Teaching DevOps at the Graduate Level: A report from Polytech Nice Sophia

Benjamin Benni, Philippe Collet, Guilhem Molines, Sébastien Mosser, Anne-Marie Déry-Pinna

Teaching DevOps at the Graduate Level
A report from Polytech Nice Sophia

Benjamin Benni[1], Philippe Collet[1], Guilhem Molines[1,2], Sébastien Mosser[1], and Anne-Marie Pinna-Déry[1]

[1] Université Côte d'Azur, CNRS, I3S, France
{benni,collet,molines,mosser,pinna}@i3s.unice.fr
[2] IBM France Lab, guilhem.molines@fr.ibm.com

1 Introduction

The massive evolution of IT development towards new Web architectures, from service-oriented to micro-services, clouds and containers, call for changes in the way software is developed, deployed and maintained. DevOps has emerged as a set of practices bridging software development (*Dev*) with software operations (*Ops*) [1]. DevOps makes up a model in which development, quality assurance, releasing, deployment, operation with infrastructure management, and maintenance are integrated and automated as much as possible. With automation and monitoring present at all stages, a DevOps approach is supposed to reduce the time between a change (*e.g.*, a commit) and its availability in production, while mastering quality.

From a teaching perspective, hiring companies for software engineering students are currently in the middle of a technological transformation to introduce DevOps pipelines in their organizations, while agile and continuous integration practices are still in the process of being digested. It is clearly necessary for our students to be aware of such practices to complement their background in software engineering and architecture, and also to make a difference at recruitment time. At first sight, it seems easy to integrate DevOps principles with software development projects and other courses dealing with large software systems or software architectures. Still, different issues arise when materializing the course. As DevOps mainly deals with a technological pipelines, a trade-off must be found between using a complete and relevant stack, and understanding the DevOps principles and its pillars: platform, deployment, testing, and people [2]. Furthermore, using toy examples over the isolated elements of a DevOps pipeline would transform the course in a set of basic tutorials, missing a comprehensive point of view of both the principles and the end-to-end technological hands-on.

In this paper, we report on a course dedicated to "*N-tiers Architectures and DevOps*", which aimed at introducing DevOps while tackling these identified issues. It is taught at the graduate level at Polytech Nice Sophia since 2015. The target audience is 4^{th} year (graduate) students specialized in software engineering and architecture. In the remainder of this paper, we discuss the identified challenges to construct this course, as well as the vision to implement it. We

then give some details on the course content and on the used case studies. We conclude by summarizing results and discussing future development.

2 Challenges & Vision

We believe that Software Architecture and DevOps are two sides of the same coin: one needs DevOps concepts to properly implement and deliver complex architectures, and complex architectures justify such an approach. The course follows a project-based approach to support both parts and we rely on the development dimension of the project to create a continuum between architecture and operations. When materializing the course, we then identified the following challenges:

- Even if the technological stack can be hard to apprehend and deploy, tools are just a means to an end, and the course must focus on the pillars associated to DevOps: platform, deployment, testing, and people [2]. As a consequence, the course must focus on the concepts, and use tools only as an illustration. Moreover, coupling architecture to DevOps is important as both approaches complement each others, and the course must smoothly merge these two dimension to support a fully-fledged curriculum.
- We defend that toy examples are not enough, and delivering such a content using isolated labs cannot lead to the comprehensive point of view we envisioned. It is important to rely on a project-based approach where students will be confronted to real-life choices, at architectural, development and operation levels.

As a consequence, the course must provide theoretical concepts for architecture design, software development and operational deployment around a shared project that will be used as a backbone during lab assignments. To simulate real-life software engineering, the labs must be defined thanks to an open and informal specification expressed in business terms, and it will be up to the students to design the right architecture, implement it in an iterative way and support its deployment thanks to a continuous delivery pipeline.

3 Course Content

In the school of engineering, the presented course is taught to 4^{th} year students (graduate level) that have chosen a specialization in software engineering and architecture. It is thus an optional course in the master curriculum with a capacity of 50 students per year.

3.1 Overall Organization

The presented version is the result of merging two course slots, each one over a half-day along a full semester, so that the course is scheduled on each Friday

for the spring session. It notably enables to easily and dynamically focus a day or half of it to a specific topic, *i.e.*, a software architecture topic or an element of the DevOps pipeline, or to give time for the main project development (cf. Figure 1).

Week	Friday Morning		Friday Afternoon		
	08:00 - 09:00	*09:15 - 12:15*	*13:30 - 14:30*	*14:45 - 17:45*	
6	**Introduction to Software Architecture**		**DevOps Overview**	*Mutation Testing Lab*	***Kick-off***
7	*Poly'Event Architecture definition (unsupervised)*		**Arch. for Testing**	*Mutation Testing Lab*	
8	*Mutation Testing Lab (unsupervised until further notice)*		**Func. & Int. Tests**	*Poly'Event Architecture definition*	
9	**Soft. Components**	*Poly'Event Project*	**Arch Dojo #1**	*Poly'Event Project*	***Poly'Event Project Implementation***
10	**Winter break**				
11	**Interoperability & WS**	*Poly'Event Project*	**Cont. Integration**	*Poly'Event Project*	
12	**Cont. Integration**	*Poly'Event Project*	**Build plan & Pipeline**	*Poly'Event Project*	
13	**Technical interview (Minimal & Viable Product)**		*Poly'Event Project (unsupervised)*		
14	**Persistence**	*Poly'Event Project*	**Deployment**	*Poly'Event Project*	
15	**Arch Dojo #2**	*Poly'Event Project*	**Soft. Containers**	*Poly'Event Project*	
16	**Q&A, Stepback**	*Poly'Event Project*	Scaling	*Poly'Event Project*	
17	**Easter Break**				
18	*Poly'Event Project (unsupervised)*		**Technical Interview (Almost-final Product)**		
19					
20		**Architecture Exam (3 hours)**		**DevOps Exam (3 hours)**	

Fig. 1. 2018 planning of the "N-tiers Architectures and DevOps" course

Prerequisites for the course are the following:

- A strong background in object-oriented programming, with fluency in Java;
- The knowledge of some software engineering principles and tooling, *i.e.*, life-cycle, code versioning (Git), unit testing (JUnit), automated construction (Maven);
- Notions of design and UML, mainly to abstract from the associated project code through component and deployment diagrams.

These prerequisites are all coming for the mandatory courses defined by the graduate program followed by the attendees.

The teaching team has slightly evolved over time, but it has been constantly led by a full-time professor and an industrial partner who holds a part-time position in the school in addition to his daily job as a software architect. The team is completed by two other teachers, making a specialized pair for each axis, software architecture and DevOps. This enables each pair to easily follow student project development according to each axis. Students are organized in teams of four, and the course is known to require a strong investment in software development from them. The same case study is addressed under the two different and complementary axes, and students have to work on the development of a system that implements the specifications associated to the chosen case study as lab assignment.

To support the development of such a system, we implemented a reference system named *The Cookie Factory* (TCF) [3] (see Section 4.1 for details). In the first weeks, the course focuses on the concepts associated to n-tiers architectures and the pre-requisites associated to DevOps, *i.e.*, understanding modularization

and testing. To support this task, students are asked to analyze the implementation of TCF. They quickly identify that it is implemented as a single monolith that needs to be modularized at all levels (business implementation, test, and deployment). This step helps them to get confidence with the project technological stack, as well as to identify why and how a DevOps approach is a good fit for such class of systems.

In the following sections, we describe the content of each course axis and show how the reference case study and developed project help in building a consistent solution to the identified challenges.

3.2 Software Architecture

For the software architecture part, we focus on the definition of an n-tiers architecture using software components (implemented as EJBs using the Java EE framework). A part of the architecture is also developed in .Net, emphasizing the need to support system interoperability using Web Services. An introductory

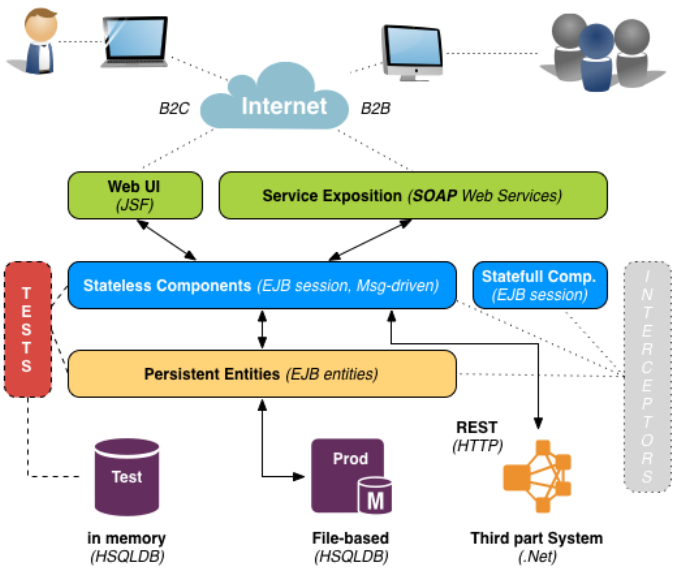

Fig. 2. Technological stack for the architectural axis

course is setting up the work context and the technological stack (cf. Figure 2), while the next courses introduce several principles and some associated technologies using the TCF case study as an illustration:

- Notions of software architecture, layers, and diagrams to represent them;
- The many architectural viewpoints, with focus on functional (work at the interface level), development (modularity and dependency management), and deployment viewpoints;

- *Object-relational mapping* (ORM) variants, and related architectural patterns;
- Introduction to *Enterprise Java Beans* (EJBs), with an overview of the bean types (entity, session, message), their business focus, as well as the principles of inversion of control and dependency injection;
- A focus on session beans, defining a 3-tiers architectures, and introducing stateful and stateless principles and impacts on an architecture;
- Introduction to the notion of services (being different from Web Services technical implementation), contracts and the impact of their different kinds (no contract, light form, strong contract), discussion on bad practices (*e.g.*, REST **is** different from CRUD);
- Focus on domain-driven design, and its implementation through entity beans, issues in modeling relationships, lazy loading, query languages in ORMs, etc.
- Architectural MVC pattern with its implementation in JSF over Java EE, the messaging paradigm and its implementation in JMS, light form of aspect-orientation and its implementation in Java EE interceptors.

As shown on the planning (Figure 1), the conduct of the successive lectures follows the design and development of their own architecture for their project (*Poly'Event* project on the planning). Students should propose an initial architecture with only the introduction, trying to build something consistent with their own background. Then each new lecture enables to criticize their successive propositions, using the TCF case study during the lecture (*e.g.*, with *architecture dojos* where students and professor co-define an architecture respecting several properties during the lecture), and on their own project during the labs. This enables to mix the learning of many technological elements with the different notions of software architecture, their impacts and the necessary trade-offs a software architect should master in her day-to-day work. The TCF case study brings both a starting point for the project and an existing architecture to criticize and evolve as the course progresses.

3.3 DevOps

For the DevOps dimension, the aim is to address the problem of aligning a *development* (dev) team with the *operational* one (ops) to build a given piece of software. Addressed issues are notably how to slice the code into independent modules that can be compiled, tested and deployed in a continuous way, and how to properly test the integration between such loosely coupled components.

This part of the course is organized in a slightly different way. While the students starts to define their own project architecture, the first part of DevOps introduces theoretical concepts and aims at applying them in a separate lab on mutation testing [4]. The organization of this part is as follows:

- Introduction on software delivery, lifecycle and pipelines;
- Reminder on quality assessments, introduction on the different types of tests, how to architect and run them;

– Focus on functional and integration testing, and on what should be considered when running them.

With these lectures, several labs are targeted at building a mutation testing pipeline over a Java project, using Maven, scripts, and a Java source code transformation library[3]. The objective is to make concrete the creation of a pipeline using a software project to build other artifacts, run other tasks (compiling the mutant projects separately), get results (deciding whether a mutant project is passing existing tests or not).

The rest of the lectures focus on introducing the principles and technologies related to the DevOps pillars:

– continuous integration, with build on servers, separated components, their dependencies, notions of artifacts, and necessary repositories, Jenkins[4] and Artifactory[5] being chosen as technological support;
– other subjects related to continuous management, i.e., quality assessment through static code analysis, code branching for a better organization;
– deployment, with the main differences between testing and production environments, as well as test orchestration;
– software containers and virtual machines, focusing on the Docker[6] light container ecosystem, with its composition and scalability mechanisms.

As this part of the course progresses, the students have to apply the principles with the proposed technology to their projects. Considering the *platform* pillar, the tools selected by the students (*e.g.*, continuous integration server, testing framework, containers) must be justified and used accurately *w.r.t* the needs associated to their own project. At the *deployment* level, it is up to the students to mitigate the constraints from the development team, the operational context and the customer's expectations to create the right build plan. For the *testing* pillar, students know about unit tests and the course introduces integration and acceptance tests. Students must justify that the built product is rightly tested at these different levels. Finally, considering the *people* pillar, they have to modularize their code (and the associated tests, build plans,...) in a way that fits their development team and their business objectives [5].

This organization enables the DevOps part of the course to provide real practice of industrial tools applied to a non-toy N-tiers architecture that students are extending at the same time, and also to focus on application of the different principles and pillars of DevOps.

3.4 Evaluation

The evaluation of the course is organized around multiple milestones and deliveries:

[3] Spoon, `http://spoon.gforge.inria.fr/`
[4] Jenkins, `https://jenkins.io/`
[5] Artifactory, `https://jfrog.com/artifactory/`
[6] `https://www.docker.com/`

- after two weeks a first *Minimum Viable Product* (MVP) architecture should be provided by the student teams, for feedback only.
- After two other weeks, an architecture-report must be provided. It contains the following elements: use cases diagrams, business objects definition as class diagram, associated persistent-schema and object-relational mapping definition, interfaces pseudo-code definition (*e.g.*, Java like), components described by a component diagram, deployment of the defined components as a deployment diagram. Each artifact must be justified with respect to its relevance in the proposed architecture.
- At the same time, the mutation testing pipeline should be delivered (through a tagged commit on the provided Git repository). A small report is also delivered, answering the following questions: what are your directory structure and language/script choices? How are mutators compiled and applied to your target project? Which mutations did you write, and why? What issues did you run into, and how did you solve them? What characterizes good mutators?
- At mid-term, demos of the minimal viable product architecture and its associated DevOps tooling are conducted through technical interviews driven by a team of two teachers, one per axis. At that stage, the key-point in architecture is to demonstrate a walking skeleton of the technical stack, from the input entered by the user, sending a request to the Java EE component backend through a Web Service, with an interaction with a third-party service simulated in .Net. For DevOps, the focus is on demonstrating that *Continuous Integration* (CI) is mastered, compiling the project in a way that respects the dependencies among modules, relying on an artifact repository to store the produced binaries. A CI server is expected, so to support the build process and artifacts storage, through inter-dependent build plans.
- Similar demos are also conducted near the end of the course. Technical interviews are conducted by teams, switched from the previous demos. On the architecture side, students should demonstrate a comprehensive architecture, going from the persistence layer to the exposition one (*i.e.*, web services, JSF). They must be able to defend strengths of their architecture, as well as discuss its limitations and evolution capabilities. For DevOps, the pipeline should have evolved from a CI system to a fully instrumented *Continuous Delivery* (CD) pipeline, ensuring software quality through various levels of testing, and generating the product deliverables as composable Docker images.
- Codes and reports should be finally delivered before the exams.

The lab and project evaluations are completed by two final exams, one per axis. Exam subjects mix small targeted questions with a large question on a given case study, evaluating the students capability to step back on the development and DevOps practices. Each axis has also its own marking breakdown:

- Software architecture part: architecture report: 15%; intermediate demonstration: 10%; final presentation: 15%; project (code and report): 20%; final exam: 40%.

– DevOps: mutation testing: 15%; intermediate demonstration: 15%; final presentation: 15%; project (code and report): 15%; final exam: 40%.

4 Case Studies

We describe here the main reference case study, showing its features, architecture and the *kind* of complexity it exhibits to support our teaching approach. We also give a brief description of the projects submitted to students in the past years.

4.1 Reference Case Study: The Cookie Factory (TCF)

The Cookie Factory[7] is an imaginary major bakery brand in the USA, providing a plausible context to the creation of the software system. The *Cookie on Demand* (CoD) system is an innovative service offered by TCF to its valued customers. They can order cookies online thanks to an application, and select when they will pick-up their order in a given shop. The CoD system is supposed to ensure to TCF's happy customers that they will always retrieve their prepaid warm cookies on time.

As shown on Figure 3, the system is defined as layers:

– A remote client, that will run on each customer's device;
– An EJB kernel, implementing the business logic of the CoD system;
– An external partner (simulating a Bank, implemented in .Net);
– An interoperability layer between the kernel and its partners. Communication with the client is supported by an RPC (SOAP) service, and communication with the bank as a REST one.

To deliver the expected features, the CoD system defines the following internal interfaces (Figure 4):

– `CartModifier`: operations to handle a given customer's cart, like adding or removing cookies, retrieving the contents of the cart and validating the cart to process the associated order;
– `CustomerFinder`: a finder interface to retrieve a customer based on her identifier (here simplified to her name);
– `CustomerRegistration`: operations to handle customer's registration (*e.g.*, users profile)
– `CatalogueExploration`: operations to retrieve recipes available for purchase in the CoD;
– `OrderProcessing`: process an order (kitchen order lifecycle management);
– `Payment`: operations related to the payment of a given cart's contents;
– `Tracker`: order tracker to retrieve information about the current status of a given order.

[7] `https://github.com/polytechnice-si/4A_ISA_TheCookieFactory`

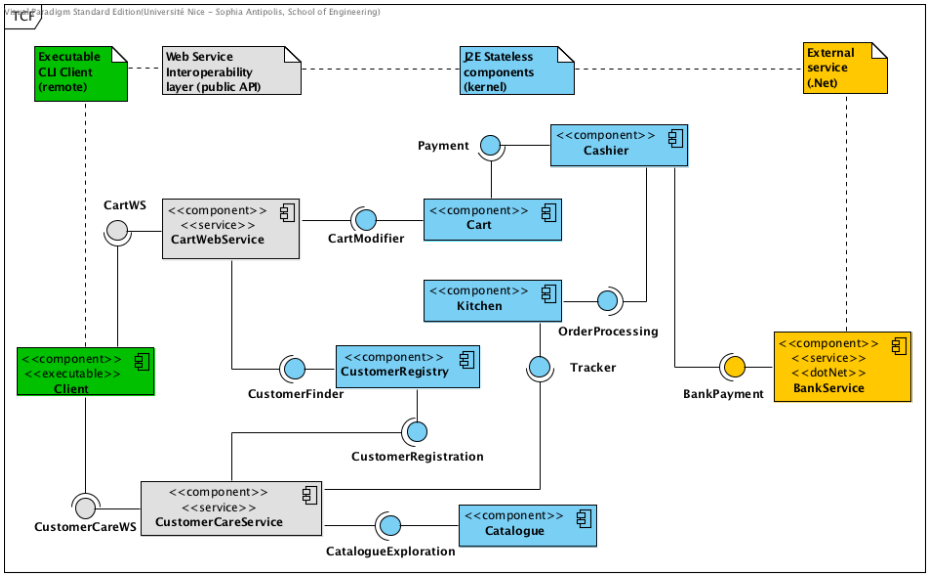

Fig. 3. Component diagram of The Cookie on Demand system

To ease comprehension by the students, the business objects are simple (Figure 5): Cookies are defined as an enumerate, binding a name to a price. An Item models the elements stored inside a cart, *i.e.*, a given cookie and the quantity to order. A customer makes orders thanks to the CoD system, and an order stores the set of items effectively ordered by the associated customer (bidirectional association).

The implementation of TCF is made of 102 Java Classes, representing approximatively 3,000 lines of code. As the focus of the course is an introduction to software architecture, we made the choice to go as lightweight as possible with respect to the tooling. We thus decided not to deploy a real set of application servers and use embedded artifacts instead. This is the very justification of using TomEE+ as Java EE container (instead of a classical Tomcat or Glassfish container) and Mono as .Net implementation (instead of the classical Visual Studio technological stack). We advocate that the execution details are not important when compared to the complexity of designing the right system. In addition, mapping this demonstration to existing application servers is pure engineering, with no added value.

4.2 Case Studies to be developed by students

As previously mentioned, each year, a different product case study is proposed for the development stage, each being presented like the TCF specification part, with a product vision, examples, personas and related epics. We give here a brief description of these projects:

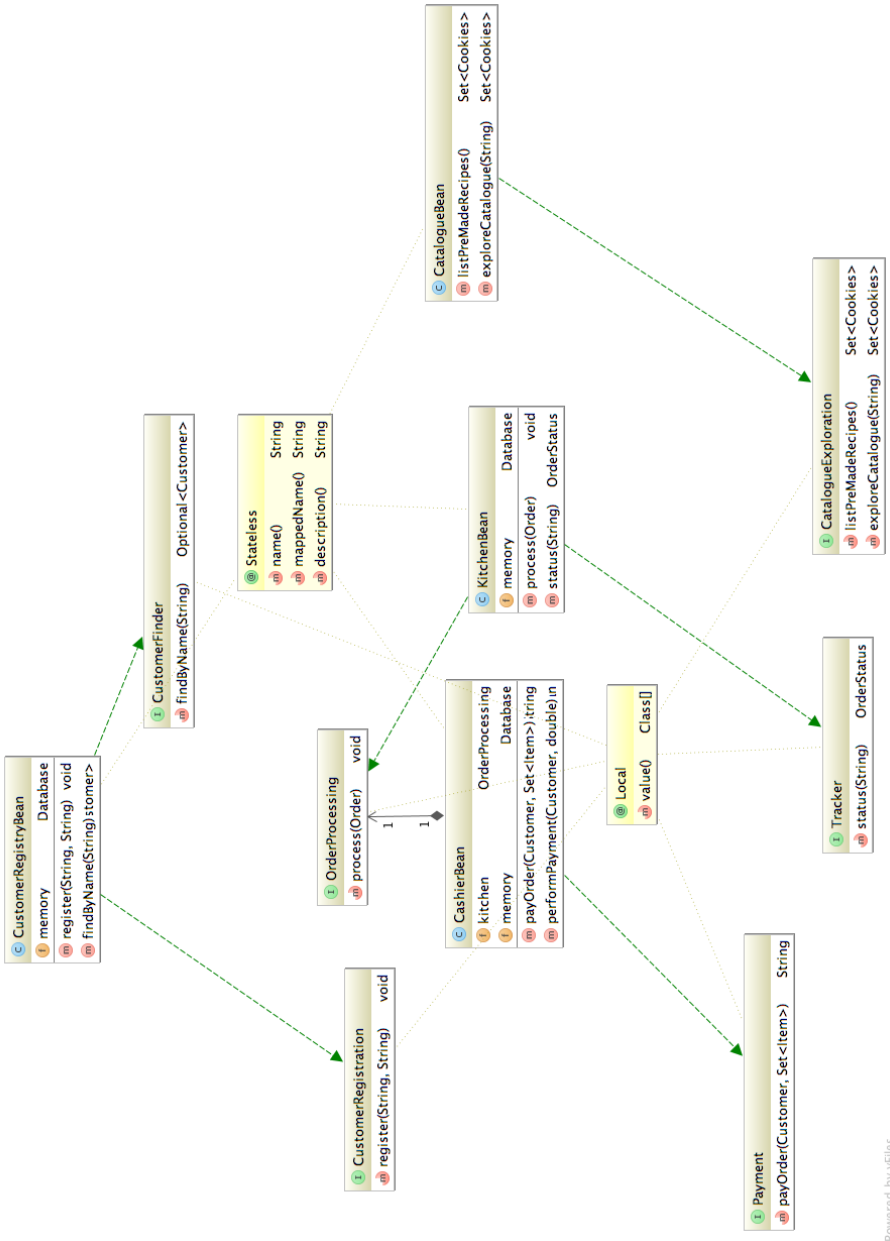

Fig. 4. Interface details of The Cookie on Demand system

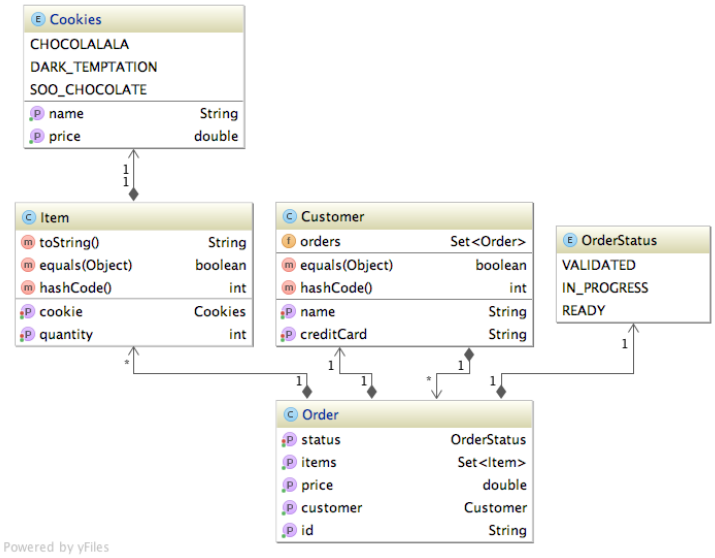

Fig. 5. Business objects of the Cookie on Demand system

- *PolyEvent* is an event management system at the scale of an academic campus, events being internal or external, with booking of premises, possible catering, etc. The system should be generic enough to target different campuses, and should handle both the planning stages and the event day and six personas are defined (logistics manager, premises manager, accounting manager, an external event organizer, cleaning company contact, campus event manager).
- A *Disloyalty card* is a loyalty card targeting a specific commercial zone instead of a retail chain, with a sponsoring from the town council of the zone (*e.g.*, gifts, parking discount with any purchase) to encourage customers to visit as many shops as possible in the area. This kind of card boosts local shops and reward customers who shop in the promoted zone. The developed system should be deployable in medium to large cities, with few changes between a deployment to another. The system card can be used as a payment method for small amounts, and frequent buyers get a VIP status with more advantages. Personas are different kinds of buyers, *e.g.*, a town employee and a shop manager.
- *Isola 3000* is a ski resort management system for a company owning two resorts, the main feature being lift tickets selling and automatic access control to the ski lifts. A ticket is an NFC card and lifts are connected with different means to the main resort (ethernet, wifi, radio waves, nothing at all). Outdoor screens show slope availability and are connected in real-time with connected lifts and patrolmen. Tickets are sold online or at counters, many pre-built offer are proposed with different discounts, premium statuses, and

specific durations or area restrictions. As these offer might evolve, statistics over sales are necessary.

– *PolyTweet* is a social-network based solution to solve communication problems between students, faculty members and administrative staff within the school. The system should foster information sharing, by publishing short messages to channels, the school exposes several public channels available to external users (for integration purpose with the public website), files can be attached to messages (*e.g.*, pictures, lab descriptions). Communication channels can be created as open internal channels, with moderators. To evaluate the return on investment, metrics over the whole system usage should also be computed and displayed.

5 Conclusion

Results. The course is close to its full capacity since 2015 (137 students out of 150 slots on 3 years). It is evaluated by the project delivery (code, report and oral defense), complemented by two exams (case study, 3 hours). We push students to stop being consumers of tools, and instead become DevOps architects able to identify what is necessary and how tools from the state of practice can be assembled to support a given project. The discussions and interviews made with partners' contacts and interns' tutors are strongly positive on that point. Recruiters clearly state that such a knowledge makes a strong difference between candidates at recruitment time (interns or permanent positions). At the student level, the course received a highly positive feedback in evaluations. Student expressed as comments their surprise about the importance of the *people* pillar. We also noticed that even students who do not specialize into software architecture after the course are introducing the DevOps philosophy in their projects.

From an academic research point of view, building this course also led to interesting questions about service containerization from a software engineering point of view that lead to a publication in the domain of software composition [6].

Future Development In the future, we naturally plan to continue to improve the content and organization of the course. Next year, we will change the way the different elements of the pipeline are introduced. The mutation testing pipeline is not perceived by students as useful as we envisioned, while this organization pushes the application of advanced concepts as containers to the end of the timeline. This prevents students from stepping back from their DevOps realization, and transitively from their software architecture as well. Our plan is then to introduce all principles and pillars of DevOps earlier, together with basic realizations for each part, i.e. a basic pipeline with deployment and a simple *dockerization*, so that they can be applied to the project. Then the advanced concepts, and related technological elements, will be introduced and applied. By using this course as a prerequisite for some specialization course, we aim to deliver to students specialized skills (*e.g.*, micro-service development, user

experience) while keeping in mind the close relationship that exists between development and operations, leveraging our experience in teaching agility and user experience.

References

1. Bass, L., Weber, I., Zhu, L.: DevOps: A software architect's perspective. Addison-Wesley Professional (2015)
2. Shaw, J.: The Four Pillars of DevOps: Agility for the Enterprise (Agile Cambridge). `https://www.slideshare.net/johnfcshaw/four-pillars-of-devops-john-shaw-agile-cambridge-2014` (2014) Accessed: 2017-01-10.
3. Mosser, S.: The Cookie Factory (J2E 7 reference implementation), version 2.2. `https://github.com/polytechnice-si/4A\_ISA\_TheCookieFactory` (2017)
4. Woodward, M.R.: Mutation testingits origin and evolution. Information and Software Technology **35**(3) (1993) 163–169
5. Evans, E.: Domain-Driven Design: Tacking Complexity In the Heart of Software. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2003)
6. Benni, B., Mosser, S., Collet, P., Riveill, M.: Supporting Micro-services Deployment in a Safer Way: a Static Analysis and Automated Rewriting Approach. In: Symposium on Applied Computing, Pau, France (April 2018)