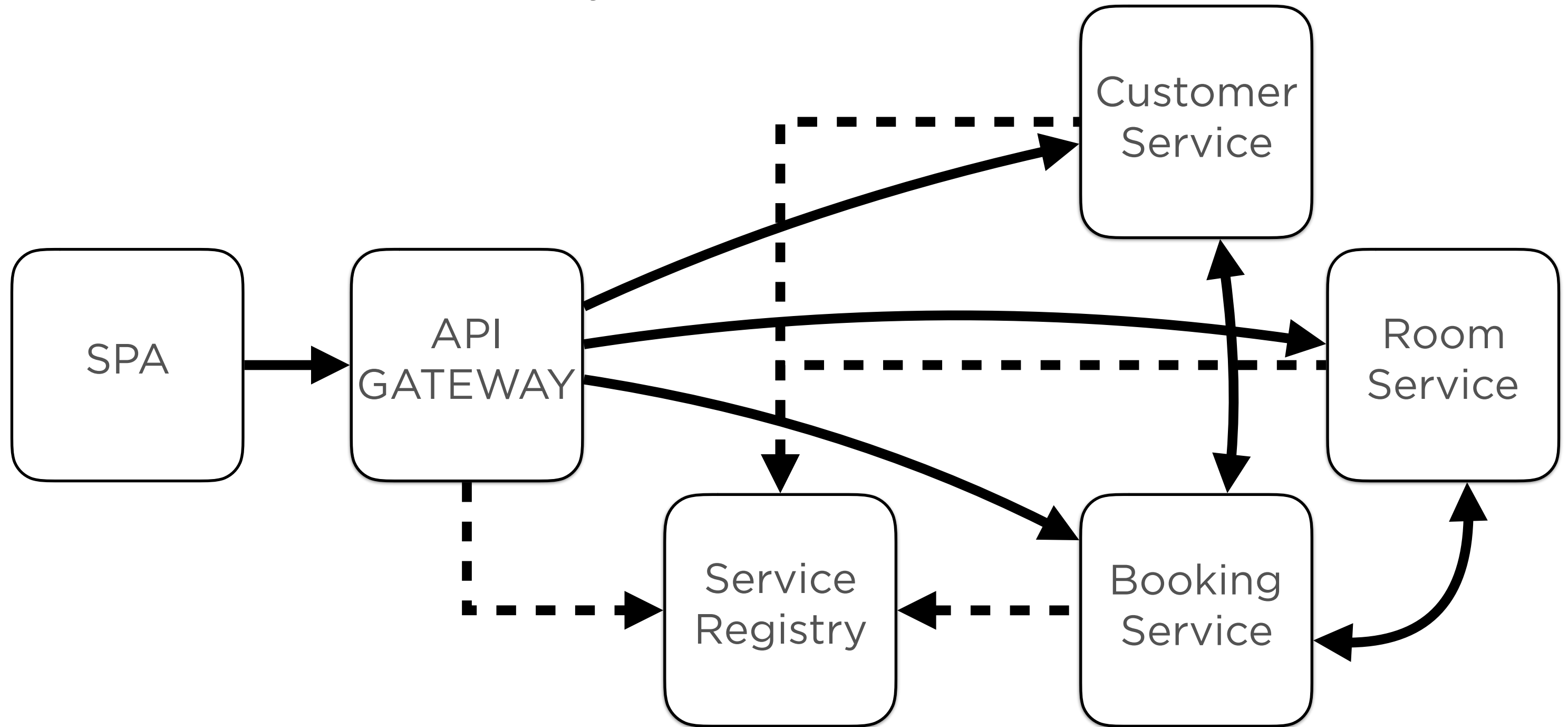
Used to verify the interaction with a dependency.

Fake

Used to simulate business behavior.

Project Overview

Project Overview

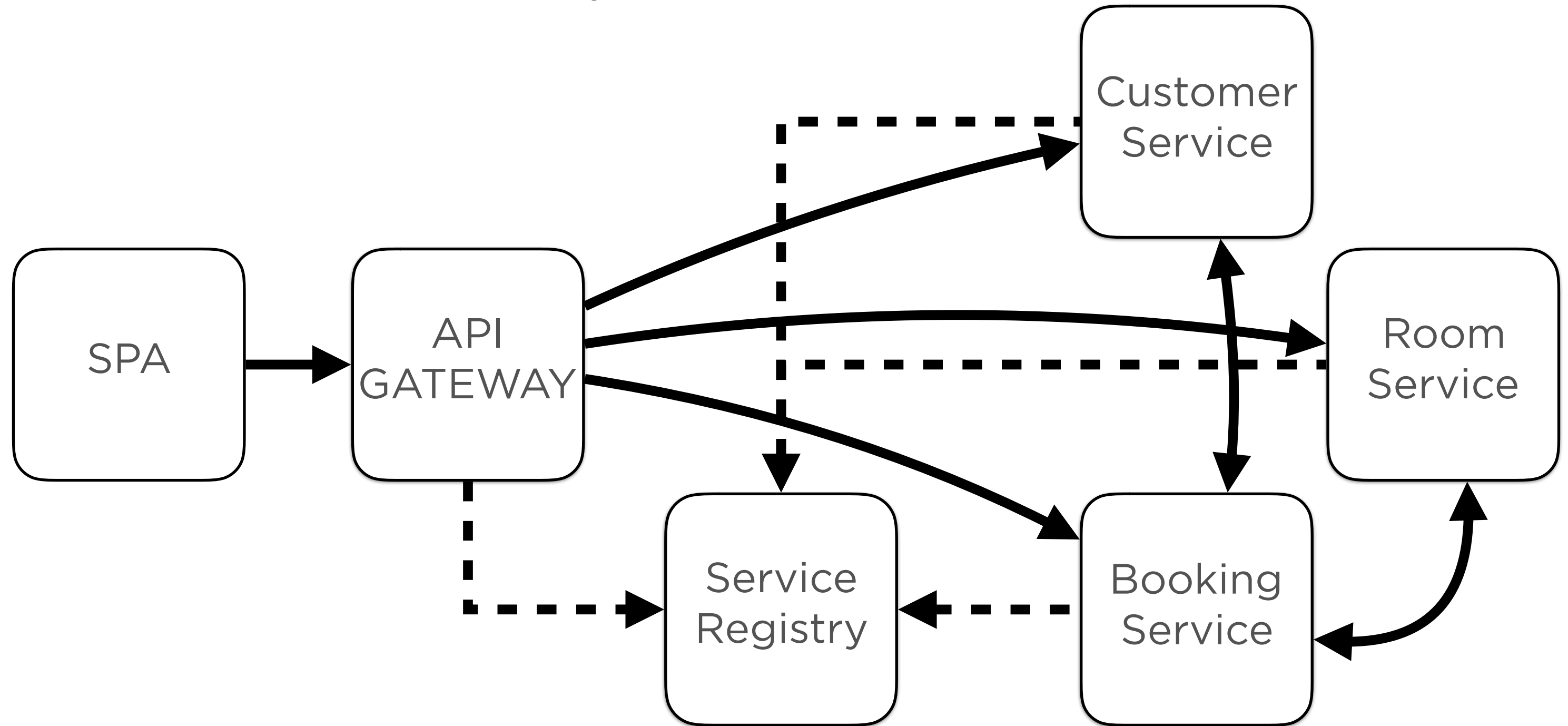


Microservice

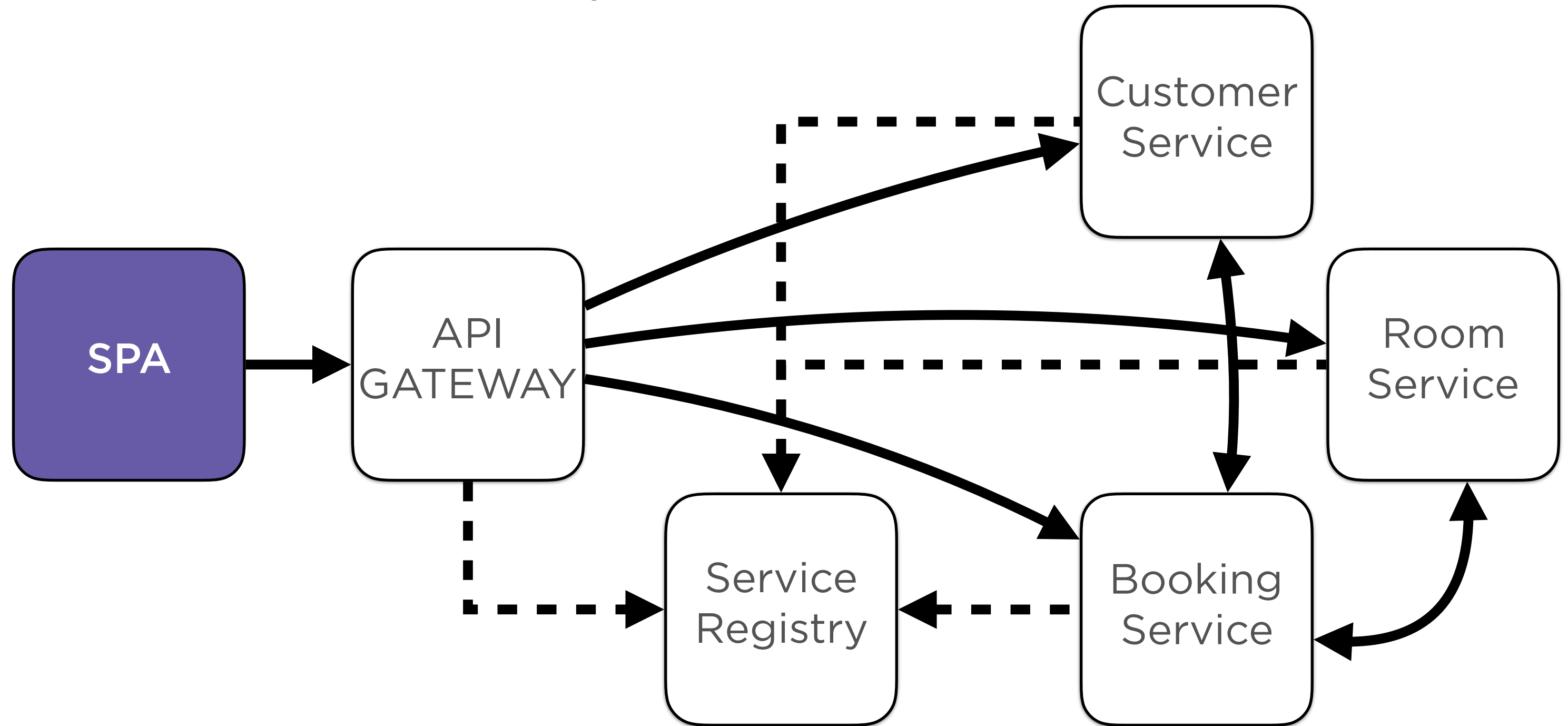
A functionally decomposed service that performs a single specialized business need.

Microservices typically communicate via RESTful APIs with JSON as the communication medium.

Project Overview

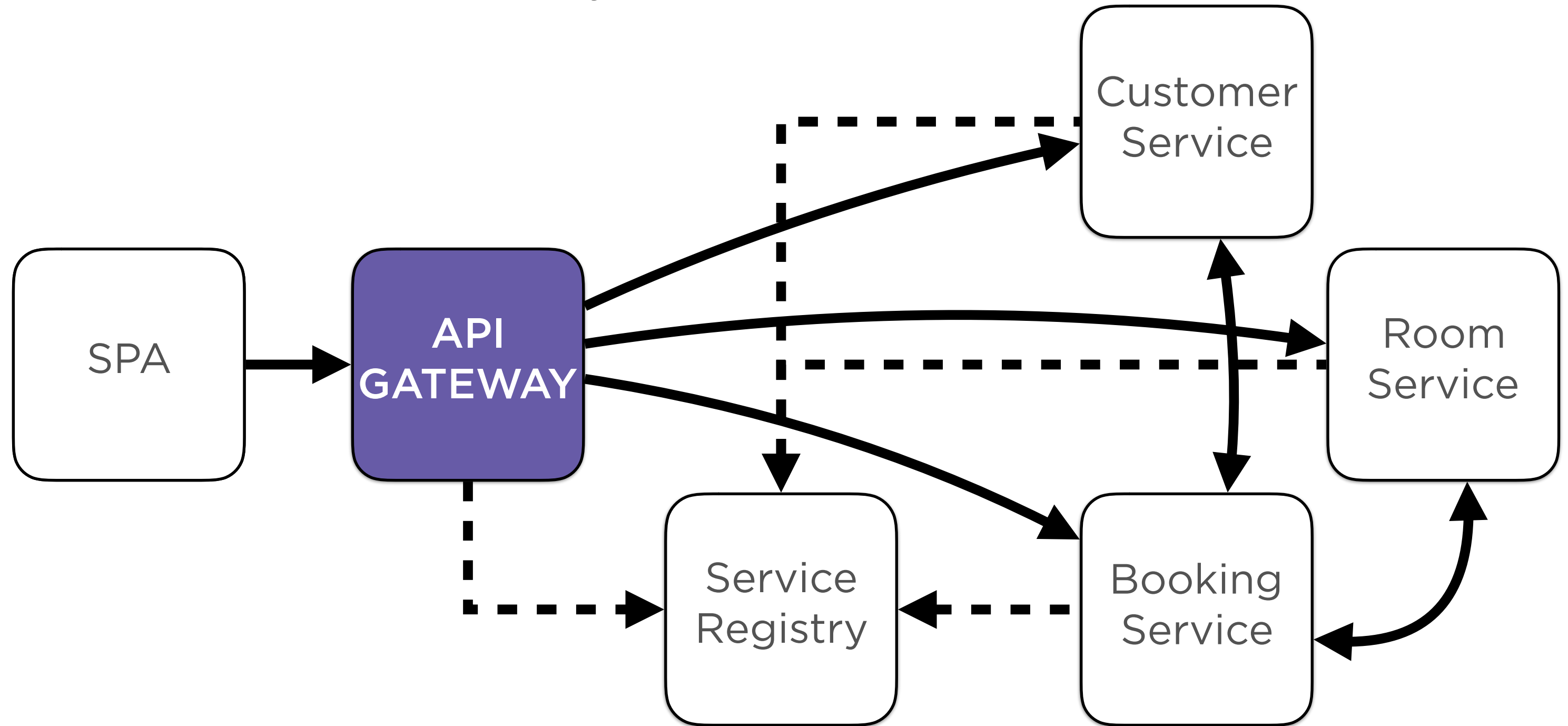


Project Overview



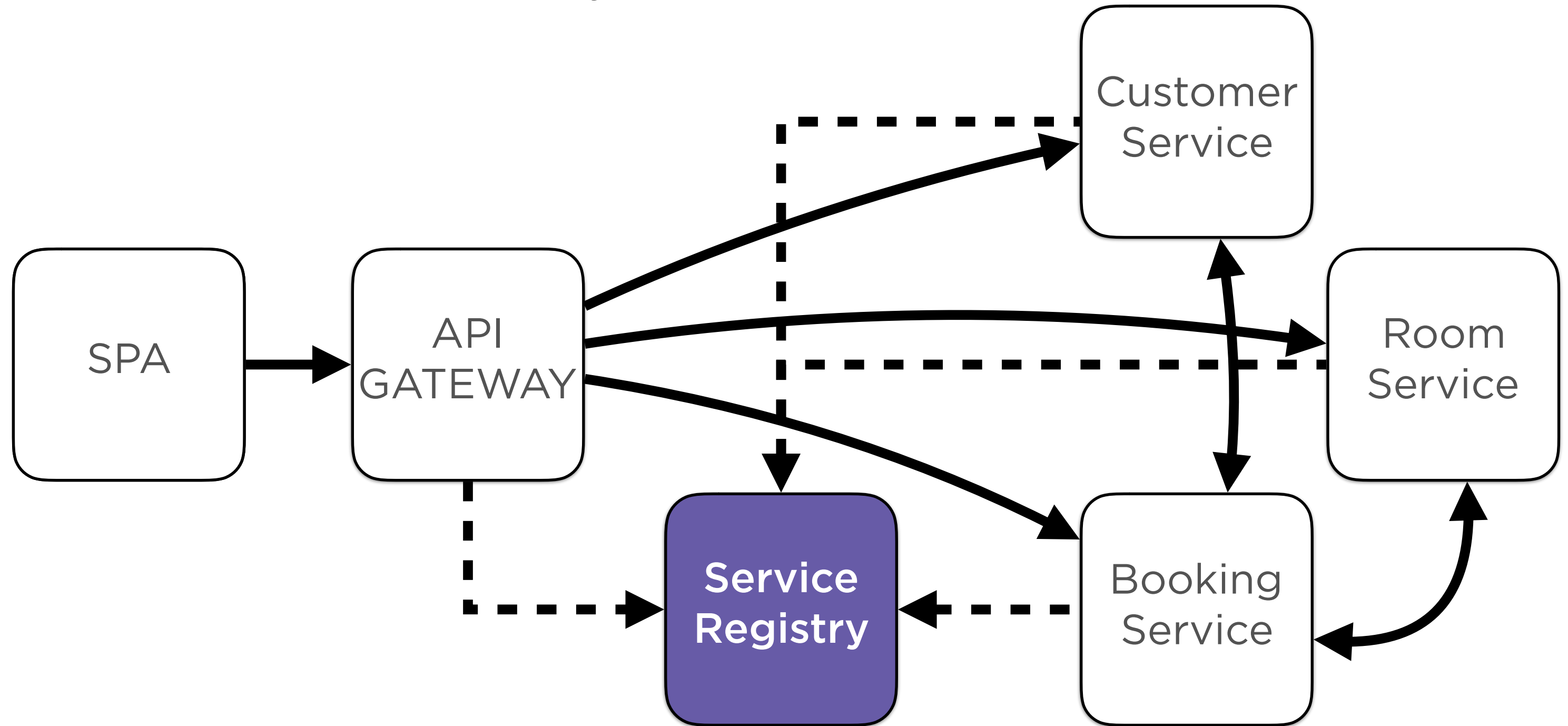
SPA - Single Page Application, the interface through which users will interact with your application

Project Overview



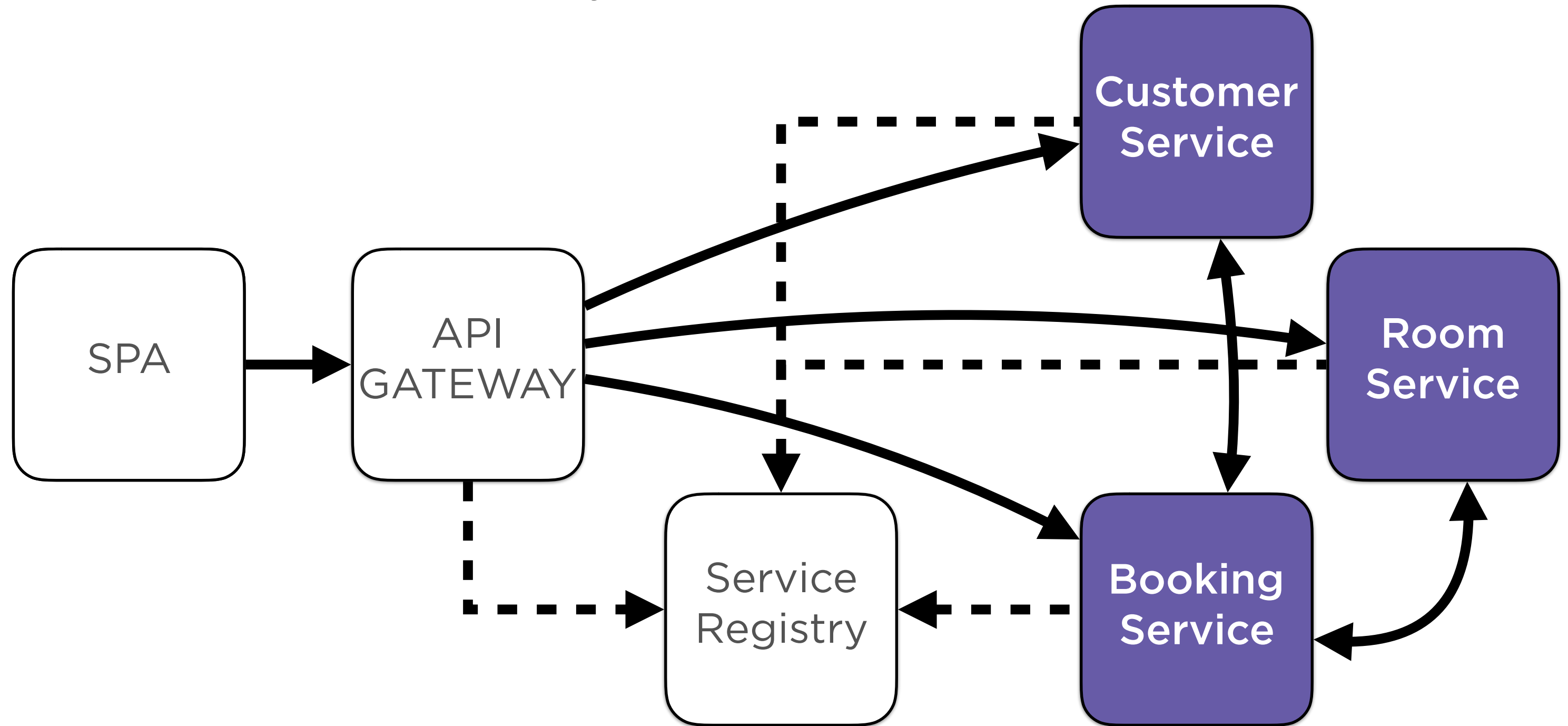
API Gateway - Point of abstraction between the SPA and backend services allowing each to evolve a change without adversely affecting the other.

Project Overview



Service Registry - Service for maintaining the location and availability of services.

Project Overview



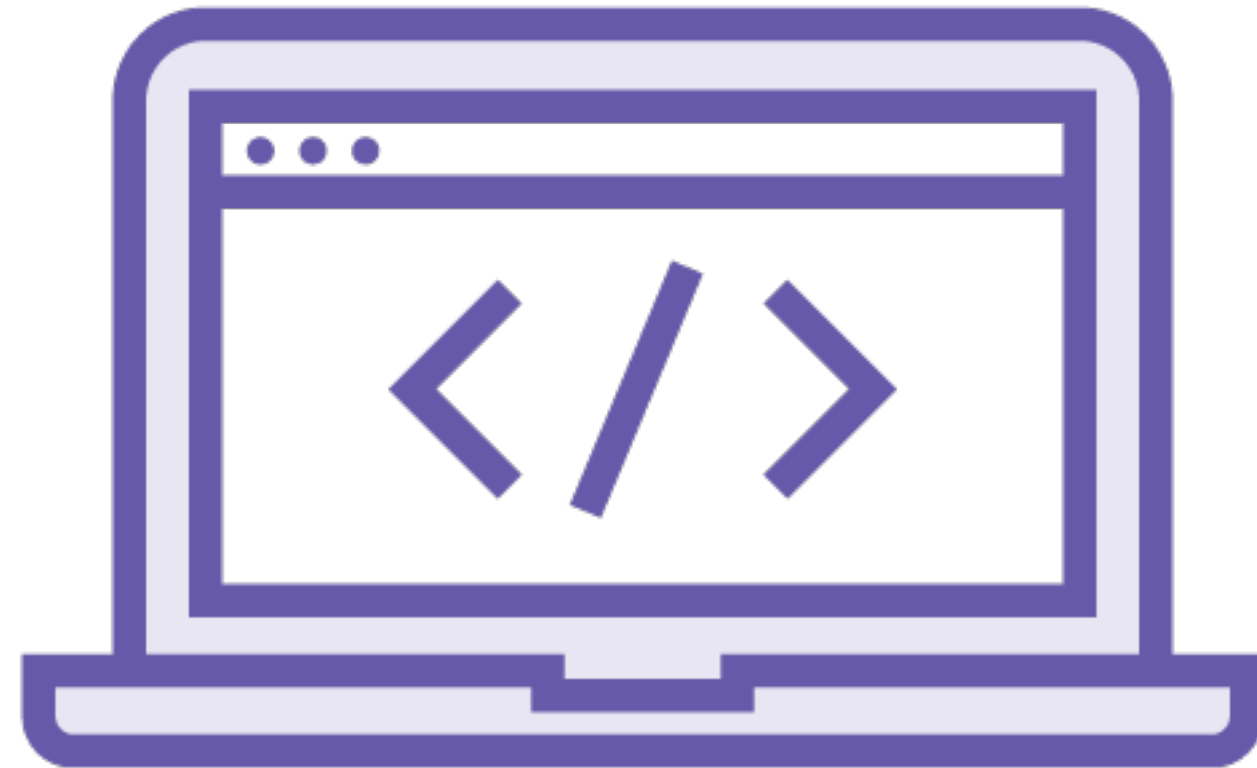
RESTful Services - The actual individual services that perform business behavior.

Test Driven Development

Write a Failing “Red” Test



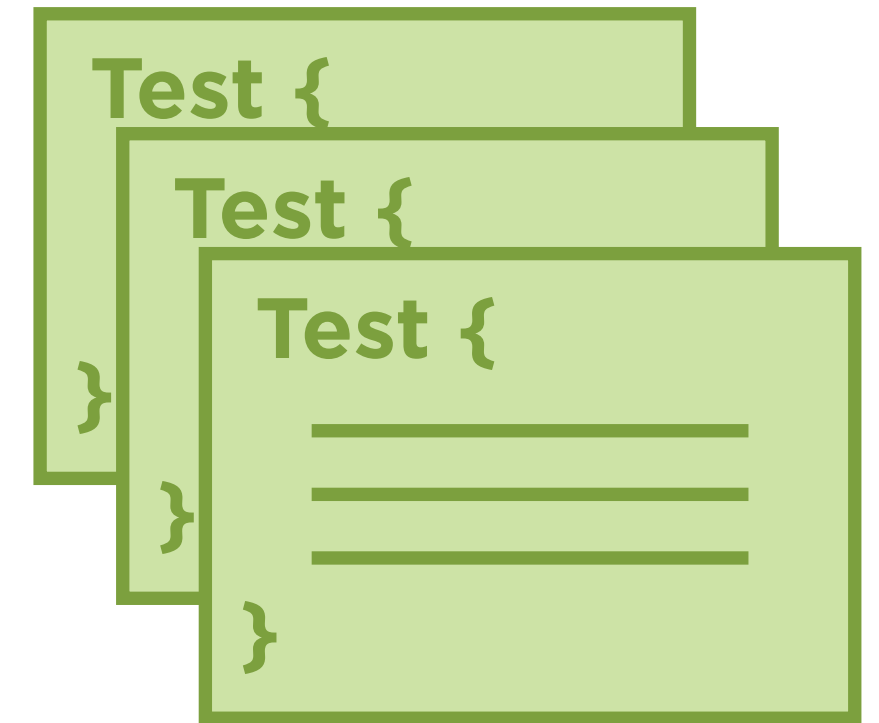
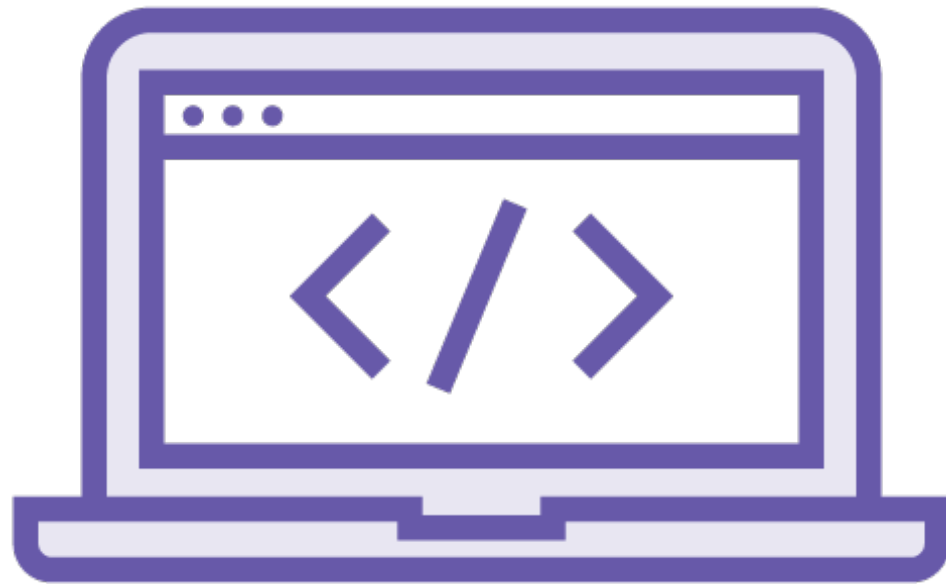
Implement the Feature



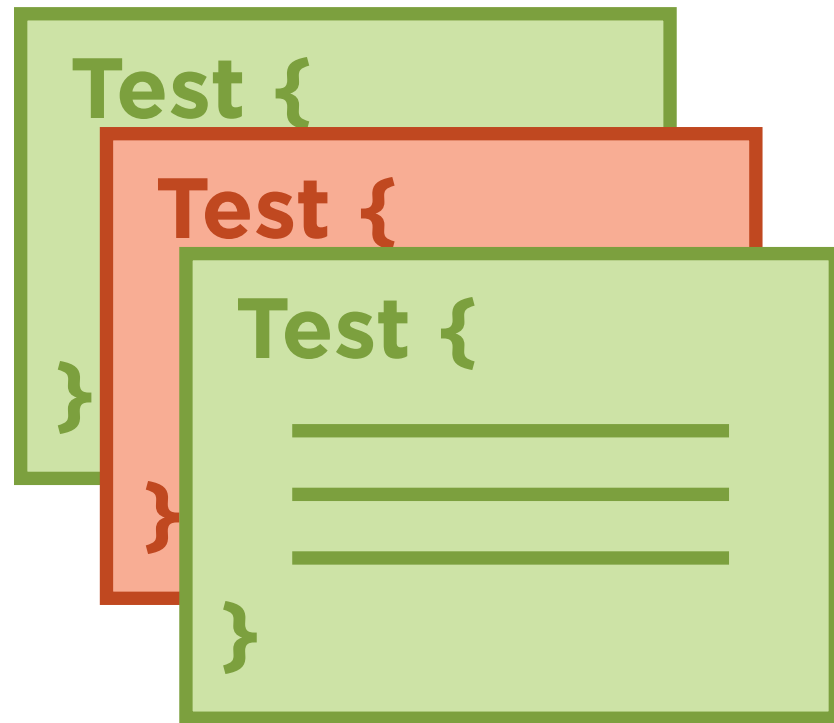
Run the Test Until it Passes (“Green”)



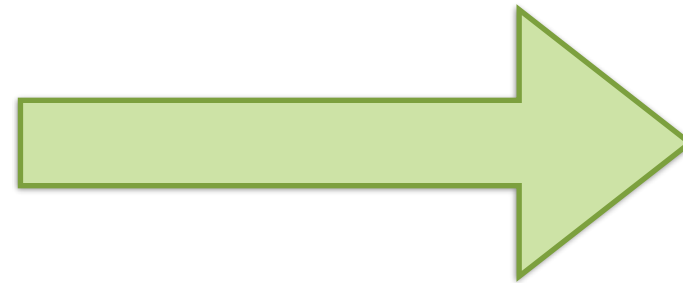
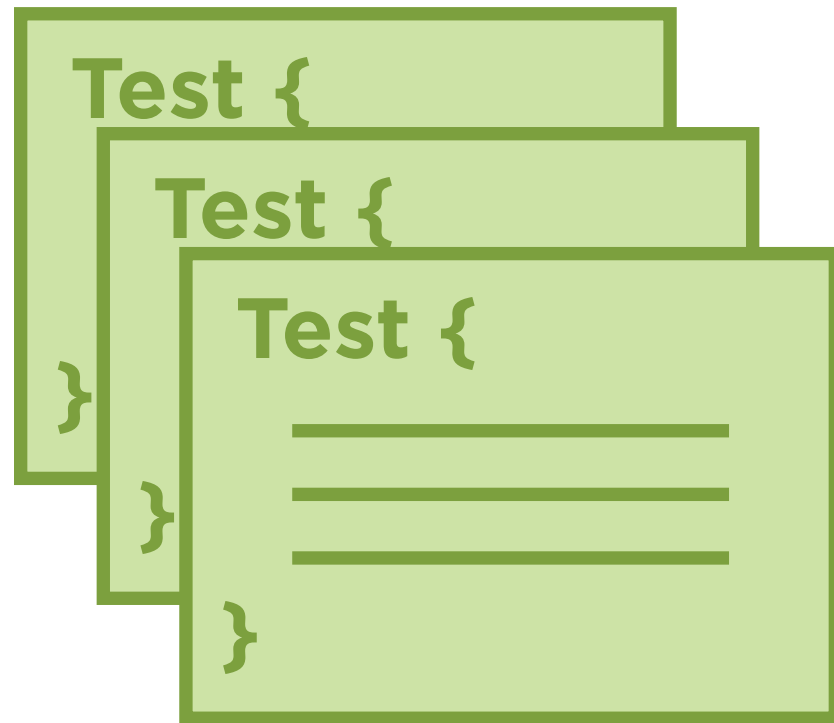
Add New Features and Test Cases



Detect Regressions



Refactor with Confidence



Test Driven Development Drawbacks



- Can be a burden when writing a complex feature
- Tests tend to be more tightly coupled

Demo

Developing new features using TDD

Create mocks for dependencies

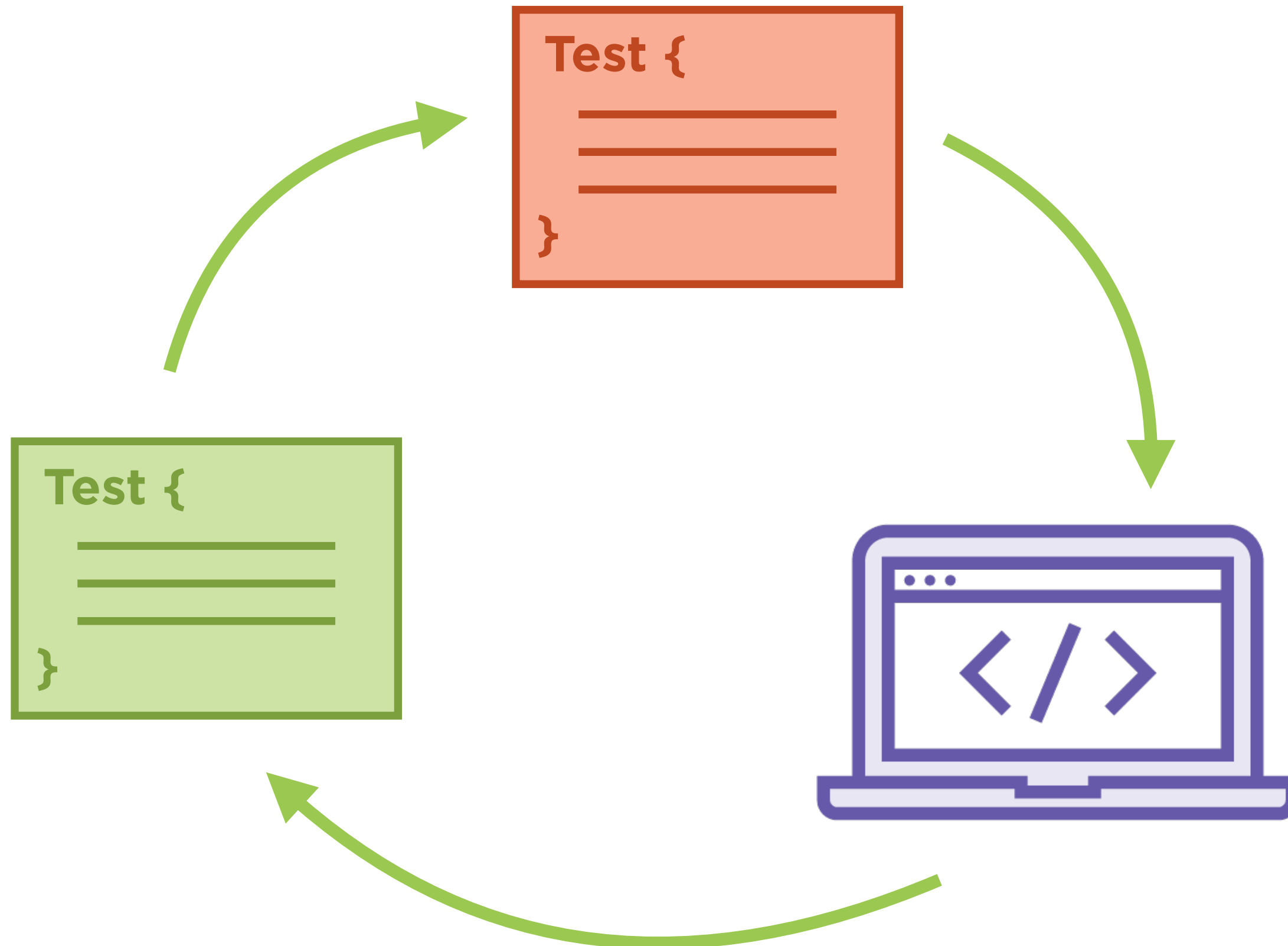
Id: 1

Description: Add room lookup

Summary: Add the ability to lookup rooms by their room number.

Return an error message for invalid room number format (non-numeric, empty)

Return an error message if no room is found



Review

The key characteristics of unit tests

The different kinds of mocks

Test Driven Development