1

User manual

1.1 Dashboard

1.1.1 Top section dashboard

The top of the dashboard gives the user access to the following information:

- A link to the calendar of the trainer's current batch.
- Personal information about the trainer.
- Information about the batch.
- The progress of the batch, measured in number of days completed and number of subtopics completed

1.1.2 Bottom section dashboard

The bottom of the dashboard gives the user access to the following information:

- A link to the calendar of the trainer's current batch.

1.1.3 All Batches

The All Batches tab gives the user access to the following functionality:

- Search for keywords matching fields in the all batches table using the search bar in the top right corner.
- View information about all batches, including batch name, trainer, batch type, and the start and end date of the batch

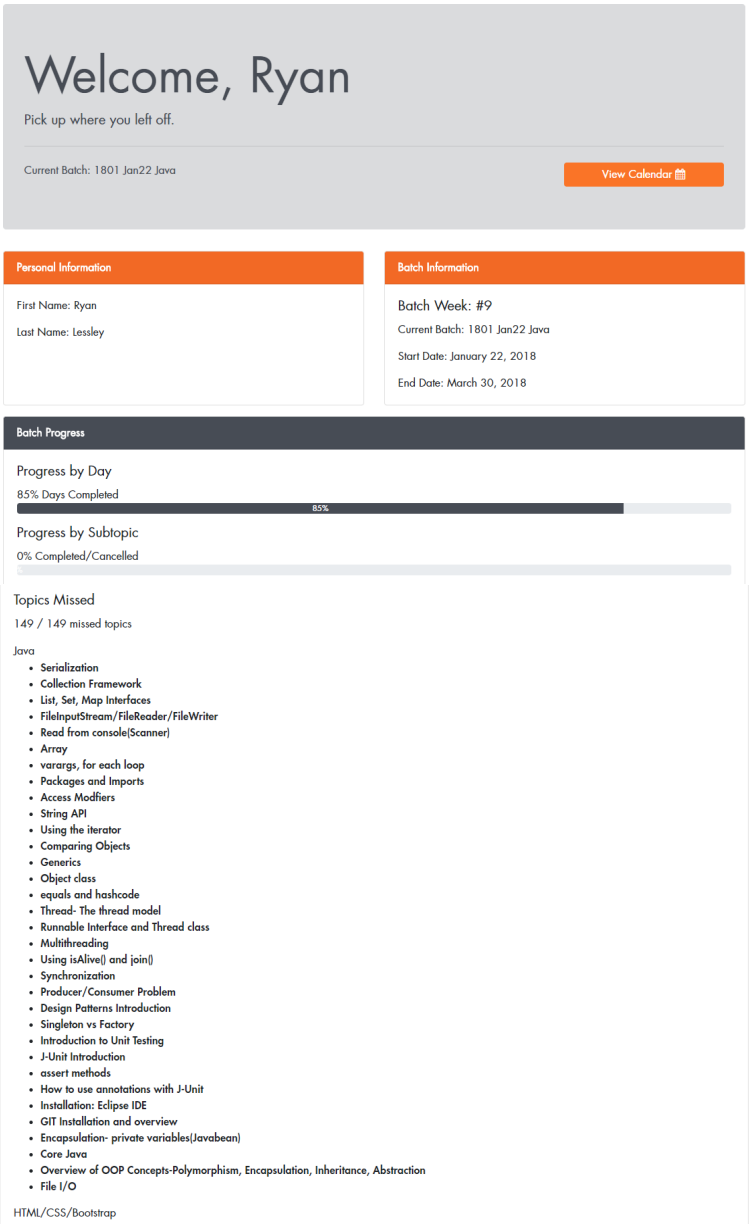
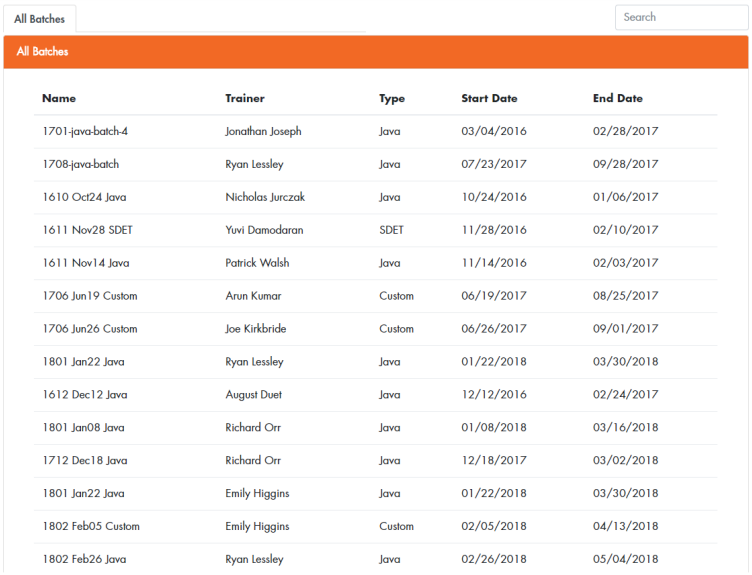
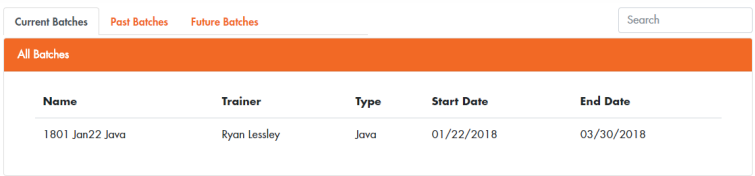


Figure 1.1: Bam dashboard



Name	Trainer	Type	Start Date	End Date
1701-java-batch-4	Jonathan Joseph	Java	03/04/2016	02/28/2017
1708-java-batch	Ryan Lessley	Java	07/23/2017	09/28/2017
1610 Oct24 Java	Nicholas Jurczak	Java	10/24/2016	01/06/2017
1611 Nov28 SDET	Yuvi Damodaran	SDET	11/28/2016	02/10/2017
1611 Nov14 Java	Patrick Walsh	Java	11/14/2016	02/03/2017
1706 Jun19 Custom	Arun Kumar	Custom	06/19/2017	08/25/2017
1706 Jun26 Custom	Joe Kirkbride	Custom	06/26/2017	09/01/2017
1801 Jan22 Java	Ryan Lessley	Java	01/22/2018	03/30/2018
1612 Dec12 Java	August Duet	Java	12/12/2016	02/24/2017
1801 Jan08 Java	Richard Orr	Java	01/08/2018	03/16/2018
1712 Dec18 Java	Richard Orr	Java	12/18/2017	03/02/2018
1801 Jan22 Java	Emily Higgins	Java	01/22/2018	03/30/2018
1802 Feb05 Custom	Emily Higgins	Custom	02/05/2018	04/13/2018
1802 Feb26 Java	Ryan Lessley	Java	02/26/2018	05/04/2018

Figure 1.2: All Batches



Name	Trainer	Type	Start Date	End Date
1801 Jan22 Java	Ryan Lessley	Java	01/22/2018	03/30/2018

Figure 1.3: All Batches

1.1.4 My Batch

The My Batches tab gives the user access to the following information:

- Current Batches – Information about the batch(es) currently in progress that a trainer is responsible for
- Past Batches – Information about the batch(es) that have been completed that a trainer was responsible for
- Future Batches – Information about the batches that the trainer will be responsible for in the future

1.1.5 Calender

The Calendar grants the user the following functionality:

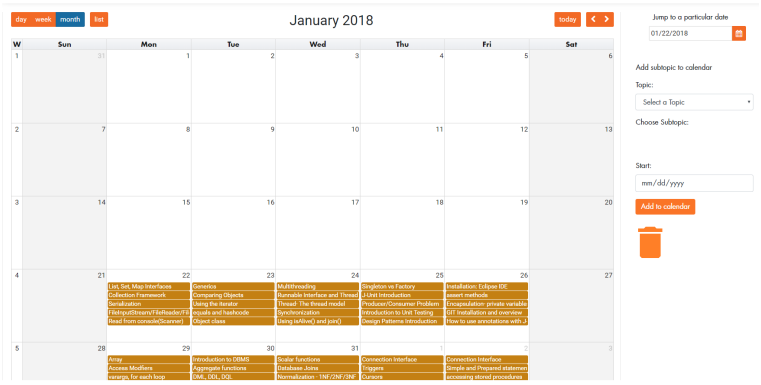


Figure 1.4: Calender view

- View the schedule of subtopics to be covered on a certain date for a batch
- Add a subtopic to the calendar by:
  - Selecting the topic the subtopic you wish to add is located.
  - Choose the subtopic to add to the date.
  - Select the date to cover that subtopic.
  - Select “Add to Calendar”.
- Add a subtopic to the calendar by:
  - Selecting the topic the subtopic you wish to add is located.
  - Dragging the subtopic to wish to add to the date on the calendar.

- Change the status of a subtopic by:
  - Clicking on the subtopic you wish to change the status of in the calendar until the status you wish to give it is reached.
- Change the date a subtopic is covered by:
  - Dragging the subtopic to the new date you wish to cover the subtopic.
- Remove a subtopic from the schedule by:
  - Dragging the subtopic you wish to remove from the schedule to the trash can icon.

1.1.6 Curriculum Editor

The Curriculum Editor gives the user the following functionality:

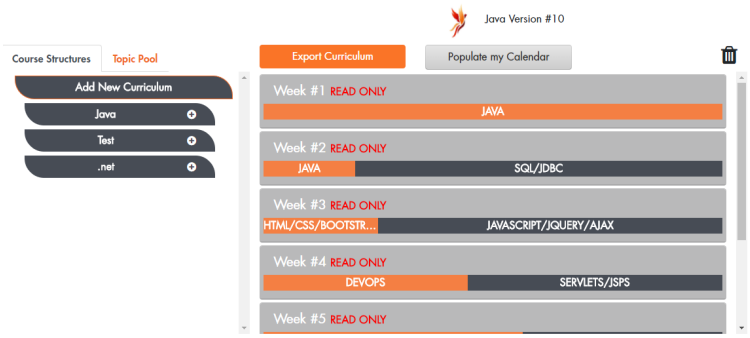


Figure 1.5: Curriculum Editor

- Export Curriculum – Exports an Excel file containing the curriculum plan based on the current curriculum in the editor
- Populate my Calendar – Populates the calendar for the current batch with the current curriculum in the editor
- Delete Curriculum Version – Delete the curriculum version by:
  - Clicking the trash can icon to the top right of the Curriculum Editor when the version you wish to delete is displayed.
  - Confirm that you want to delete that version of the curriculum.

The Course Structures tab side menu gives the user the following functionality:

- Add New Curriculum Label – Allows the user to create a new curriculum label that can maintain one or more curriculum versions by:
  - Clicking “Add New Curriculum”.
  - Entering the name of the label into the text box.
- Add New Curriculum – Allows the user to create a new version of a curriculum under an existing label by:
  - Clicking the “plus” symbol next to the label you wish to create a new curriculum version for.
  - Populating each week inside of the newly created curriculum version template.

Topic Pool

The Topic Pool tab side menu gives the user the following functionality:

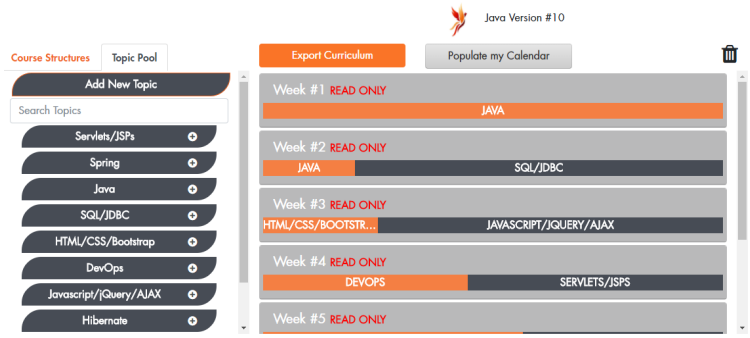


Figure 1.6: Topic Pool

- Add New Topic – The user can create a new topic for the topic pool by:
  - Clicking “Add New Topic”.
  - Entering the name of the new topic into the text box.
- Add New Subtopic – The user can create a new subtopic for a topic by:
  - Clicking the “plus” icon next to the topic you wish to add a new subtopic to.
  - Entering the name of the new subtopic into the text box.
- Search Topics – Allows you to search for subtopics by name.

1.1.7 Objectives Manager

The Objectives Manager's Subtopics Completion Chart gives the user the following information:

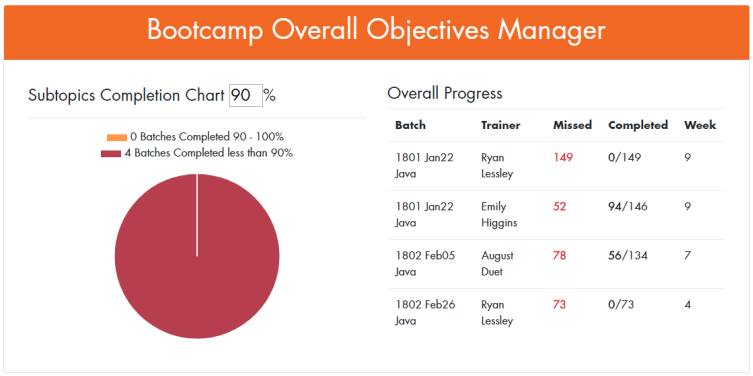


Figure 1.7: Objectives Manager

- A pie chart showing the number of batches that completed less than x percent of subtopics and the number of batches that completed x percent or more subtopics.
- An overall progress table for each active batch, showing the batch name, the trainer, number of missed and completed sub tasks, and the current week the batch is in

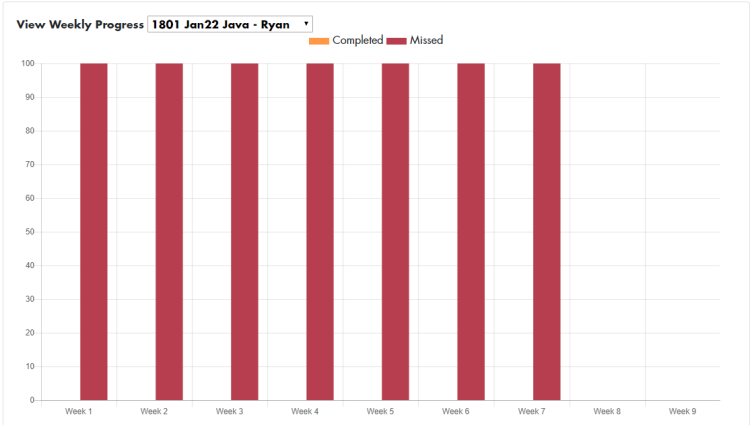


Figure 1.8: Weekly Progress

The Objectives Manager's Weekly Progress Chart gives the user the following information:

- A bar graph showing the percentage of subtopics completed for each week of a batch.

2

Overview

2.1 BAM

BAM is Revature's batch management platform for recording and retrieving batch information. It is built with a cutting edge, micro service architecture to allow for modularity, system resilience, and fast development. It's infrastructure services include a circuit breaking metric display, multiple discovery services for heartbeat monitoring, a configuration server for business service backups, and an API gateway that handles redirecting requests to different business services.

2.2 Infrastructure

2.2.1 Discovery Service

Discovery services are used to monitor business micro-services through the use of heart beats. This application utilizes Eureka to send heart beats out on timed intervals. This helps reduce API calls to micro-services that are no longer operational.

2.2.2 Configuration Service

Configuration Service used to apply a uniform configuration across specified micro-services. This application uses Configuration Server to provide configurations to any backup services that are needed to be spun up. It helps reduce repeated configuration files across micro-service applications.

2.2.3 API Gateway

API gateways are used to provide a single endpoint for client's to access and it can apply filters across data. This application uses Zuul Netflix

dependency to redirect client requests to different online business micro-services. It helps direct traffic throughout the system and allows for dynamic trafficking with the help of the circuit breaker.

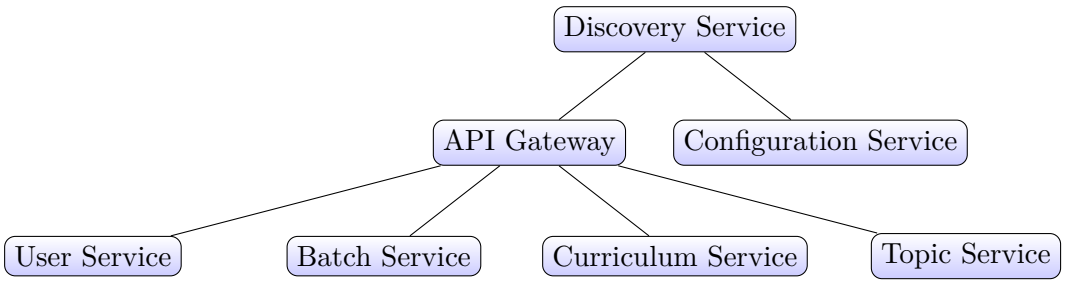
2.2.4 Circuit Breaking Service

Circuit breaking is used to improve BAM's resiliency. This application uses Hystrix dashboard and Hystrix circuit breaker to handle errors with fallback methods. The Hystrix dashboard allows easy monitoring of all business service clusters and their circuit breakers.

2.2.5 Business Services

Business services are the micro-services that handle any application specific operations. In BAM, these business services consist of: User micro-service, Batch micro-service, Curriculum micro-service, and Topic micro-service.Each of these micro-services persist data and also expose internal endpoints for any other business micro-service to request resources.

2.2.6 Infrastructure Design



3

States

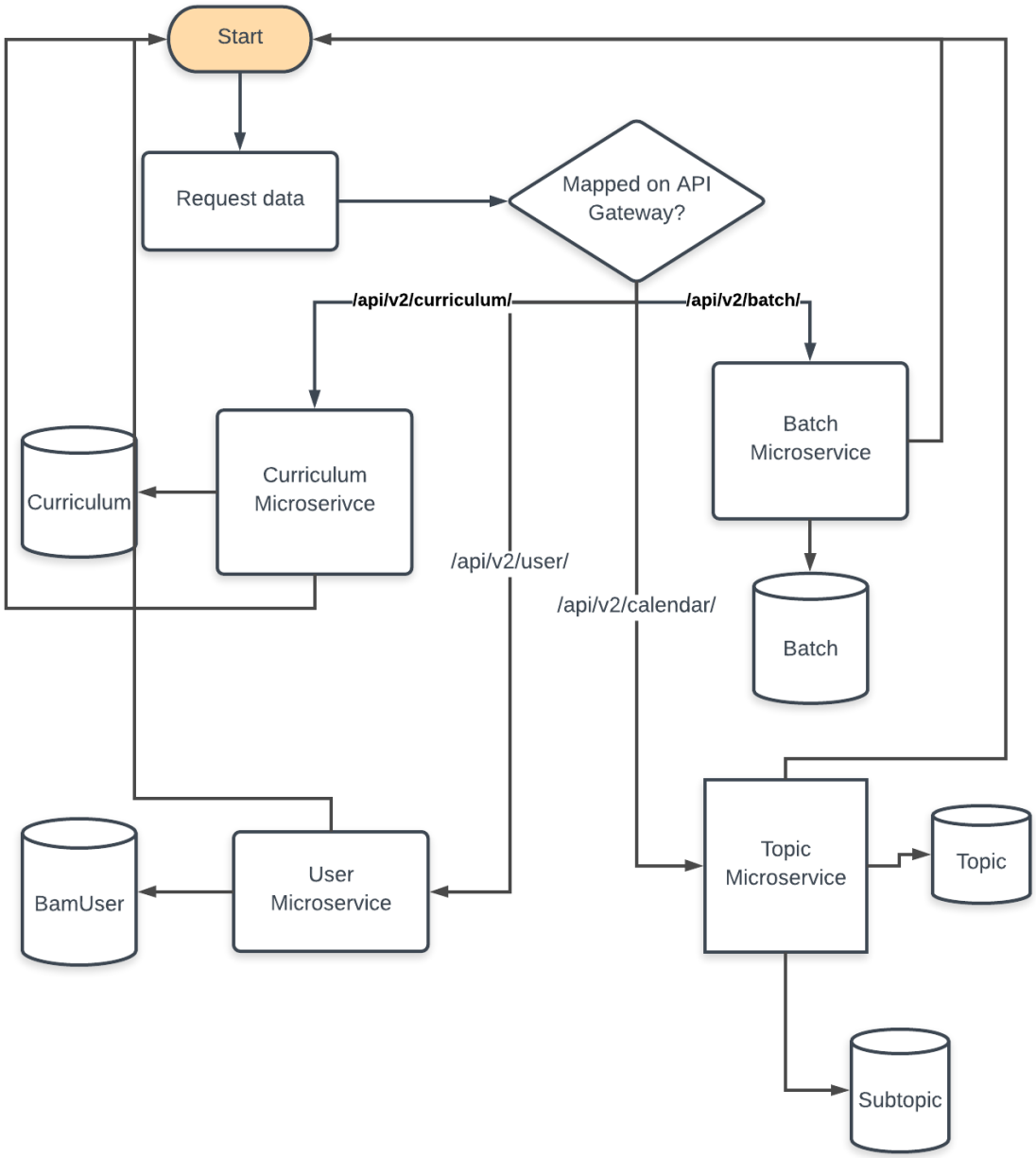


Figure 3.1: State Diagram

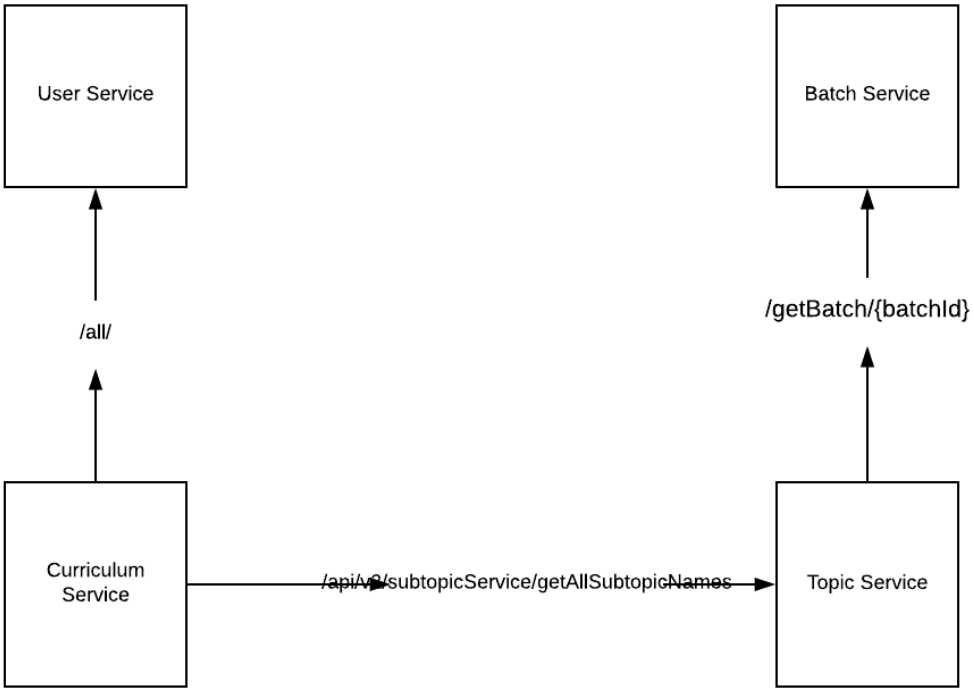


Figure 3.2: Internal Connection

4

System Requirements

- Java 8
- Maven
- Spring boot starter 1.5 +
- Angular 4 +
- Angular Cli 1.4 +
- CLI tool

5

Curriculum Service

5.1 Abstract

The curriculum micro-service is crucial to our design as well as the original monoliths. In the monolith, the curriculum model contained information from curriculum service and the topic service. Each of those services retrieved data from their respective repositories for the curriculum model. When converting the monolith to a micro-service architecture, the micro-services are determined by the model and the internal service communication is handled via REST templates. The services in the original monolith are now basically the internal service communication that is going on in between the business services. We chose to keep many of the same endpoints as the original monolith to reduce the necessary changes in the front end application Janus.

5.2 Service Architecture

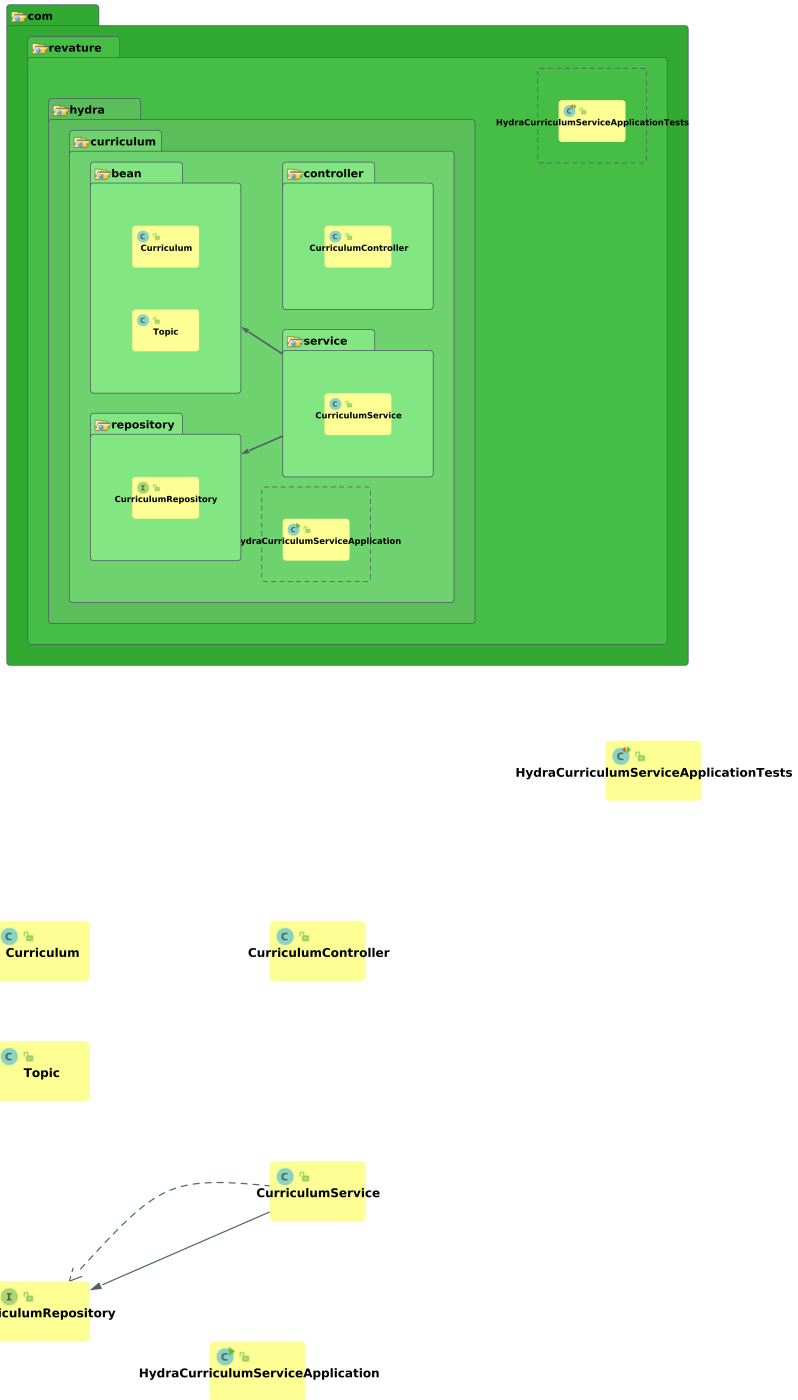


Figure 5.1: Curriculum Service Infrastructure

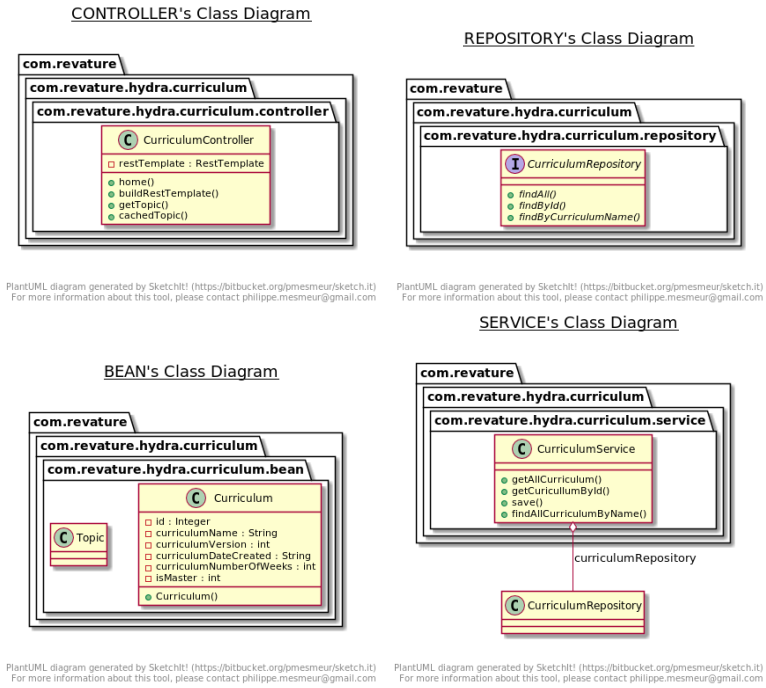


Figure 5.2: Curriculum Service Hierarchy

6

Batch Service

6.1 Abstract

The batch micro-service is a major component of this micro-service system. It communicates with the user micro-service which gives individual details about each user in the system. They are tied together with keys but stored individually in completely different databases. The batch micro-service stores batch data within the batch repository. From here, it can send data to any other micro-service that needs batch data.

6.2 Batch Architecture

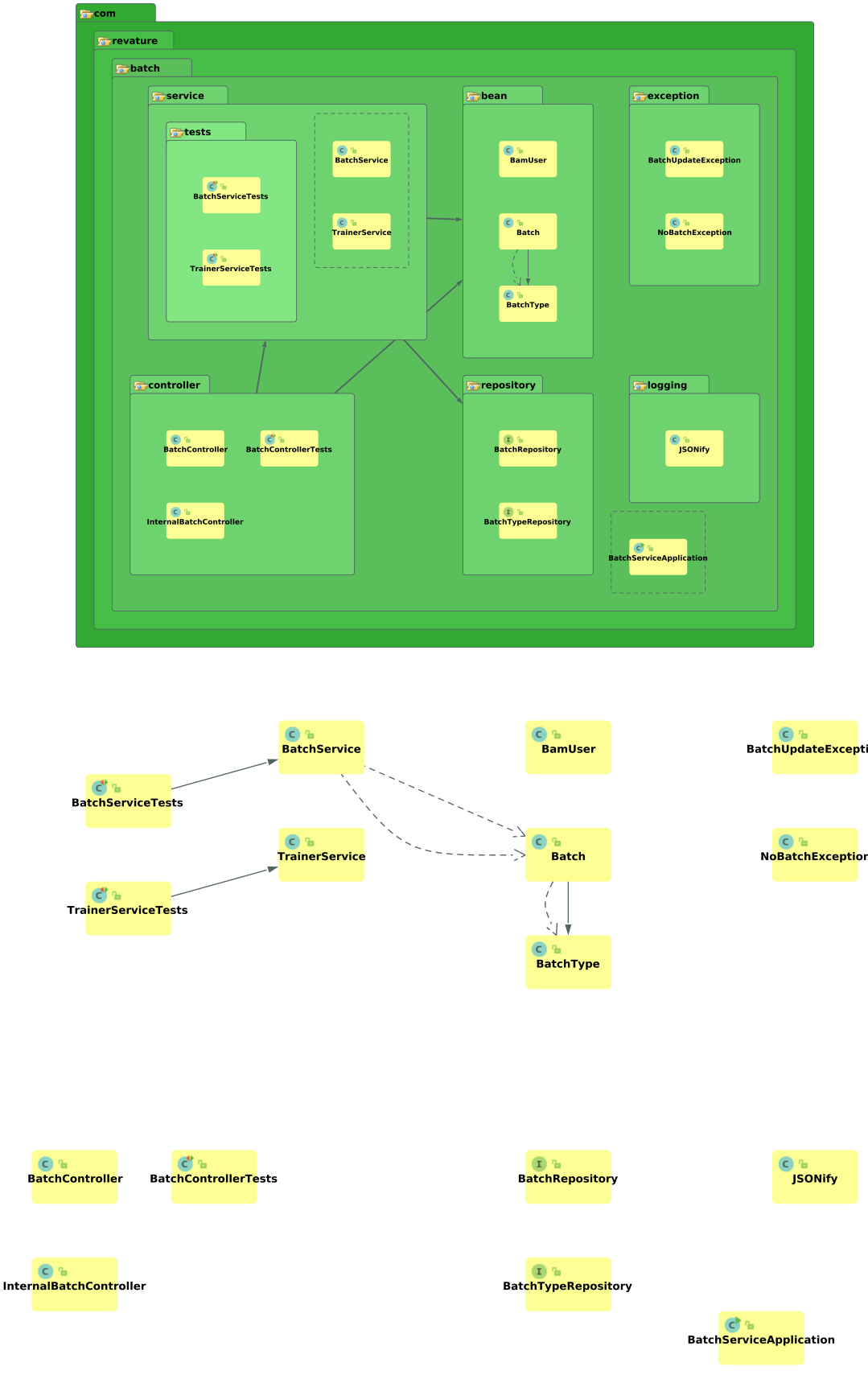


Figure 6.1: Batch Service Infrastructure

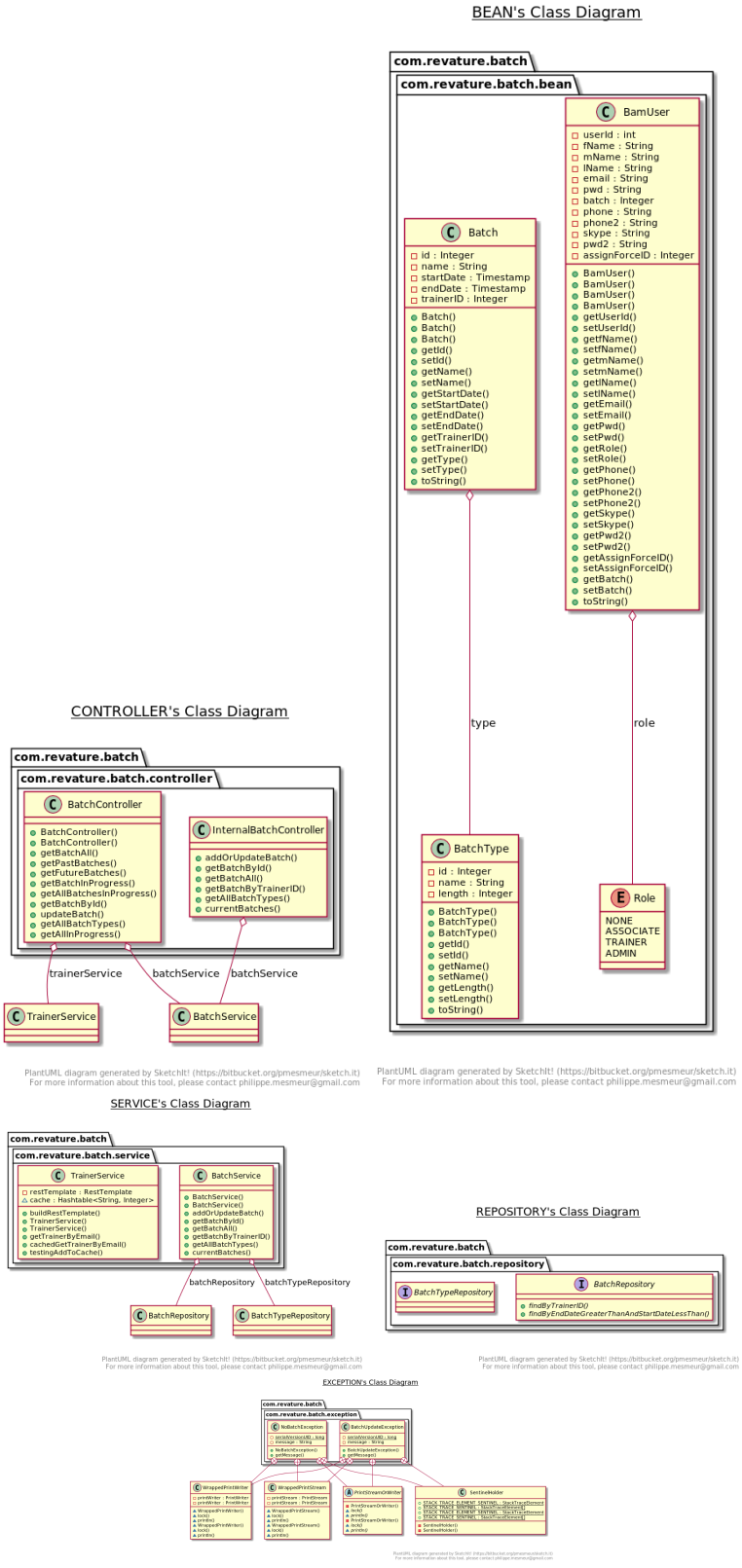


Figure 6.2: Batch Service Hierarchy

7

Topic Service

7.1 Abstract

The topic micro-service is a large component that handles subtopics, topics, and calendar models. The original monolith had each model separate along with a controller and service for each respectively. We broke down the code from the original monolith by the models, however, the topic and subtopic models were similar enough to merge them into one micro-service. By doing this, we reduced the number of overall micro-services we had as well as the internal service communication that would have had to have taken place. The calendar model was not originally found within the monolith, so we decided to simply add the routes to the topic micro-sevice, since the calendar is only composed of a topic and subtopic.

7.2 Service Architecture

Figure 7.1: Topic Service infrastructure

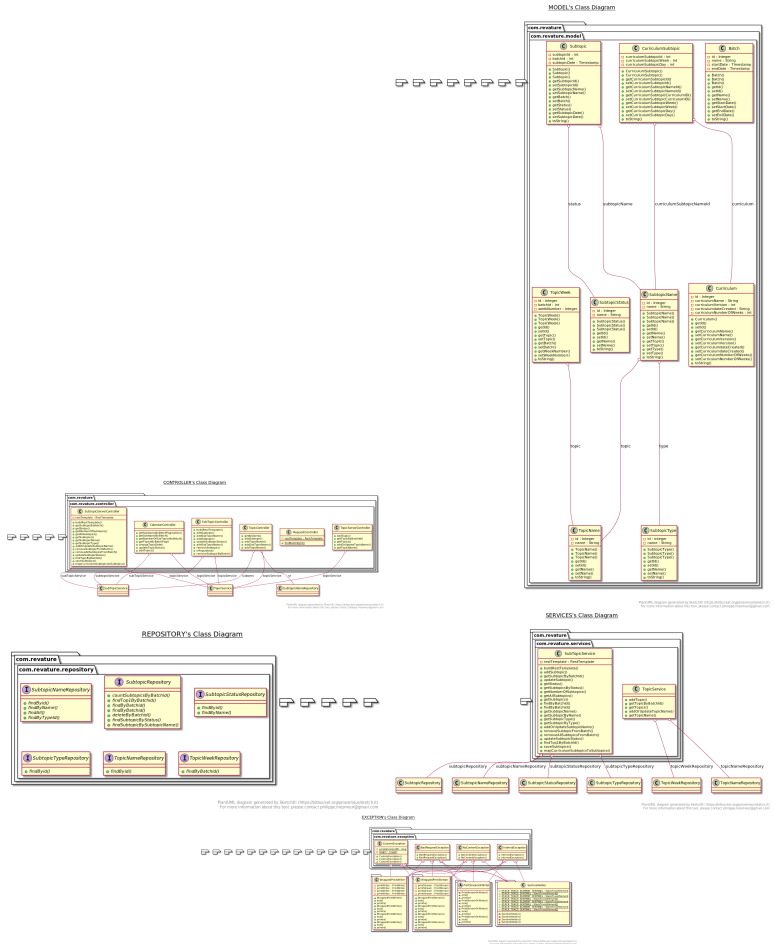


Figure 7.2: Topic Service Hierarchy

8

User Service

8.1 Abstract

The user micro-service is the basic component of your micro-service system. It is responsible for all user data and persisting it within a repository. Any other business service that needs user data must contact the user micro-service. The initial user model in the monolith application, receives data from a BAM user controller and stores it in the BAM user repository. In this application, we have gotten rid of the BAM user controller and the services.

8.2 Service Architecture

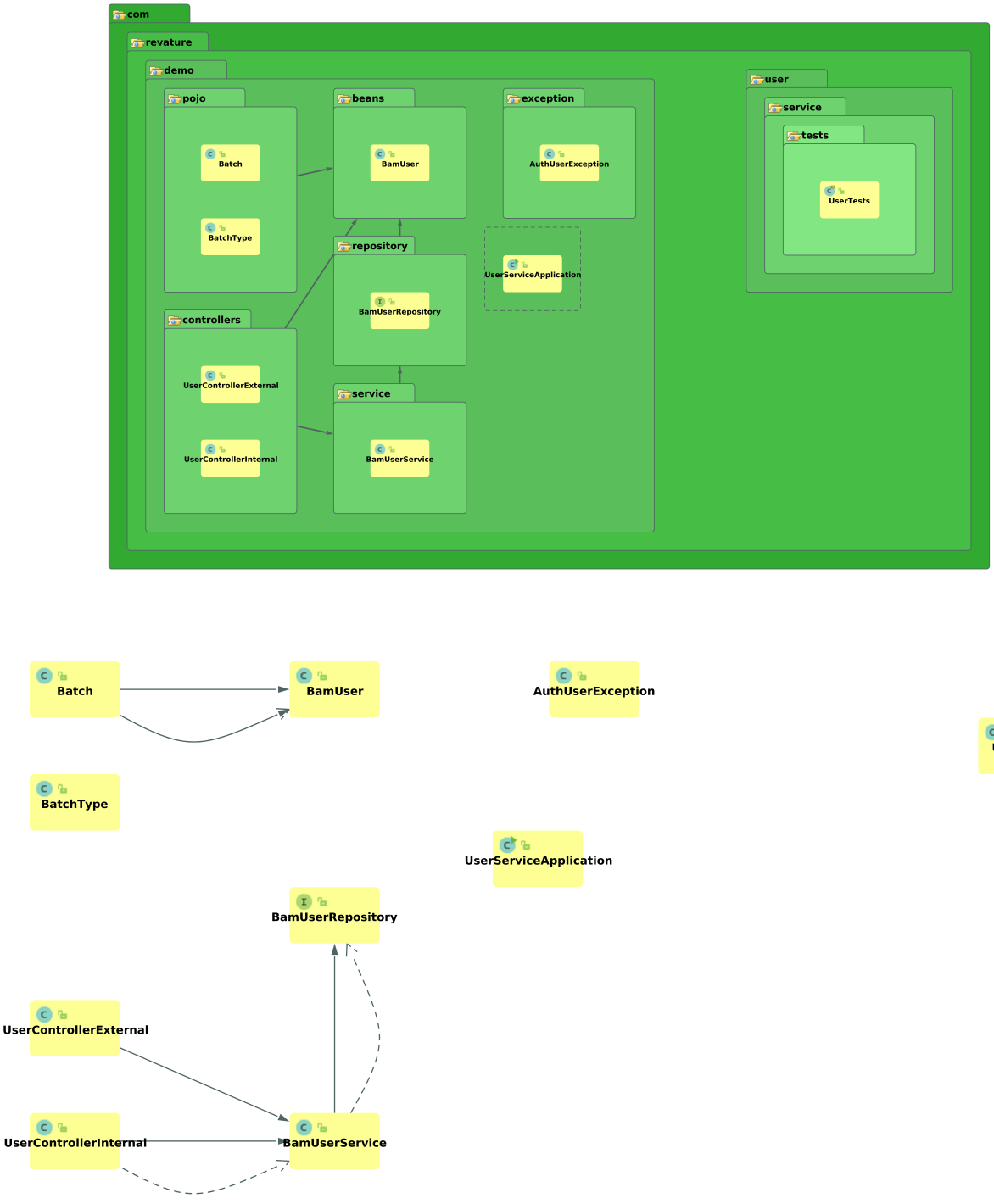


Figure 8.1: User Service Infrastructure

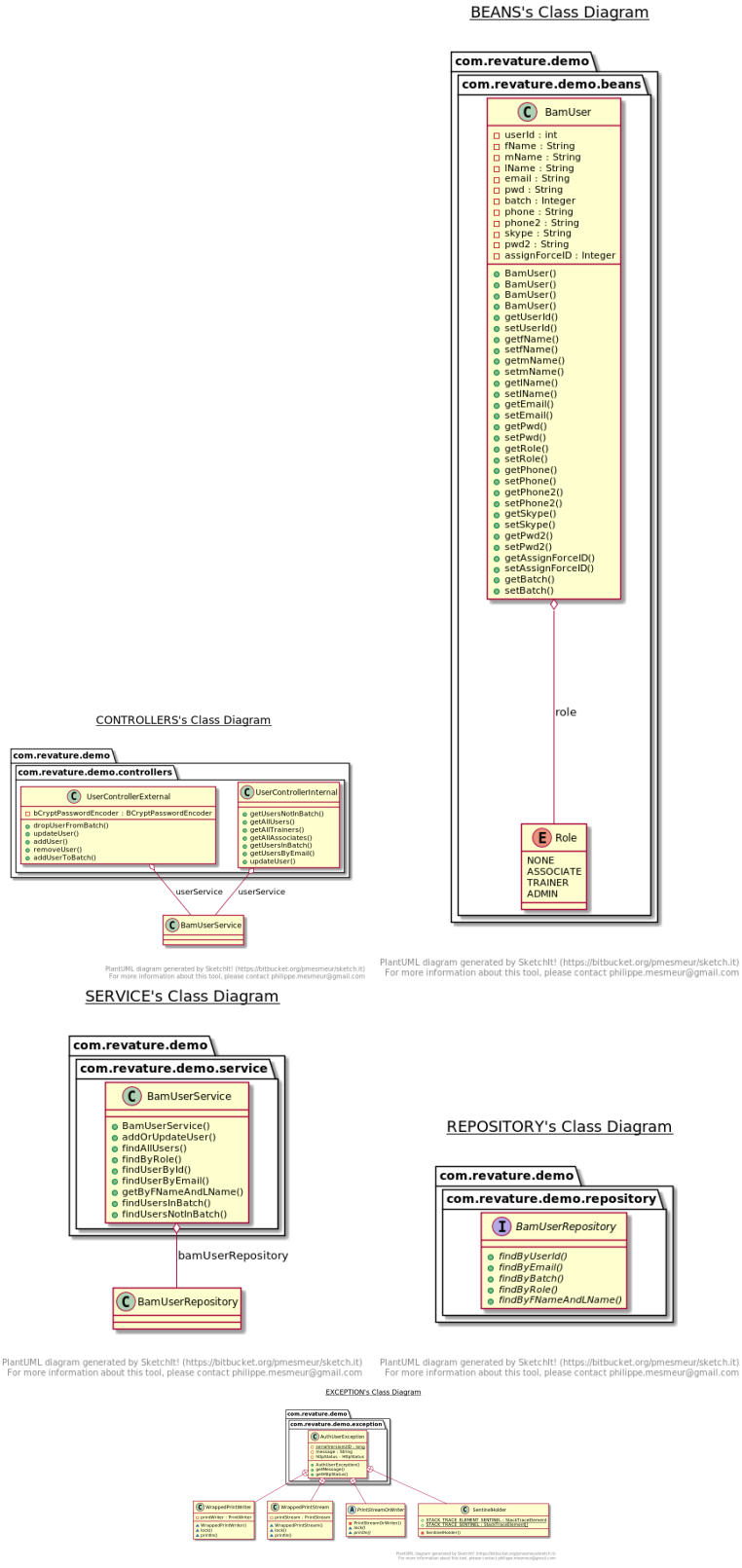


Figure 8.2: User Service Hierarchy

9

API routes

9.1 User Service API routes

9.1.1 Post request

- To drop user: /api/v2/user/drop/userId
- To update user: /api/v2/user/update/
- To register user: /api/v2/user/register/
- To remove user: /api/v2/user/remove/userId
- To add user to batch: /api/v2/user/addUserToBatch/userId/batchId

9.2 Batch Service API routes

9.2.1 Get requests

- To get all batches: /api/v2/batch/all/
- To get list of all past batches for the trainer: /api/v2/batch/inprogress?email=email
- To get all in progress by trainer: /api/v2/batch/allinprogress?email=email
- To get batch by id: /api/v2/batch/byid/id
- To get batch types: /api/v2/batch/batchtypes
- To get current batches: /api/v2/batch/currentbatches
- To get past batch by trainer email: /api/v2/batch/past?email=email
- To get future batch by trainer email: /api/v2/batch/future?email=email

9.2.2 Post requests

- To update or add batch: /api/v2/batch/updatebatch

9.3 Curriculum API routes

9.3.1 Get requests

- To get all curriculum: /api/v2/curriculum/all/
- To get curriculum by id: /api/v2/curriculum/getcurriculum/id
- To get curriculum by schedule: /api/v2/curriculum/schedule/id
- To get all topic names: /api/v2/curriculum/topicpool/
- To get all subtopics: /api/v2/curriculum/subtopicpool/
- To make master curriculum: /api/v2/curriculum/makemaster/id
- To synchronize curriculum with a batch: /api/v2/curriculum/syncbatch/id

9.3.2 Post requests

- To add curriculum: /api/v2/curriculum/addcurriculum/
- To delete curriculum version: /api/v2/curriculum/deleteversion/

9.4 Topic API routes

9.4.1 Get requests

- To get list of subtopics depending on what page: /api/v2/calendar/subtopicspagination/ba
- To get list of subtopics for batch: /api/v2/calendar/subtopics/batchId
- To get number of subtopics: /api/v2/calendar/getnumberofsubtopics/batchId
- To get topics by batch: /api/v2/calendar/topics/batchId
- To update status: /api/v2/calendar/statusupdate/subtopicId/batchId/status

9.4.2 Post requests

- To update calender date: /api/v2/calendar/dateupdate/subtopicId/batchId/date
- To add topic: /api/v2/calendar/addtopics

10

Challenges

10.1 Inadequate Documentation

We started off the project without any useful documentation. We only had a SQL script from the previous batches. The code Java code itself had java docs with no concrete definition of goal of the methods and classes.

10.2 Vague definition of business objectives

There was no stakeholder for the project, we started off the project by discovering what was the application supposed to do by looking at the code and experimenting with the application itself.

10.3 Revature's Given Services

In the beginning of the project, we were handed 4 micro-services that were supposed to kick start the project. They were the core micro-services: service registry and discovery, API gateway and configuration service. They are the projects: hydra-discovery-service, hydra-gateway-service and hydra-config-service. All three of them were never tested. The configuration service was using a gitlab repository that not only didn't have suitable configuration for our micro-services but also no fall back configuration. The API gateway service was copied from another project and had all the filters an swagger classes commented out.

10.4 Hardware limitations

Starting off the project, we weren't given any help from Revature. We weren't given enough hardware (EC2 memory) to run our micro-services, or

RDS databases or even the ability to create any users on the RDS database that was used in the original monolith.

10.5 Front-end Issues

The initial front-end wasn't not properly set with CORS, which means that we didn't have the ability to get data from the back-end using the given services. The front-end data models were identical to those of the back-end in the monolith, meaning that they were tightly coupled and most of the methods and models are redundantly used.

11

Solutions

11.1 Documentations

We started by writing documentation for next teams to work on this project, so they won't run into the same issues we had starting off the project. We have the document you're currently reading (Project design document) and every github repository has its wiki and milestones.

11.2 Core Services

We rewrote the core three micro-services: hydra-discovery-service, hydra-gateway-service and hydra-config-service. We also created configurations matching our micro-services names and fall back configurations. We also have setup the micro-services for containerization using Docker.

11.3 Hardware Limitations

We used our own EC2s and RDS to fill the hardware gap.

11.4 Front-end solution

We fixed the front-end models. They aren't tightly coupled anymore and we have fixed cores using HttpClient interceptors except for calendar service on the front-end because it was referenced everywhere though out the application and changing it in the time given would cause more problems than solve them. For calendar it starts having CORS problems 50 percent of the time.

12

Bugs

12.1 Front-end

The front-end services except for the interceptors need to redone to allow more flexibility and usage though out the application.

12.2 Configuration service

Because we had to use our own EC2 we couldn't make use of the configuration service. The configurations of the micro-services are tightly coupled.

12.3 Dashboard service

Because we ran through many hardware problems we couldn't configure hystrix-dashboard properly. The service hydra-dashboard needs to be redone or not used.

13

Potential Improvements

13.1 Use containerization

In micro-service architecture using containerization is ideal and save many debugging problems. Use docker swarm or PCF to configure and connect the micro-services.

13.2 Use Message Queue

In the current implementation we are using rest templates for internal service interaction. It is ideal for micro-service architecture to use streams to exchange data between services.

13.3 Add channel to Hystrix dashboard

Hystrix dashboard can used Turbine to track the health and circuit breaking of the services.

13.4 Use and Add YOUR custom configuration

Add your custom configurations to your configuration repository to match your architecture needs.

13.5 Front-end

Redo all of the services except the interceptors and everything will work flawlessly.