INFO303 Course Project

Contents

1. Due Dates and Marks

Phase 1 – Monday 23rd March @ 12noon. **15 marks**.

Phase 2 – Friday 15th May @ 5pm. **15 marks**.

2. Project Overview

The Customer Rewards Allegiance Scheme (CRAS) allows participating vendors to provide tiered price books for their customers. Customers can become members of CRAS by signing up via the CRAS web client. If a customer spends a sufficient amount with a participating vendor then their account will be moved to a higher tier customer group that has access to a price book with better prices and deals.

CRAS is operated in conjunction with the Vend point of sale system:

`https://vendhq.com`

Sales made with participating vendors via Vend will be collected and correlated with customers who are also registered members of CRAS.

Note that the CRAS system is entirely fictional. Also note that Vend already have several loyalty related features available in their existing product (which we are creating an alternative to with CRAS).

The INFO303 Vend store is available at:

`https://info303otago.vendhq.com/`

We will create accounts for you soon. You do not need access to the Vend store for phase 1. You might however find the Vend developer documentation useful:

`https://docs.vendhq.com/`

We will be making use of Vend's **`sale.update`** webhook to capture sale data in close to real-time:

`https://docs.vendhq.com/tutorials/guides/webhooks/introduction#saleupdate`

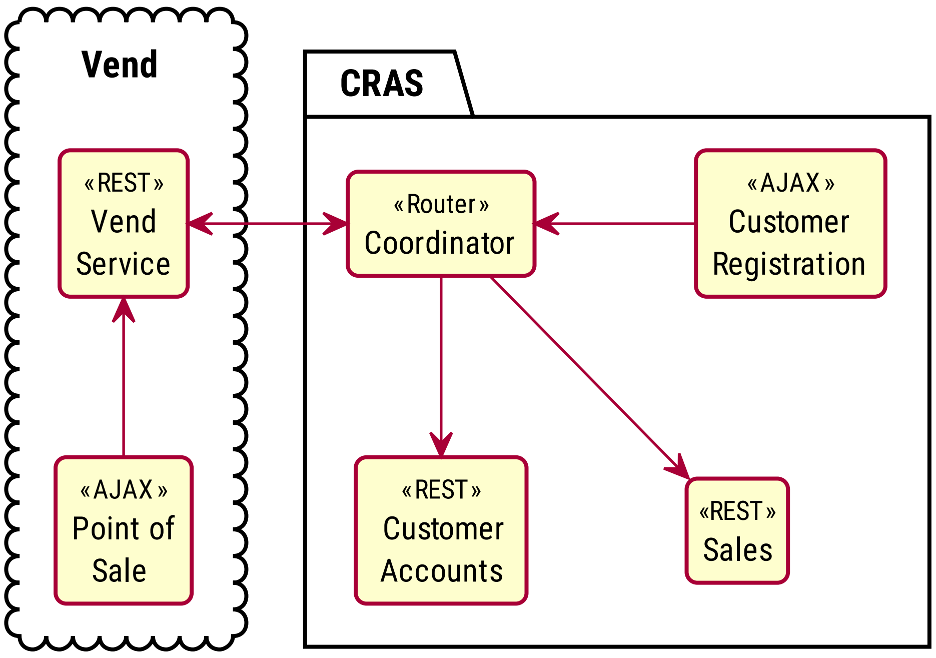
We will also be working with the **`customer`** and **`customer-group`** resources:

`https://docs.vendhq.com/reference/2/spec/customers`
`https://docs.vendhq.com/reference/2/spec/customer-groups/`

Vend have generously created a demo account for us to use in this project and have been very supportive of related student projects over the last few years. Please avoid causing any unnecessary load on Vend's servers.

3. System Architecture

3.1. Organisational Overview



You are responsible for implementing the contents of the CRAS package shown in the above diagram.

The two RESTful services are phase 1 of the project. The AJAX client, coordinator, and persistent storage (not shown in this diagram) will make up phase 2.

3.2. Simplifications to the Project

- Your services should contain proper authentication and authorization mechanisms. These issues will be covered in this course very close to the final deadline, so authentication and authorization for your services are not required for this project.

3.3. Services

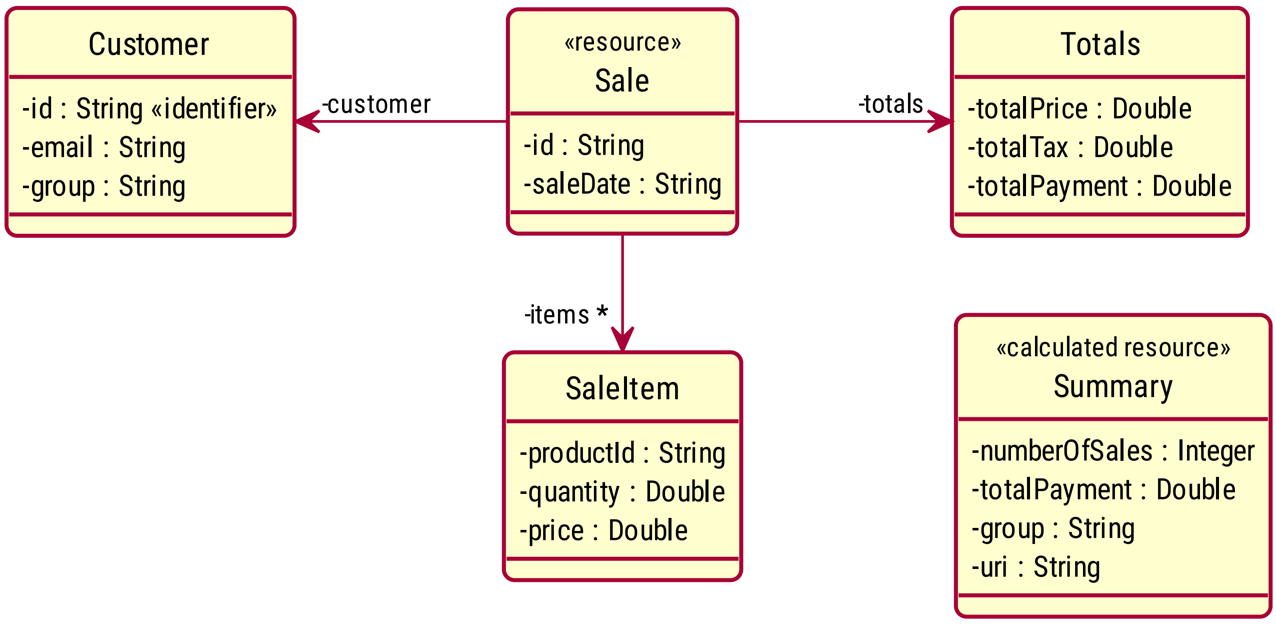
Both services are RESTful services that should be implemented using the Jooby framework.

The following describes the domain model and service operations. You need to design the service APIs yourself.

3.3.1. Sales Service

This is a RESTful service that collects sale information received from Vend.

Domain Model



This domain model is designed to be compatible with the webhook payload that is received from Vend when a sale is completed.

Note that complete sale objects (including customer, total, and sale item objects) will be passed to the service in a single JSON payload – there is no need to provide methods for adding customers, totals, etc. to the sale beyond the usual getters and setters.

The summary is a calculated resource that will aggregate the sales for a specific customer. The *group* field is based on a threshold determined by total payment and represents which customer group a customer belongs to.

Service Operations

The service needs to support the following operations:

- A client needs to be able to add a new sale.
- A client needs to be able to get all sales for a specific customer.
- A client needs to be able to get a summary of the sale data for a specific customer.
- A client needs to be able to delete a sale.

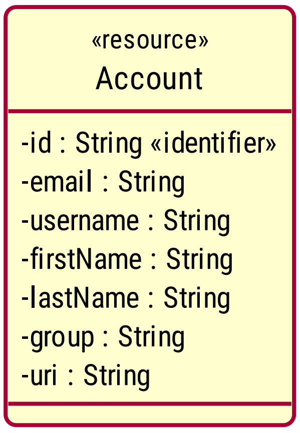
You can create your own dummy customer groups and threshold logic for testing with. For example, groups "Regular Customers", and "VIP Customers". When a customer makes $5000 worth of sales they are moved from the "Regular Customers" group to the "VIP Customers" group.

Storage

For phase 1, collections based storage is all that is required. Remember that there will be many sales for each customer, so choose appropriate data structures.

3.3.2. Customer Accounts Service

Domain Model



Service Operations

The service needs to support the following operations:

- A client needs to be able to create a new customer account.
- A client needs to be able to update the details of an existing account.
- A client needs to be able to get all registered customer accounts.
- A client needs to be able to delete a customer account.

Storage

For phase 1, collections based storage is all that is required.

3.3.3. Clients

Being able to quickly verify that your services are functioning correctly is an important part of debugging distributed systems. As such, we expect you to create a test client using appropriate frameworks that tests each operation of each of your services.

You can use JUnit if you wish. A class with a main method that calls each operation and verifies that the results are correct is acceptable.

You should have at least two JUnit tests, or two classes with main methods. One for each service.

3.4. Submission and Requirements

Implement the two services as described in section 3.3.

- Each service should be contained within its own Gradle project. Use the starting project from lab 2 for these services.

  You will need to ensure that the two services each run on different ports so that they can both be run at the same time. The main method in the in the starting project of the **`lab03-angularjs`** project shows you how to change the server port.

- Each service project should include a YAML file containing the OpenAPI specification for the service.

Implement the two clients as described in section 3.3.3. You will need one project for each client (since Codegen assumes one client per project). Use the starting project for **`lab03-retrofit`** for these projects.

Submission will by via the INFO303 GitBucket. More details on how to submit will be made available closer to the deadline.

3.5. Minimum Requirements for Phase 1

As mentioned in section 8.4 of the course outline we have some minimum standards that must be met before we will mark a submission. Submissions that do not meet this standard will receive a mark of 0.

The minimum requirements for phase 1 are as follows:

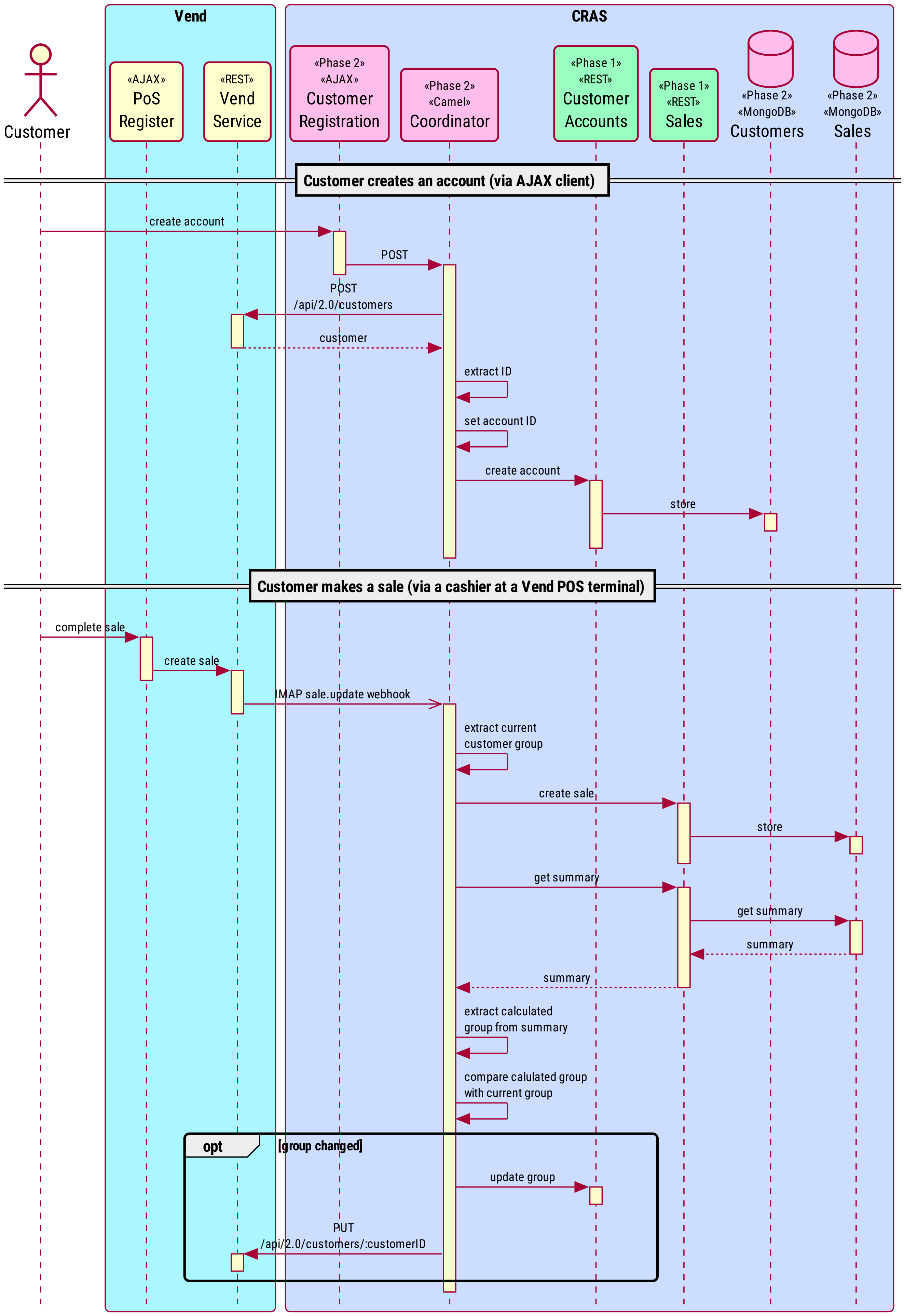
- A customer account must be able to be created in the *customer accounts* service via an HTTP request.
- Your projects must all build without any errors.
- Your code should be reasonably presented. Code should be formatted and use appropriate identifier names. Industry standards for naming should be adhered to where appropriate.
- Your Git history should show incremental development over a reasonable period of time. Submissions that contain very few commits over a short period of time will be accepted.

  If you have trouble with your Git repository then let us know ASAP so we can help you to fix it.

4. System Interaction Diagram

The sequence diagram on the next page gives you an overview of the entire system.

Phase 1 components are shown in green. Phase 2 components are shown in pink – you do not need to implement anything relating to the phase 2 components in phase 1. For those who have color vision problems (or black and white printers) there are also stereotypes indicating which phase a component belongs to.


Vend
CRAS
Customer
«AJAX» PoS Register
«REST» Vend Service
«Phase 2» «AJAX» Customer Registration
«Phase 2» «Camel» Coordinator
«Phase 1» «REST» Customer Accounts
«Phase 1» «REST» Sales
«Phase 2» «MongoDB» Customers
«Phase 2» «MongoDB» Sales
Customer creates an account (via AJAX client)
create account
POST
POST /api/2.0/customers
customer
extract ID
set account ID
create account
store
Customer makes a sale (via a cashier at a Vend POS terminal)
complete sale
create sale
IMAP sale.update webhook
extract current customer group
create sale
store
get summary
get summary
summary
summary
extract calculated group from summary
compare calulated group with current group
opt
[group changed]
update group
PUT /api/2.0/customers/:customerID