4th Year Project – Computing Systems and Operations

Introduction

The 4th Year Project is the capstone of your undergraduate degree in Development and DevOps. It is your opportunity to integrate everything you have learned across the programme and apply it to a substantial, real-world style project.

Your task is to design, build, and deliver a working software solution that is supported by a production-quality DevOps pipeline and operational platform. This means your project is not just about writing code, but about demonstrating how modern software is delivered, deployed, and operated.

The project mirrors professional practice. You will begin by identifying a real user problem, progress through design thinking and prototyping, and then implement a complete solution using CI/CD pipelines, infrastructure as code, observability, security, and resilience.

This project is also your chance to showcase your abilities to future employers. The technologies you choose, the practices you adopt, and the way you communicate your work will all highlight your readiness to operate as a professional software engineer and DevOps practitioner.

Requirements

You need to fulfill two components to your project:

- Develop a working and useful product
- Build a Devops platform

DevOps Platform Requirements

To meet the DevOps requirements for the platform, your project should include the following:

- **Continuous Integration (CI)** with automated testing and quality gates.
- **Provisioned infrastructure** using Infrastructure as Code to provision Cloud environments, networks and systems*.
- **Continuous Deployment/Delivery (CD)** with at least one advanced deployment strategy.
- **Operational platform** on cloud-native technologies (containers, Kubernetes, or serverless).
- **Observability** – monitoring, logging, dashboards, and alerting.
- **Security practices** – secrets management, vulnerability scanning, least privilege access, network controls.
- **Resilience evidence** – fault tolerance, scaling, recovery from failures.

* You may consider public Cloud or private Cloud environment, such as XOA.

Defining Scope of Your Product

The product can be a topic of your choosing using any technologies, but should broadly demonstrate your skills you have learned within the course and your proposal will need to be agreed with the lecturer. It is important to give this due consideration to be successful.

Below are some guidelines to consider:

- **Picking an objective for your project**
  Choose a problem or domain you are passionate about (or at least think you will enjoy). Your objective should be concrete enough to deliver within two semesters but meaningful enough to demonstrate value to a real or potential user.

- **Identify a user requirement**
  Base this on a genuine need – ideally drawn from placement experience, personal interest, or a domain aligned with your career path. Think about who the "user" is (e.g., developer, IT operator, business user) and what pain point you are solving.
- **Consider technologies you would like on your CV**
  Select technologies that will stretch your skills and be relevant to future job roles (cloud platforms, DevOps tools, programming frameworks, AI integration). Ensure they support your project goals.
- **Investigate existing solutions and identify gaps or opportunities**
  Review tools, platforms, or products that solve similar problems. Identify what they do well, where they fall short, and how your solution offers a unique or improved value.
- **Define a unique value proposition for your solution**
  State why a user would care about your solution. Focus on the benefit to the user (faster, simpler, cheaper, more reliable, more secure).
- **Make your objective realistic and achievable in 2 semesters**
  Define a Minimum Viable Product (MVP) that you can commit to delivering well within the timeframe. Then identify some additional goals you may wish to iterate to extend the application over time. These may evolve as you prototype your proposed solution, but its important to start with a realistic goal.

Gathering Data from Users - Ethics

If can you identify real (or potential) users for your project you can conduct user surveys, testing and get feedback to enhance your prototype development. These may be other students, friends, someone from your placement or elsewhere.

This is a good way to gain experience of working with real users and requirements and is encouraged.

If you do decide to work with other people to develop your project requirements and you are gathering data you **must seek approval from the Ethics Committee** to ensure you follow all required guidelines for privacy, security etc. Talk to your lecturer to ensure you follow the procedure to seek approval early in the semester. There is a committee that runs in the first semester that reviews all ethics requests and need to meet that deadline.

Stages of the Project

Once you have chosen the topic for your project will go through two stages:

- Stage 1: Scope, design and prototype
- Stage 2: Implementation

Stage 1 – Design and Prototyping

The first stage of the project is to define the product you will deliver and a proof of concept of the Platform it will run on.

You will do this through investigation, prototyping and testing. **Design Thinking** is an increasingly popular approach in industry and is very effective at rapidly producing solutions and understanding challenges early.

Here is an outline of the Design Thinking approach to developing a prototype for your project.

- **Empathize – understand user's needs and context**
  Conduct interviews, surveys, or research to understand your target user and their challenges.
- **Define – state the user's problem clearly**
  Translate insights into a specific, testable problem statement.
- **Ideate – generate and evaluate solution options**
  Brainstorm multiple approaches, considering trade-offs in technology, usability, cost, and maintainability.
- **Prototype – build and iterate on a working prototype**
  Create a simple version of your solution to demonstrate key ideas. Focus on usability and user experience.
- **Test – validate the prototype with user feedback**
  Present your prototype to peers, mentors, or potential users. Refine iteratively based on feedback.

Start by taking your MVP scope and iterating to create a prototype. Then you can prototype subsequent enhancements to prove out more functionality you would like to achieve.

In semester 1 you will also design your platform. This needs to be a proof of concept, so going into the 2nd semester you are ready to implement both the platform and product.

The platform proof of concept should consider the following:

- Source code and pipeline tools (github, gitlab, circleci, etc).
- Dev, test and production environments
- Cloud environment to be used (or on-prem/hybrid if appropriate)
- Infrastructure design - network, servers, storage etc.
- Application framework - Docker, Kubernetes, Serverless etc.

- Provisioning toolset - Ansible, Terraform etc.
- Deployment tools - GitOps tools or others
- Logging and monitoring tooling

You should already have a knowledge of a lot of the tools in this space, but if there are new ones you wish to use because they offer certain benefits, be sure to test them out so there are no surprises later. Create a proof of concept that shows your basic processes working correctly.

Deliverables for Stage (30%)

- Documentation of objectives, user problem, Value Proposition.
- Log of Prototype build and iteration process.
- Documentation of investigations, final MVP functionality and design.
- Prototype Demonstrations - mid-term (MVP) and end of semester.
- Design and Proof of Concept of core DevOps Platform

You will need to have completed and submitted your stage 1 deliveries by the end of the first semester.

Use Jira and Confluence to capture ongoing design work, investigations, results, activity and progress. Use source control to keep track of your code.

Stage 2 – Implementation

Now you have a target MVP and a proof of concept platform, you can start working on your implementation.

Below is a list of the areas you should ensure are part of your process and platform:

- **Use Scrum or Kanban to drive project management**
  Work in sprints or iterations. Track progress in Jira.
- **Track work and document decisions**
  Use Confluence and Architecture Decision Records (ADRs) to document rationale.
- **Implement a production-quality solution with CI/CD**
  Build automated pipelines for testing, building, and deploying your code. Integrate quality gates.
- **Provision infrastructure using Infrastructure as Code**
  Define infrastructure (VMs, containers, networks, secrets etc) in reproducible scripts (e.g: Linux Scripting, Terraform, Ansible).

- **Deploy to development, test, and production environments**
  Promote releases across environments with toolsets such as GitOps and using strategies like canary, blue/green, or rolling deployments
- **Implement observability – logging, metrics, dashboards, alerting**
  Define Service Level Indicators (SLIs) such as latency, error rate, and availability. Configure dashboards and alerts.
- **Apply DevSecOps practices**
  Integrate code scanning, dependency scanning, secrets management, and least-privilege access.
- **Test functional and non-functional performance**
  Verify correctness, scalability, resilience, and recovery under stress conditions.

Deliverables for Stage 2 (40%)

- Project Documentation (see below)
- Full product and platform implementation in source control.
- Automated CI/CD pipeline.
- Deployment strategy, documented and demonstrated.
- Provisioned environments (dev/test/prod).
- Demonstrations of:
  - Product Functionality.
  - DevOps platform - environments, build/release processes and pipeline.
  - Observability (live dashboards/logs).
  - Resilience under failures and load.

Note it is important you can demonstrate your CI/CD process end to end. To do you must release your MVP by the middle of the second semester and then demonstrate subsequent release process. This can be a continuous delivery or continuous deployment release.

Project Documentation

Documentation should include the following information:

- Final application design and implementation details.
- User documentation.
- DevOps Platform design and implementation
- Deployment strategy
- Runbook / operational guide.
- SLOs, SLIs

- Comprehensive Test results (functional and non-functional).
- Security measures implemented.

Project Log and Reflection (20%)

To be undertaken by each student separately, to demonstrate contribution and reflection.

- Log of progress (captured in Jira/Confluence) - this should capture your investigations, design decisions and week by week progress. This should capture your contribution to the project.
- Reflection: lessons learned, challenges faced, and future improvements.

Final Project Presentation (10%)

- Prepare slides and poster for your project
- Present your project at the end of year
- Answer detailed questions about your project.

Successful projects will demonstrate both technical depth (in the design, implementation, and automation of your platform) and professional breadth (in the way you analyse, document, and present your work).

Completing this project will give you the chance to showcase your ability to think, build, and operate like a professional software engineer and DevOps practitioner — and demonstrate to future employers that you are ready to contribute to modern technology teams.

Schedule

Stage 1 - Delivery Schedule

Delivery	Schedule	Percentage
Documentation of objectives, user problem and Value Proposition.	Week 3	5
MVP prototype demo	Week 9	10
Documentation of final MVP functionality and design.	EoS 1	10

Platform Design document and demo.	EoS 1	10

Stage 2 - Delivery Schedule

Delivery	Schedule	Percentage
Provisioned platform and MVP deployment with logging	Week 9	10
CI Pipeline demonstrating build and test automation	week 9	10
CD pipeline demonstrating automated Project deployment with Iterations	EoS	10
Application Monitoring and non functional testing (fault tolerance and scaling).	EoS	10

Each delivery needs to be a practical demonstration with accompanying document to describe design, implementation and testing.

Personal Documentation

Delivery	Schedule	Percentage
• Log of work and progress	EoS 1	5
• Log of work and progress	EoS 2	5

• Reflection: lessons learned, challenges faced, and future improvements.	EoS 2	10

Appendix

Tips and Points for Consideration

Here are a some considerations, tips and suggestions for the various stages of your project.

Technical Considerations

Some of technical aspects you should consider as you develop your project.

- **Storage** – type, scalability, and resilience.
- **Hosting environment** – cloud, on-prem, or hybrid; justify your choice.
- **Security** – authentication, authorization, secrets management, vulnerability scanning.
- **Multi-user / multi-tenant support** – design considerations if relevant.
- **Tooling** – justify choice of CI/CD, monitoring stack, and IaC tools.
- **Testing strategy** – unit, integration, load, resilience, and security testing.
- **Collaboration practices** – Git branching model, code reviews.

Prototyping

During this phase establish a clear objective as early as possible and then iterate through the Design Thinking process multiple times to develop solutions. Evaluate your results against some kind of criteria.

Investigate tools, libraries and frameworks that you want to use and rapidly build prototypes to try these out to see how they effective they are for your use case.   Do some investigation to find what technologies are available, create a shortlist of options that offer distinctly different benefits and try them out. Don't spend too long on this. Have a criteria to evaluate against and document your results.

It is likely technologies you are familiar with will allow the fastest progress, but you may have reasons for investigating other technologies, for instance if they offer a significant benefit or if it there is a new technology you wish to learn. For instance you may want to use the opportunity to implement AI Ops tools.

A few tips for prototyping:

- Evaluate your prototypes from multiple perspectives - functionality, usability, security, multi tenancy etc - put yourself in the position of an end user.
- Build the bare minimum to test out ideas, functionality and technologies - move FAST and iterate often. Build out step by step and test as often as you can (remember code a little, test a little).
- Focus on what you don't know or understand first - a big value of prototyping is it allows you to learn the unknowns early and well before committing to the main delivery. Don't just build what you know how to do - this is a waste of time.

Implementation

Select your target environments (e.g. Cloud, on-prem etc). Ensure they are provision from the outset so you can easily reconstruct and replicate them. You will need at least a test and prod environment, but possibly others.

You will need to select the tooling you wish to use for your pipeline and environments. Try these out to check they will all work together as expected and manage your process and environment as expected. You may also want to investigate some AI Ops tools.

Make sure you have decided early on what your CI/CD strategy is going to be - Continuous Delivery or Continuous Deployment. If Continuous Delivery, then you will need to have release strategies such as Canary or Green/Blue. If you implementing Continuous Deployment you will need ensure you have mitigated risks e.g. comprehensive automated testing process and gates. Its important you design this upfront and do not over engineer your solution and lose time.

It may be tempting to continue developing your prototype code, but it is often better in the long run to build a production release from scratch (based on the design of your prototype). This is because you will likely have chopped and changed your prototype a lot it may not be a very 'clean'. Ensure you have followed best practices by using classes, design patterns, logging, exceptions, security, testing etc. for production quality code. AI coding tools are likely to be productive at this stage.

Make good use of project management techniques and Jira to schedule and track your development. Set clear weekly targets on what you intend to achieve to avoid burn out! Work to a regular release schedule.

You may use a public Cloud environment, such as Azure, AWS, GCP etc. or an on-prem ‘Cloud’ VM environment e.g. XOA. Each has pros and cons. Public Cloud has many services, preconfigured control planes but has limited free usage. XOA is free always, but will require more provisioning. You will need setup a control plane with multi-node configuration and manage networking, security, storage etc. This is more work but will give you more in-depth understanding of infrastructure and unlimited access (in theory). Terraform does have a Xen module, but you would need to experiment to see if this can be made to work.

Using AI Coding Tools

I am assuming you will need to use AI coding tools to complete everything on this project brief, so here are some thoughts on making the most of these tools. There are plenty of other resources for this, so get as much info as you can to get productive.

However you produce you code, you need to be able to answer questions about the code and documentation you have submitted. As always, **if you cannot explain something you have submitted, you will not get marks for it.**

Here are some suggestions for using tools:

- There are a growing number of coding tools - VSC with Copilot, Cursor, Windsurf, Claude, Gemini etc. This is a good opportunity to improve your skills with these tools to enhance productivity. I suggest you decide early what tool(s) you are going to use and stick with them. Make sure you use appropriate tools for the job.
- You can experiment with vibe coding in the prototyping phase, but remember you need to understand what has been generated so you can turn it into production quality code later. You will also need to iterate and that can be much harder if you don’t understand your code.
- **Don't** over rely on AI coding tools. As a rule, use AI tools to do what you already know how to do, which can significantly enhance your productivity. Remember you will need to change and evolve your code rapidly and if you don’t really understand what’s been coded this will cost you much more time than it saves later on.
- AI is very good for: creating well defined functions and classes, code snippets, unit tests, code based on design patterns. The more specific and detailed you can the better the outcome.
- AI tools are increasingly good at reading and learning context and looking at multiple files. Make sure you are context aware and ensure only the necessary files are in the context.
- AI tools still make mistakes and can’t always fix them. You will need to help out on occasions, so once again, make sure you understand the code being generated. You can ask it to fix a

problem, but if it doesn't get it pretty quick I suggest you dig in.

- AI tools have been known to decide to rewrite code when you don't expect and you can loose functionality and code you have created. This only happens occasionally and probably getting better but it can be bad when it does. My recommendation is constantly commit changes the tool has made so you can revert to the previous version if it suddenly "looses the plot".
- Chat is generally more productive than inline when you are making larger changes, but sometimes its quicker just to get in and make code changes directly (so long as you know your code). Alternate between these different modes to get the most productivity.
- You can ask chat questions as well as get it to generate code, but make sure you are clear as sometimes it can make unexpected changes.
- You can ask copilot to explain code inline. This can be useful, particularly if you are not sure about the code that's been generated.