SERVICE ORIENTED ARCHITECTURES

ACIT3855 – WINTER 2024

AGENDA – LESSON 9

- Assignment 1
- Quiz 8
- Topics:
  - Issue Fixing
- Lab 8 – Demos
- Lab 9 – Issue Fixing

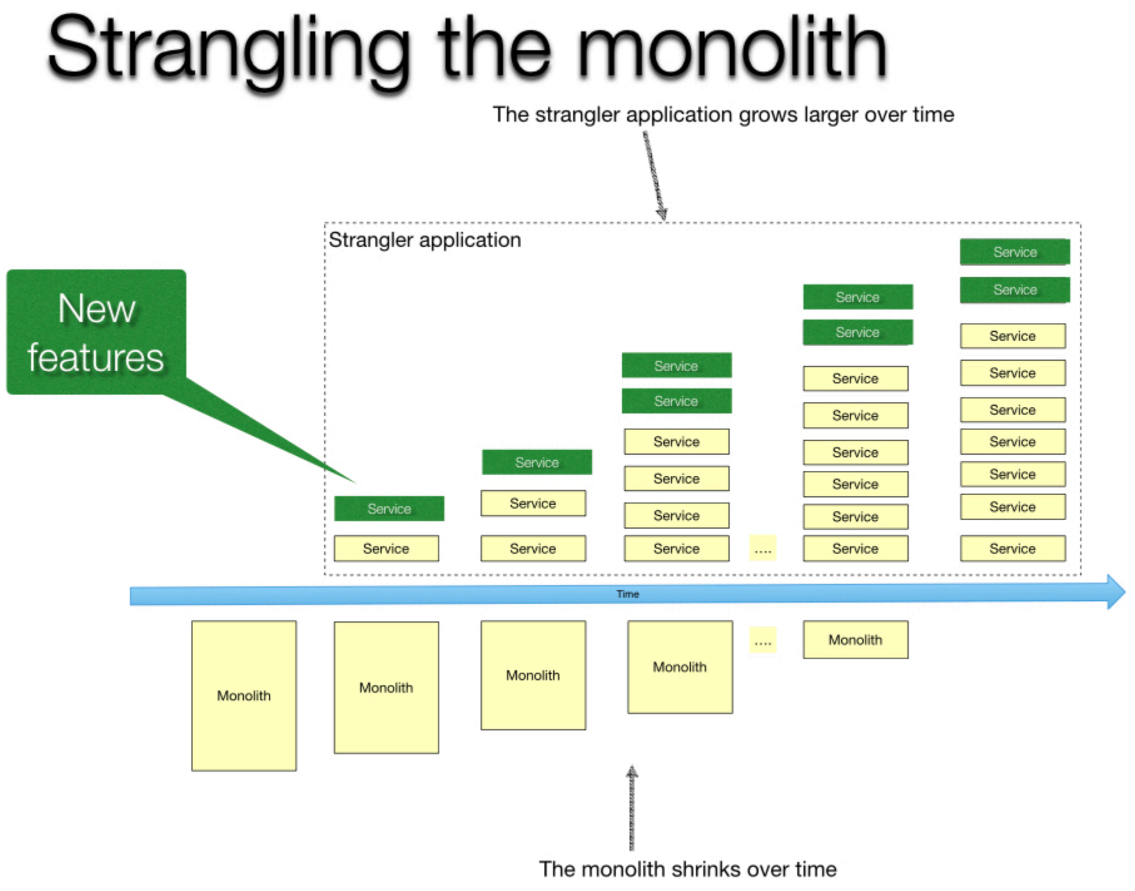
COURSE SCHEDULE

Week	Topics	Notes
1	• Services Based Architecture Overview • RESTful APIs Review	Lab 1
2	• Microservices Overview • Edge Service	Lab 2, Quiz 1
3	• Database Per Service • Storage Service (SQLite)	Lab 3, Quiz 2
4	• Logging, Debugging and Configuration • Storage Service (MySQL)	Lab 4, Quiz 3
5	• RESTful API Specification (OpenAPI) • Processing Service	Lab 5, Quiz 4
6	• Synchronous vs Asynchronous Communication • Message Broker Setup, Messaging and Event Sourcing	Lab 6, Quiz 5
7	• Deployment - Containerization of Services *Note: At home lab for Monday Set*	Lab 7, Quiz 6 (Sets A and B) , Assignment 1 Due
8	• Midterm Week	Midterm Review Quiz
9	• Dashboard UI and CORS	Lab 8, Quiz 7
10	• Spring Break	No Class
11	• Issues and Technical Debt	Lab 9, Quiz 8
12	• Deployment – Centralized Configuration and Logging	Lab 10, Quiz 9
13	• Deployment – Load Balancing and Scaling *Note: At home lab for Monday Set*	Lab 11, Quiz 10 (Sets A and B)
14	• Final Exam Preview	Quiz 10 (Set C), Assignment 2 Due
15	• Final Exam	

ASSIGNMENT 1 - INSIGHTS

- What are the key features of a Microservices Architecture?
- When would you choose a Microservice Architecture?
- What types of companies are using a Microservices Architecture?
  - Why?
  - How do they usually start out?

MONOLITH TO MICROSERVICES – THE STRANGLER PATTERN



Gradually refactor your monolithic application to microservices in a phased approach.

This is the most common path to an MSA.

DASHBOARD USER INTERFACE

Dashboard UI

- Contain information such as stats, analytics, schedules, messages and much more
- Typically dynamic – changes as the system data changes

Single Page Application (SPA)

- Dynamically rewrites the page in the browser based on user input and/or requests to a backend
- Typically developed using a JavaScript framework like Angular or React
- Creates a more traditional desktop experience to the user as the page is not completely reloading after each action.

SOFTWARE ISSUES

- Also known as a Software Bugs
- Errors, flaws or faults in a program that causes it to produce an incorrect or unexpected result or behave in unintended ways. All software programs have issues, but the severity of those issues varies greatly and not all are always fixed.
- The process of finding and fixing them is called debugging.

It is likely that a large portion of your work in Software or IT will be debugging.

Debugging can be both fun and frustrating. And sometimes stressful too (i.e., when close to a due date)

TECHNICAL DEBT

- Also known as design or code debt.
- It is the implied cost of additional rework caused by choosing an easy (limited, suboptimal) solution now instead of using a better approach that would take longer.
- Technical Debt is typically represented as a type of software issue – either improvements or bugs causes by the easy solution.

Typically time pressure or lack of knowledge results in Technical Debt.

Technical Debt is not always bad if it lets you get your software out to market or the customer within your project constraints. For long lived software products, it needs to be managed.

QUIZ 8

- Open Book
- You have 15 minutes to complete it
- Then we will go through our current set of software issues and technical debt

TRIAGE OF SOFTWARE ISSUES

- Triage term is used in the Software testing / QA to **define the severity and priority of new defects**.

- Both developers and testers raise software issues during development of a software product.
  - This includes both bugs and technical debt.

- On an ongoing basis, the team or the team leads triage these new software issues in the project management tool (i.e., JIRA)
  - Usually more frequently near a software release when there is active testing and issue fixing happening

- This process determines which software issues will be fixed and when.

REFACTORING

- Often Technical Debt is addressed by **Refactoring**.
- Refactoring is re-implementing a feature, capability or component where the functionality stays the same:
  - The what is largely unchanged
  - The how is changed
- Basically it is replacing one implementation with a new implementation that is *hopefully* more robust or perhaps less "buggy".

PRIORITY AND SEVERITY OF ISSUES

- **Priority** – How urgently an issue should be addressed? Can be subjective.
- **Severity** – What is the impact of the issue? Can be objective.

Priority is the *order in which the developer should resolve a defect* whereas Severity is the *degree of impact that a defect has on the operation of the product.*

PRIORITY EXAMPLES

- High
- Medium
- Low

The higher the priority, the sooner the issue should be fixed.

SEVERITY EXAMPLES

- Critical – Blocking, there is no workaround
- Major
- Moderate
- Minor
- Cosmetic

Note: Sometimes an issue with a Cosmetic severity can have a High priority. Typically this is when the issue is important to the business.

REPORTING ISSUES

- Make sure you record a good set of detail when reporting issues. Typically a single sentence isn’t enough.
- Some items you should record:
  - Observed Behavior
  - Expected Behavior
  - Steps to Reproduce the Issue (if you can)
  - When the Issue was Observed (which build, release)
  - Impact of the Issue

ISSUE 1 – DASHBOARD UI: AUDIT LOG INDICES OUT OF SYNCH

- **Observed**: The displayed Audit Log index and the corresponding Event are out of synch. The index is ahead of the data (i.e., the event data is for the previously displayed index).
- **Expected**: The Audit Log index and data should be synchronized.

What is the Severity of this Issue?

What is the Priority of this Issue?

ISSUE 2 – KAFKA TOPICS AREN'T PERSISTING

- Observed: The events topic in the Kafka broker does not persistent when Kafka is brought back up, such as when you stop your VM, start it again and then run "docker-compose up –d".

- Expected: Kafka topics are meant to persist, similar to our MySQL database.

What is the Severity of this Issue?

What is the Priority of this Issue?

ISSUE 3 – STORAGE SERVICE DOES NOT CONNECT ON STARTUP

- Observed: Storage Service sometimes does not connect to Kafka on startup, requiring the service to be stopped and restarted.
- Expected: The Storage Service should be able to connect to Kafka on startup, even if it has to wait until Kafka is up and running.

What is the Severity of this Issue?

What is the Priority of this Issue?

ISSUE 4 – RECEIVER SERVICE RECONNECTS TO KAFKA FOR EACH EVENT

- This is more like Technical Debt. The software works now, but we believe it could be improved to increase performance.

- Hypothesis: Currently each time an event is received at the Receiver Service, a KafkaClient object is created which need to establish a connection to the Kafka service. If we establish this connection once when the Receiver Service is started, then we don't have to create a new connection each time an event is received and will be able to publish the message to the Kafka topic quicker. This will result in a faster response back to the client.

What is the Severity of this Issue?

What is the Priority of this Issue?

ISSUE 5 – STORAGE SERVICE LOSING KAFKA CONNECTION

- Observed: After a period of time, when jMeter is used to send in events to the system, the Storage Service will receive the first event and then not process any subsequent events until it is stopped and restarted.

- Expected: The Storage Service should process events indefinitely until it is stopped or the Kafka broker goes down.

- Problem: MySQL connections are not configured correctly. Look into settings like pool_recycle and pool_pre_ping on the SQLAlchemy create_engine function

TROUBLESHOOTING TIPS IN MICROSERVICES

- First step in debugging is to be able to consistently reproduce the issue
  - This may only be possible in a test or production like environment
  - Don’t assume that if you can’t reproduce it in your own development environment it doesn’t exist!

- Logging is your friend when troubleshooting microservices, especially in your test and production environments
  - Make sure you have a good set of log messages, add more if you need to help you troubleshoot
  - Establish a set of rules for use of each of the logging levels
  - Write key information to the logs, like event ids, to enable Distributed Tracing

TROUBLESHOOTING TIPS IN MICROSERVICES

- Have a 3rd party (i.e., tester, peer) review and test your fixed issue

- Once you've fixed an issue, record:
  - How to reproduce it
  - What you did to fix it
  - This will help the 3rd party tester and code reviewer

- Make sure manual or automated tests are added that try to detect this issue in future

TODAY'S LAB

Today you will:

- Demo Lab 8 by the end of class
  - Show the Dashboard UI working both locally and on your VM
- Work on Lab 9 – It is due for Demo next class