# Quality View Workshop Character Sheets

(intentionally left blank)

## Penelope, Project Manager

Penelope is an experienced project manager who has been working at the University for a long time. She manages different types of projects and is known for her self-confidence and tendency to be pushy. She worked on projects related to the central registry of the university and has a hard deadline of six months to complete this current project.

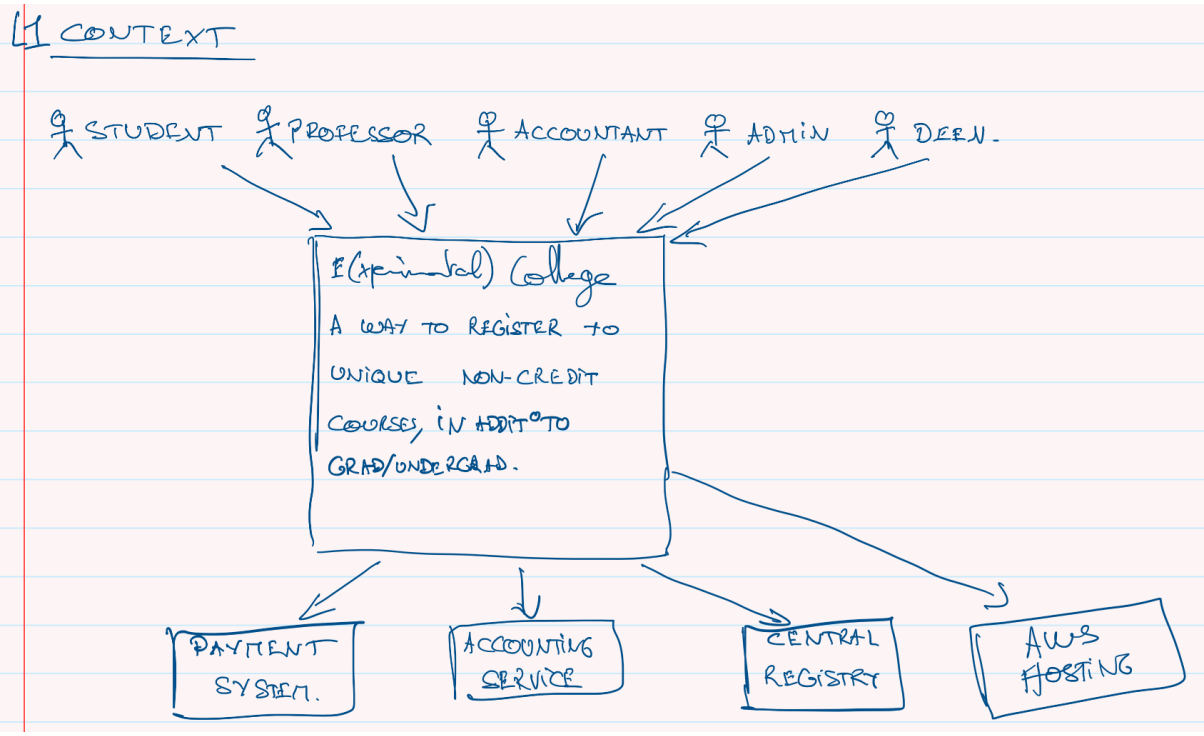
---

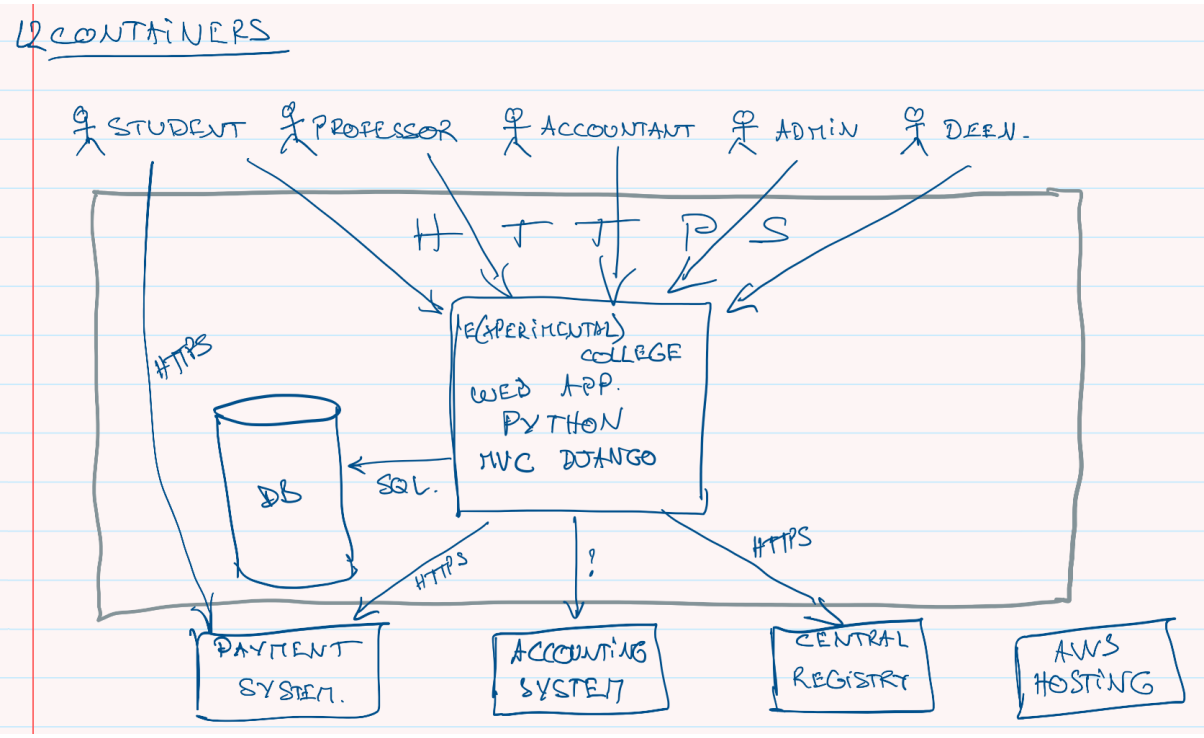
Penelope was assigned to this project from the beginning, but she only recently started to have the time to deal with it. Although she does not have a technical background, she knows the domain and the users. She also has knowledge of the central registry system of the university.

At the start of the project, Penelope received an Archi draft that was one year old and likely outdated. She identified the top three missing features; the first must-have feature was the ability to request invoice receipts by email. The second feature, which should be included, is the option to make payments by check or cash. Finally, the third feature, which would be nice to have, is the ability for multiple students to register for multiple classes in a single purchase.

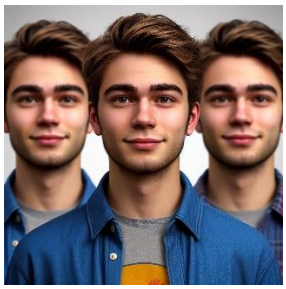
Penelope performed some manual tests and found UX and functional bugs for the Registry and Payment. She had serious doubts about the quality of the code implementing these features. Despite this setback, she remains determined to complete the project on time and to deliver a high-quality product that meets the needs of the users.

*“At the start of the project, Penelope received an Archi draft that was one year old and likely to be outdated”*



## Julian, Junior Developer



Julian is a junior developer. He is a resourceful individual who managed to complete his latest project almost entirely on his own. This is his second project since finishing his studies. He prefers to work alone and did not feel the need to interact with others while working on this project. However, he can be a bit stubborn at times.

---

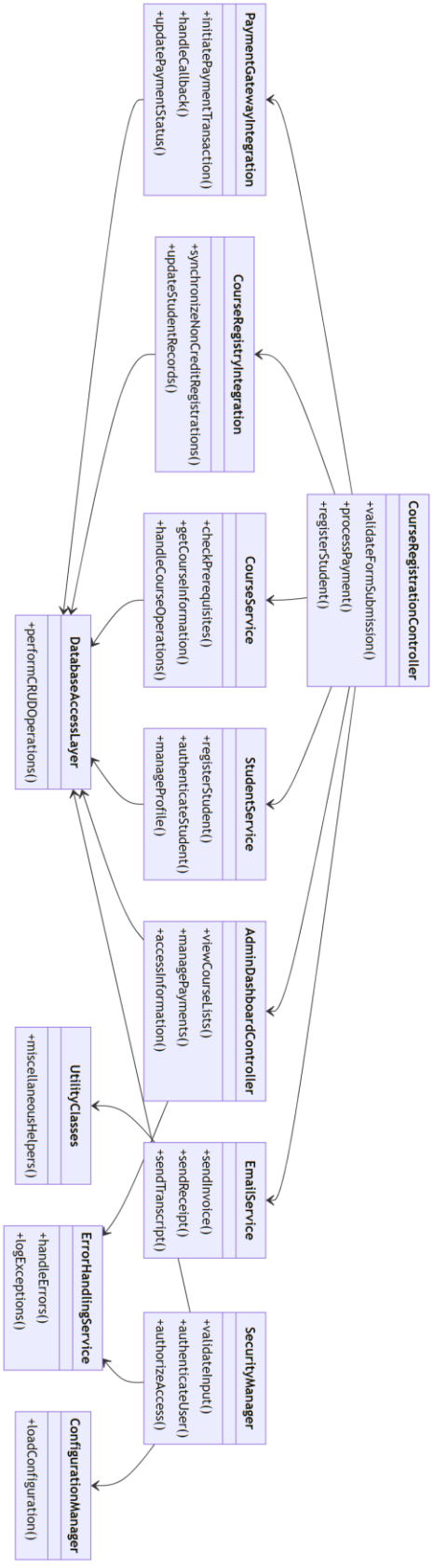
Julian is feeling a bit challenged by his current project and is happy to have some help. Yesterday, just before our workshop, he generated a class diagram , as well as a listing of lines of code and test coverage for each class. Overall, he is happy with the CourseRegistrationController, CourseService, and StudentService classes but not with the DB Access Layer and ErrorHandlingService classes. The rest is so-so. Julian wrote almost all of the code himself, relying on his junior coding skills. He knows about Git but is unaware of good engineering practices like continuous integration, integration testing and security. He also does not know how to test difficult cases like DB code or integration with external services.

In an effort to make the system secure and robust, Julian implemented a SecurityManager and an ErrorHandlingService. He delegated the AdminDashboardController to Ingmar, his intern. Julian is confident that he can code in any module except AdminDashboardController, DatabaseAccessLayer, SecurityManager, and ErrorHandlingService. These modules are somewhat getting out of control!

He also delegated to Ingmar the compilation of a bug listing.

When Julian started the project, he did not write any test. However, as the project progressed, he realized the importance of unit testing and started adding tests wherever he could. Some classes, like the DB or external system integration classes, are not easily testable. As a result, Julian spends a lot of time doing manual testing.

*“Yesterday, just before our workshop, he generated a class diagram…”*



## Ingmar, Intern

Ingmar is a geeky individual with a strong interest in new technologies. He is very curious and always eager to learn new things. He is also an avid video gamer, this reflects his passion for technology and coding.

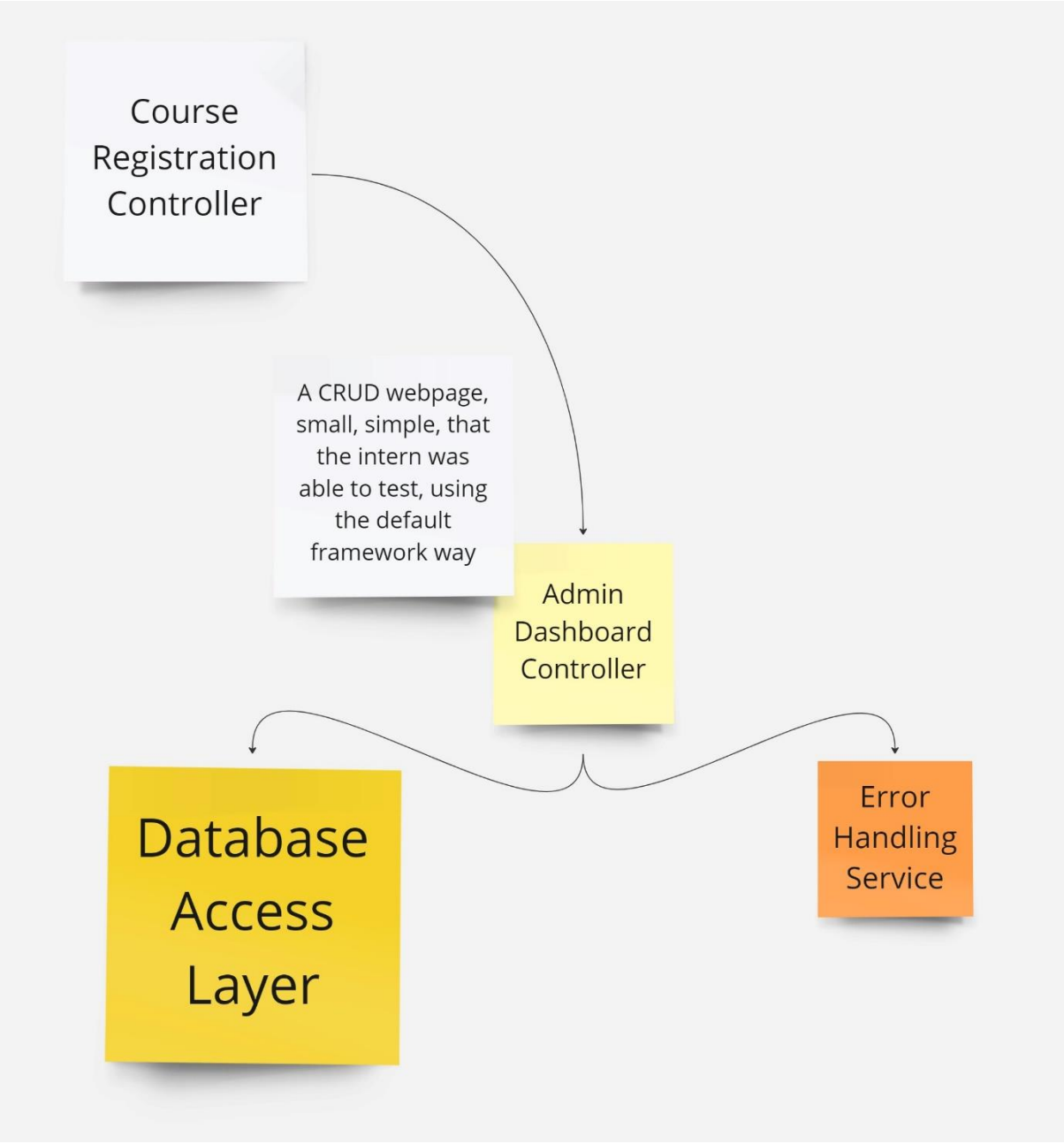
---

Ingmar wrote the AdminDashboardController. He is very knowledgeable about the controller and its dependencies and has even drawn a class diagram of the classes before the workshop. He learnt about CRUD testing at university and was able to test the admin dashboard with the framework. Julian, his supervisor, gives him enough autonomy.

Despite his skills, Ingmar struggles sometimes with the APIs of DB Layer and ErrorHandling. He often feels that he is only able to work in the AdminDashboardController. However, he spent three months coding the admin dashboard, thus demonstrating his dedication.

Just before this workshop, Ingmar was asked to gather a list of modules impacted by bug fixes over the previous months. He did a meticulous job and compiled a table that he brought to the workshop.

*“He has even drawn a class diagram of the classes before the workshop”*



## Sandy, Senior Developer

Sandy is an experienced developer with over 10 years of experience. She has previously worked on the central registry code. She joined the project two weeks ago and is new to the codebase. Sandy is a perfectionist and sees all the problems. She is a compulsive problem solver and is knowledgeable about good coding practices.

---

Before joining the project, Sandy received an old class diagram that was outdated. She conducted a bug analysis from the records and a testing analysis from the code, but there were no integration tests. Sandy is not familiar with this codebase, especially the dependencies between the classes.

From inspecting the code:

- She has a gut feeling that there are security issues in SecurityManager.
- She discovered that admins could see courses and payment, but not student personal information.
- Sandy was able to "hack" the system and access this information.
- The code does not map the domain, everything goes into a unique tangled DB, and basically, everything knows everything.
- There is no DB server backup.
- The CourseRegistrationController and CourseService are the two main classes containing almost all the logic, yet they know all the rest, which makes changes more and more difficult to do, requiring modification of the code at different places.
- Almost nothing has been done regarding deployment.
- There is a potential latency issue when accessing the central registry, which is sometimes slow because all calls are done synchronously.
- The DatabaseAccessLayer is bloated and getting out of control, and changing to a new DB provider would be problematic.
- There is no CI.
- SecurityManager and ErrorHandlingService: these are two transversal concerns, the classes are getting too big and they don't really do the job well.
- There are some unit tests, but they are not run automatically.
- Utility classes are a mess of different, unrelated concepts.

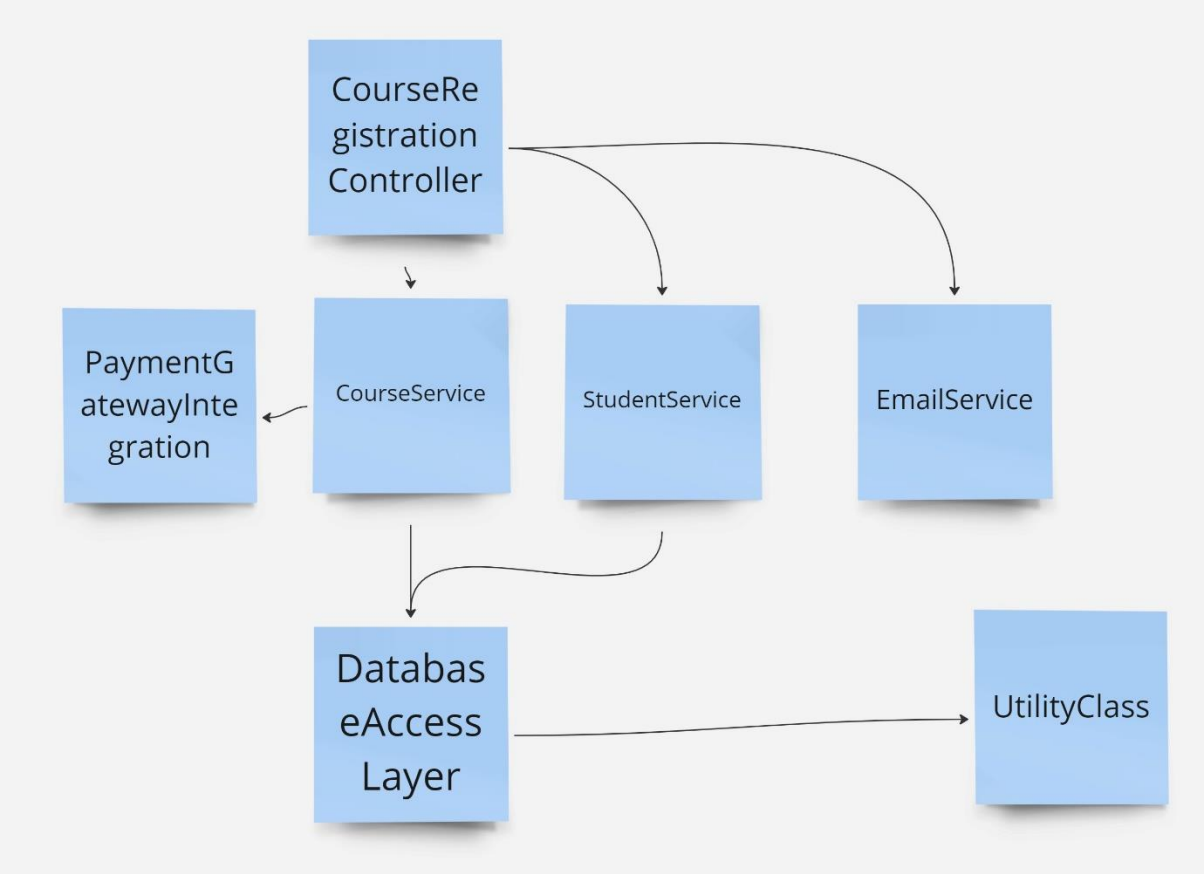
- There are no end-to-end or functional tests.

Sandy checked the CourseRegistrationController, which, despite being a god class, is tested, and the DatabaseAccessLayer, which is not tested. Overall, the code she checked is difficult to work with. Sandy also checked the SecurityManager code and spotted some obvious dangerous flaws in this code.

Obviously, Julian, the junior dev who built the product, did not know about quality metrics and mostly assessed quality based on gut feeling. Sandy conducted a cyclomatic complexity analysis and brought a listing to the workshop.

## Sandy, Senior Developer

*“Before joining the project, Sandy received an old class diagram that was outdated.”*



## Sandy, Senior Developer

*"Sandy conducted a Cyclomatic complexity analysis and brought a listing to the workshop."*

| Class Name | Cyclomatic Complexity | Quality |
|---|---|---|
| CourseRegistrationController | Low (3) | Good |
| PaymentGatewayIntegration | Medium (7) | Fair |
| CourseRegistryIntegration | Medium (6) | Fair |
| CourseService | Medium (5) | Good |
| StudentService | Low (4) | Good |
| EmailService | Medium (6) | Fair |
| AdminDashboardController | Medium (5) | Good |
| DatabaseAccessLayer | Medium (6) | Fair |
| SecurityManager | High (10) | Poor |
| ErrorHandlingService | High (8) | Poor |
| UtilityClasses | High (9) | Poor |
| ConfigurationManager | Low (3) | Good |

# Appendices

## Project Summary

*E(xperimental) College*

Local college now offers unique non-credit courses in addition to the usual grad/undergrad courses, and they need a registration and payment system

Users: students, college users, admin, and accounting

### Requirements:

- existing central student/class registry is NOT integratable--only https web form access allowed
- accept payments
- track registrations
- non-credit registrations must be duplicated in central registry (but not by hand!)
- students enter the central registry if their payment succeeds
- updates/deletes from the central registry is okay as a manual process but preferably automated
- payments can be credit card, bank account withdrawal, check or cash
- course prereqs
- students can request invoice/receipts and/or transcripts (which should be emailed)
- multiple students register for multiple classes with one purchase
- admins must see course lists, payments, but not student personal info

### Additional Context:

- very little budget for IT
- 'How do they register and pay?' was a neglected requirement for many months
- hard deadline for class registration in 6 months

## Project History

- Project ongoing for 1 year
- Mostly built by Julian, a Junior dev
- Ingmar, an intern joined a few months ago
- Tight 6m deadline!
- Penelope, the project manager, will be much more present now
- Sandy, a senior dev, joined 1 week ago