Module Code
COM00055H

BSc and BEng Degree Examinations 2023–24

DEPARTMENT OF COMPUTER SCIENCE

Engineering 2: Automated Software Engineering (ENG2)

Open Individual Assessment

Issued: October 5th, 2023, 12:00 (noon)

Submission due: January 18th, 2024, 12:00 (noon)

Feedback and marks due: February 22nd, 2024, 12:00 (noon)

All students should submit their answers through the electronic submission system: http://www.cs.york.ac.uk/student/assessment/submit/ by January 18th, 2024, 12:00 (noon). An assessment that has been submitted after this deadline will be marked initially as if it had been handed in on time, but the Board of Examiners will normally apply a lateness penalty.

Your attention is drawn to the section about Academic Misconduct in your Departmental Handbook: https://www.cs.york.ac.uk/student/handbook/.

Any queries on this assessment should be addressed by email to Dr. Antonio Garcia-Dominguez at a.garcia-dominguez@york.ac.uk. Answers that apply to all students will be posted on the VLE.

Rubric:

Your submission should include a report and an implementation. You must submit a single zip file containing your report in PDF format, and your implementation files. The report should use single-spaced lines using a sans-serif 11-point font (e.g. Arial). Submissions in a different format (e.g. a Microsoft Word document in a RAR archive), **will be penalised**. Note the page limits: parts of answers that go beyond the page limit may not be marked. Use IEEE citation and referencing: references must be listed at the end of the document and do not count towards the page limit.

Your exam number should be on the front cover of your assessment. You should not be otherwise identified anywhere on your submission.

1 The Exercise

You are to develop and document a data-intensive system, specifically a social media platform around video content. You will do this in two parts: 1) manually writing code for a subset of the functionality of your choosing, 2) completing the implementation by using model-driven engineering to automate repetitive tasks and link the architecture more clearly to the implementation.

1.1 Part 1: Data-Intensive System

Your data-intensive system will be a heavily simplified social media platform, where your users will post videos and like/dislike them, and these events will drive trending hashtags and subscription lists.

The system must include the following microservices:

- A video microservice (VM), with resources for posting videos (only user, title, and hashtags for this assessment), listing videos by user/hashtag, “watching” a video (marking it as viewed), and liking/disliking a video. VM will publish an event when a new video is posted, a video is liked/disliked, and when a video is watched by a certain user.
- A trending hashtag microservice (THM), with a resource for finding out the current top 10 liked hashtags within a rolling time window of 1 hour. THM will subscribe to the events from VM to automatically maintain the trending hashtags.
- A subscription microservice (SM), with resources to allow users to subscribe and unsubscribe from hashtags, and list the next videos to watch for a given hashtag and user. SM will use the events from VM to maintain a list of the top 10 next videos to watch for each subscription, and will publish an event when a user subscribes or unsubscribes from a hashtag.

You must document the overall architecture of the system (e.g. using appropriate diagrams), and implement these microservices using a combination of request-response and event-driven approaches. The system must be packaged as an orchestration of Docker containers (e.g. by using Docker Compose), where each microservice uses its own Docker image. You must write automated tests for your microservices, and show that you have scanned your containers for vulnerabilities and resolved any detected issues.

To keep this individual assessment manageable, we will assume some simplifications:

- You should not develop desktop or web-based interfaces for these microservices. Instead, you should write one or more Java command-line clients for these microservices. The command-line client(s) should allow for posting videos, liking/disliking them, showing the

currently trending hashtags, and listing the next videos to watch for the subscriptions of a user.

- You do not have to implement authentication or authorisation for the microservices.

1.2 Part 2: Application of Model-Driven Engineering

In the second part of this assessment, you will develop a domain-specific modeling language (DSML) which will assist you in automating the development of similar data-intensive systems.

The domain-specific modeling language should describe the events, event streams, and microservices in your architecture, at a high level of abstraction. It should include at least the following information:

- Events: fields and their types.
- Event streams: name, and type of event associated with it.
- Microservices: name, links to the event streams it subscribes and publishes into, and descriptions of its various API resources (e.g. HTTP method and request/response parameters).

You must use this DSML to describe your architecture, inspired by your architectural diagrams. You must develop a graphical concrete syntax for the DSML (e.g. in Eclipse Sirius or Picto) and show what your architecture looks like in this syntax.

The DSML must support automated validation. It must check these properties:

- There should be at least one microservice.
- Every event should be used in least one event stream.
- Every event stream needs to have at least one publisher and one subscriber.
- Every microservice needs at least one “health” resource using the HTTP GET method and taking no parameters, for reporting if it is working correctly.

Automated generation of code from the DSML must be implemented, e.g. by creating a scaffold for the implementations of the microservices, and/or assisting its containerisation (e.g. by generating Dockerfiles or Compose YAML files). You must show that the newly generated code is functionally equivalent to the previously manually-written code.

2 Questions

Answer **all** questions. Note the page limits for each question. Parts of answers that go beyond the page limit will not be marked. References must be listed at the end of the document and do not count towards page limits.

2.1 Part 1: Data-Intensive Systems [60 marks]

2.1.1 Architecture

Define the overall architecture using recognised notations (e.g. UML component/deployment diagrams or C4 diagrams). Justify how the architecture can scale with increasing user demands, and be adapted to new requirements in the future (e.g. a recommendation system). [15 marks, max 2 pages]

2.1.2 Microservices

Implement the above microservices using the technologies presented during the module's lectures and practicals. Each microservice should be separately deployable and scalable, and should be accompanied by a command-line client to exercise its functionality. Microservices should be able to run locally without requiring any cloud-based resources: any required persistence solutions (e.g. a database) should run locally. Document the high-level design of these microservices, and the appropriate usage of your command-line client. [20 marks, max 2 pages]

2.1.3 Containerisation

Package and deploy the system as a set of Docker containers, orchestrated through the technologies presented during the module's lectures and practicals (e.g. Docker Compose). Discuss how the solution can scale up to larger numbers of users, and be resilient to failures (e.g. of a container, or a node). [10 marks, max 1 page]

2.1.4 Quality Assurance

Perform testing for your microservices through an appropriate combination of approaches to manage risk. Give a brief report on the actual tests, including statistics of what tests were run and what results were achieved, with a clear statement of any tests that are failed by the current implementation. If some tests failed, explain why these do not or cannot be passed and

comment on what is needed to enable all tests to be passed. If no tests failed, comment on the completeness and correctness of your tests instead. [12 marks, max 2 pages]

Inspect all Docker images you used (both your own, and any others you use) for security vulnerabilities. If you find any security issues, show how you have dealt with them, or justify why this was not feasible with the time available. [3 marks, max 1 page]

2.2 Part 2: Application of Model-driven Engineering [40 marks]

2.2.1 Metamodel

Use Emfatic/Ecore to define a metamodel for the domain-specific language described in Section 1.2. Include a class diagram of the metamodel in your report, discuss the metamodel, state any assumptions you have made, and explain any alternative design decisions that you have considered and discounted. [8 marks] (max 2 pages)

2.2.2 Graphical Concrete Syntax

Use Eclipse Sirius or Picto to define and implement a graphical concrete syntax for the metamodel you defined in Question 2.2.1. Design the model of your architecture in your language, provide a screenshot of the model in your concrete syntax, discuss and justify your syntax design and implementation decisions, and reflect on the strengths and weaknesses of the selected concrete syntax compared to alternatives. [12 marks] (max 2 pages)

2.2.3 Model Validation

Use the Epsilon Validation Language to implement any validation constraints specified in Section 1.2, which cannot be expressed in the metamodel itself. Briefly explain the rationale and implementation of each constraint. [5 marks] (max 1 page)

2.2.4 Model-to-Text Transformation

Use the Epsilon Generation Language to implement a model-to-text transformation that consumes a model that conforms to your DSML, and produces some of the code that you wrote manually in Part 1. Here are some suggestions:

- Scaffolds for the implementations of the microservices and the clients, with clear separation between automatically generated code, and hand-written customisations to write the detailed behaviour.

- Custom message beans for Kafka, implementing your events.
- Container orchestration for your application (e.g. Docker Compose files).

Discuss the model-to-text transformations and justify the organisation of the generated code. [15 marks] (max 2 pages)

3 Submission Instructions

You must submit a single zip file containing your report in PDF and your implementation files. You must not identify yourself. For example, you must not name files with your IT services (or any other) username!

When expanded, your zip file should contain folders and files matching the structure and naming conventions described below. You must replace "Y1234" with your own examination candidate number:

- Y1234.pdf (your report)
- microservices (a directory containing your microservices implementation)
  - docker-compose.yml (should be able to bring up your system with a single `docker compose up` command)
  - video-microservice (a directory with your VM implementation)
  - trending-microservice (a directory with your THM implementation)
  - subscription-microservice (a directory with your SM implementation)
  - client (a directory with your command-line client(s) for the microservices)
- modeling (a directory containing your application of model-driven engineering)
  - metamodel (a directory containing your metamodel files and any Sirius/Picto-related files)
    - Y1234.emf or Y1234.ecore (your metamodel, in Emfatic/Ecore)
      - The name and namespace URI of the root package of your metamodel should also be Y1234.
    - Y1234.evl (your EVL validation constraints)
  - m2t (a directory containing your model-to-text transformation)

  - Y1234.egl or Y1234.egx (the entry point for your model-to-text transformation)
  - *.egl (any other templates for your model-to-text transformation)
- models (a directory containing your models)
  - acme.aird or acme.model or acme.Y1234 or acme.flexmi (your model of your system)
  - any other models you may have developed to test your editor, viewer, constraints or transformations
- configs (a directory containing stored launch configurations for your model validation constraints, and your model-to-text transformations)
  - Use the naming convention “Y1234-Description.launch”
  - You should generate the code directly into the relevant `microservices` subfolder: it is recommended to organise your microservices code to clearly separate manually-written and generated code (e.g. by keeping them in separate folders)
  - See the section titled “Epsilon launch configurations” on the following webpage for some guidance on how to persist launch configurations: http://www.eclipse.org/epsilon/doc/articles/minimal-examples/

End of examination paper