# Virtual Bank

A Microservices Based Architecture

# Extreme Time Boxed (Pair) Programming

## Use Cases:

1) Register a new customer using a proof of identity & address

2) Open her/his account and while doing so check credit score using SSN

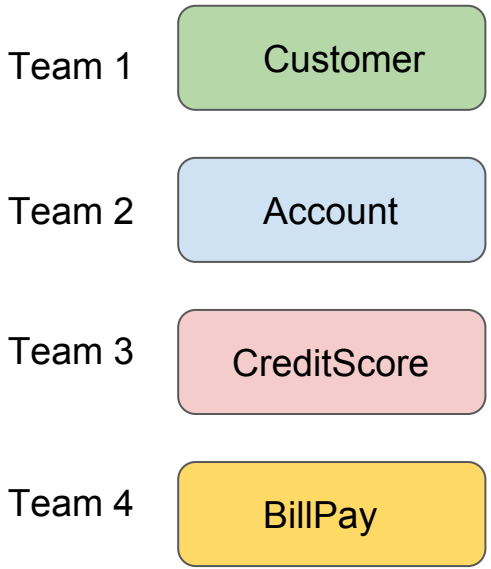
3) Schedule & Pay bills

4) For every bill payment or default, update credit score

# REST exposed bounded context resources

1) Customer
2) Account
3) CreditScore
4) BillPay

# Microservices - Divide & Rule



# Exercise Goals - A 4-Phased approach

1) Phase 1:
   a) **Focus on business logic** to create a functional microservices based setup.
   b) Based on **synchronous calls only** and leave out asynchronous concerns.
   c) **Leave out cross cutting concerns**: Authentication, Configuration, Logging, Service Discover/Location, Circuit Breaker, UI & API GW.
2) Phase 2:
   a) **Add Security**
   b) **Add Logging**
3) Phase 3:
   a) **Add Service Configuration**
   b) **Add Service Discovery/Location**
   c) **Add Circuit Breaker**
4) Phase 4:
   a) **Add API GW**
   b) **Add UI**

# Phase - 1

1) Team 1: Create a Customer Microservice exposing customer resource and its attributes in a RESTful manner using Spring Boot and associated components.

2) Team 2: Create an Account Microservice exposing customer resource and its attributes in a RESTful manner using Spring Boot and associated components.

3) Team 3: Create a CreditScore Microservice exposing customer resource and its attributes in a RESTful manner using Spring Boot and associated components.

4) Team 4: Create a BillPay Microservice exposing customer resource and its attributes in a RESTful manner using Spring Boot and associated components.

# Team 1 - Customer - Visualize

**Register New Customer:**
URI: POST /customers/
Payload/Schema: {“type”:”<consumer/business>”, “ssn”:”<ssn>”,”dob”:”<mmddyyyy>”, “idType”:”<DL/GC/PP>”, “streetAddress”:”<1234 Road St>”, “state”:”<TX>”, “zip”:”75022”}
Headers: Authorization: Basic dGVhbTE6dGVhbTFwc3dk

**Retrieve Customer:**
URI: GET /customer/{customer}
Headers: Authorization: Basic dGVhbTE6dGVhbTFwc3dk

**Retrieve SSN:**
URI: GET /customer/{customer}/ssn
Headers: Authorization: Basic dGVhbTE6dGVhbTFwc3dk

**Update Customer:**
.
**Delete Customer:**
.

# Team 1 - Customer - Schematize

Things to consider:

1) Model your domain entity Customer, keep it as single table for a start.

2) Start with an OpenAPI interface definition using Swagger (top-down) and generate REST service skeletons and work down with Spring Boot annotations
   -- or --
   Create your Spring Boot REST service using RESTController, Service & Repository annotations and then document it using Swagger's Docket.

3) Implement CRUD operations for your account against the schema you created exposing resources for HTTP verbs POST, GET, PUT/PATCH & DELETE

# Team 2 - Account - Visualize

**Register New Account:**
URI: POST /accounts/
Payload/Schema: {“customerId”:”123456”,“type”:”<current/savings>”, “openingBalance”:”25”, “monthlyFee”:”5”, “minimumBalance”:”25”, “overdraftLimit”:”200”}
Headers: Authorization: Basic dGVhbTE6dGVhbTFwc3dk

**Retrieve Account:**
URI: GET /account/{account}
Headers: Authorization: Basic dGVhbTE6dGVhbTFwc3dk

**Credit Account:**
.
**Debit Account:**
.
.

# Team 2 - Account - Schematize

Things to consider:

1) Model your domain entity Account, keep it as single table for a start.

2) Start with an OpenAPI interface definition using Swagger (top-down) and generate REST service skeletons and work down with Spring Boot annotations
   -- or --
   Create your Spring Boot REST service using RESTController, Service & Repository annotations and then document it using Swagger's Docket.

3) Implement CRUD operations for your account against the schema you created exposing resources for HTTP verbs POST, GET, PUT/PATCH & DELETE

# Team 3 - CreditScore - Visualize

**Retrieve CreditScore:**
URI: GET /ssn/{ssn}/creditscore/
Headers: Authorization: Basic dGVhbTE6dGVhbTFwc3dk

**Update CreditScore**:
URI: PUT /ssn/{ssn}/creditscore
Payload/Schema: {“event”:”<minBalancePaid/outstandingBalancePaid/paymentDefault>”, “score”:”<-15/12>”}
Headers: Authorization: Basic dGVhbTE6dGVhbTFwc3dk

# Team 3 - CreditScore - Schematize

Things to consider:

1) Model your domain entity CreditScore, keep it as single table for a start.

2) Start with an OpenAPI interface definition using Swagger (top-down) and generate REST service skeletons and work down with Spring Boot annotations
   -- or --
   Create your Spring Boot REST service using RESTController, Service & Repository annotations and then document it using Swagger's Docket.

3) Implement CRUD operations for your account against the schema you created exposing resources for HTTP verbs POST, GET, PUT/PATCH & DELETE

# Team 4 - BillPay - Visualize

**Register BillPayVendor**:
URI: POST /account/{account}/vendors/
Payload/Schema: {“vendorName”:”<AT&T/Reliant/WaterCo>”, “billType”:”<oneOff/Recurrent>”, “payDay”:”<mmddyyyy>”, “amount”:<145>}

**Create BillPayRequest**:
URI: POST /account/{account}/bills/vendor/<vendorId>
Payload/Schema: {“billType”:”<oneOff/Recurrent>”, “payDay”:”<mmddyyyy>”, “amount”:<145>}

**Retrieve BillPayStatus:**
URI: GET /account/{account}/bills/vendor/<vendorId>

# Team 4 - BillPay - Schematize

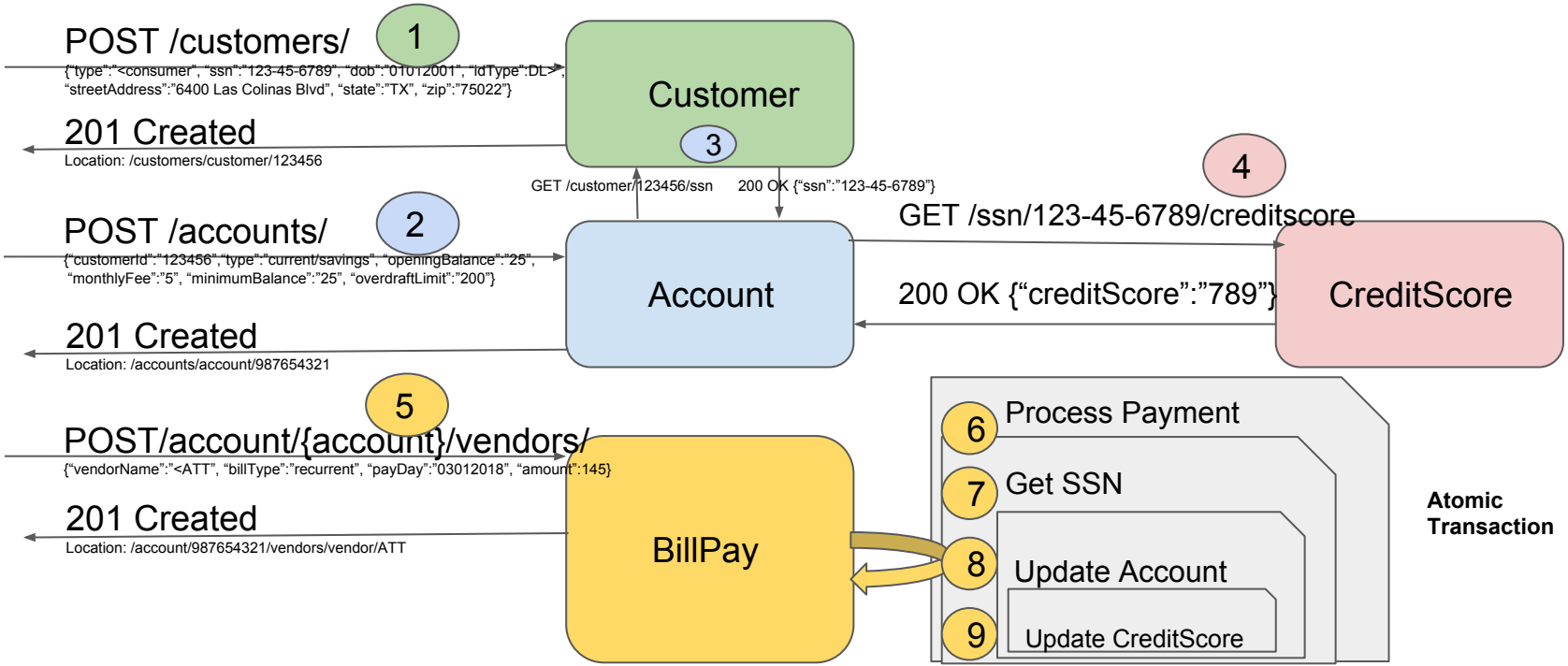
Things to consider:

1) Model your domain entity CreditScore, keep it as single table for a start.

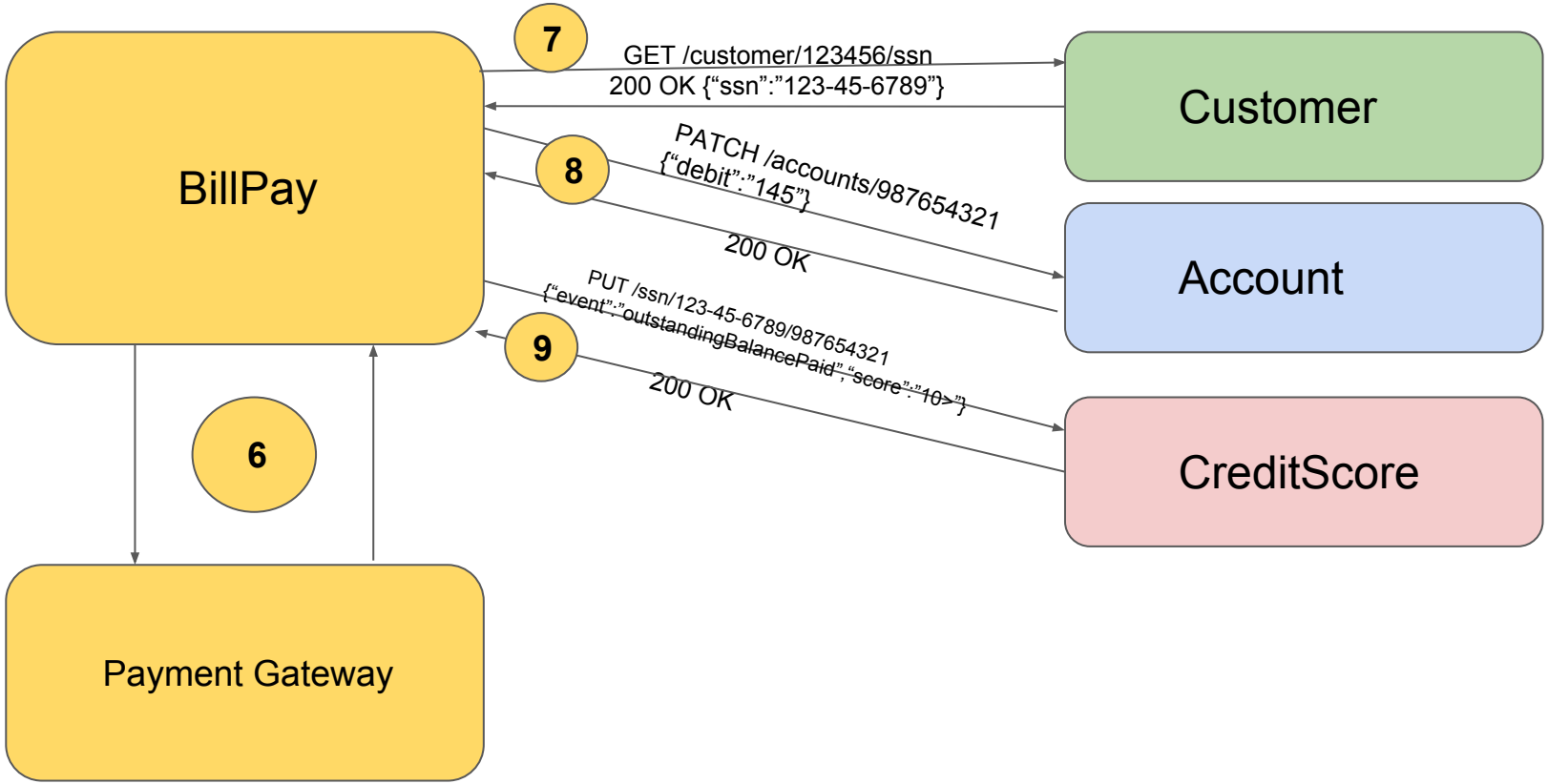
2) Start with an OpenAPI interface definition using Swagger (top-down) and generate REST service skeletons and work down with Spring Boot annotations
   -- or --
   Create your Spring Boot REST service using RESTController, Service & Repository annotations and then document it using Swagger's Docket.

3) Implement CRUD operations for your account against the schema you created exposing resources for HTTP verbs POST, GET, PUT/PATCH & DELETE

# Phase - 1 - Bringing it all together (1-5)



# Phase - 1 - Bringing it all together (6-9)



# Phase - 2

1) Adding Transaction support for 6, 7, 8, 9

2) Adding authentication and authorization

3) Adding logging