Assignment 1 Model Driven Software Engineering (TDT425) NTNU

Author: Ana Barrera Novas

1. Description of the system

The project for this assignment is originated from a Software Process course I took at my home university, as I am an exchange student. The goal was to design a software without getting to the actual implementation, only documents and prototypes.

It is a multiplatform app for the management of a driving school which has grown too large for continuing using office documents. The platform allows students, teachers and administrators to stay in contact and organize their calendar, lectures and materials.

The following sections provide a more detailed explanation of these features, but I will now give an overall description of how the app works:

The system utilizes databases to store user information, chat history, calendars, and lectures.

Once authenticated on the app, depending on the category you belong to (student, teacher or administrator) you have many different functionalities. Any authenticated user has access to actions like logging out, consulting and modifying their profile, and have access to the chat system. This chat works simply with direct messages between users of the system, making everyone able to open conversation with any other user.

As a student, you have access to a tests section where you can check your theoretical driving knowledge, being able to save the progress to the cloud and re-do them multiple times. You can also access a monthly calendar with the classes (both practical and theoretical), see the material and sign up to the theoretical ones and request personal practical lessons in a certain date at a certain time. Students can also make requests for exams when they believe they are ready, and these will pop up as notifications for administrators to accept or decline.

As a teacher, you can also access a monthly calendar but with more options. You can add, remove and modify material to a theoretical lecture on a certain date, as well as seeing the profiles of the students who signed up for the lecture. Also, you can block hours so no practical lessons are assigned to you at that time, cancel lessons and sign the confirmation of assistance of students for personal practical lessons (as they will have to pay for them). The payment system is external. If faults are detected in cars used, teachers can make requests to get the car checked out. These requests will also pop up as notifications for administrators to start the procedure to get the car to the workshop. This is done separately from the system

As an administrator, you have access to all the profiles. You can create more or delete old when new students come into scene. You can also schedule classes (both theoretical and practical) and exams and accept or decline student's requests.

2. *Current features*

Features are characteristics or end-user-visible behavior of a software system, used to specify and communicate commonalities and differences and guide structure, reuse and variation (definition given in class). With this definition I believe that I can find different levels of features. But taking into account that the project stayed in a middle-point (no real implementation), I will describe high-level features.

- **Authentication management**:
  - User login: the system implements a user database with all the information of students, teachers and administrators. It's the first step to access the system as a client to be able to access to the rest of the possible actions.
  - Profile management: from then on, users can view and modify their own information. The system will only allow administrators to modify every user in the database, create new users and delete them.
- **Chat system**:
  - Sending messages to other users regardless of their role.
  - Viewing chat history of the currently authenticated user.
- **Tests system**:
  - Creation and deletion of tests: teachers can create new theoretical tests of 30 questions with the solutions so that the correction is automatic.
  - Realization of tests and storage of results: students can make multiple tries of tests, and results are stored. Personal wrong answered questions are grouped and can be consulted at any moment.
- **Calendar system**:
  - Scheduling lessons: teachers and administrators can create new practical or theoretical classes, cancel already scheduled lessons and block hours of their calendar.
  - Sign-up for lessons: students can register to programmed theoretical classes or request practical classes. The system automatically stores the information of all the teachers' schedules and only shows available hours for the requests. Teachers can also confirm the assistance of students on practical lessons to justify the payments.
  - Material management: when accessing a theoretical scheduled lesson, a view of the details is displayed. Teachers can upload and delete material that signed-up students can view. For practical lessons, the car used can be added to the details and the assistance of the student can be confirmed.
- **Request system**:
  - Exam requests: students can fill in requests for exams when they believe they are ready. Administrators can see these requests and accept or decline them.
  - Car management requests: teachers can fill in requests for checking a specific car for faults. Administrators can see these requests and accept or decline them.

3. *Current variability mechanisms*

As defined in class, variability is the ability to derive different products from a common set of artifacts. As there is no implementation of everything explained, it can be quite hard to find examples of the mechanisms explained. Still, staying in high-level aspects, I can find a couple of abstract variability elements currently in my project.

As a more general description, even though the code wasn't written, everything was thought subconsciously with a language-based and composition-based approach for variability. While developing the design we always had in mind principles and patterns learned in Object-Oriented Programming classes.

- **User roles**: depending on whether the user is a student, teacher, or administrator, different functionalities are enabled or restricted. They have a common base as users. Every common implementation detail would be grouped and generalized and leave as fewer specific components as possible. This always leaves the possibility of adding new roles in the future for a more advanced version of the app for a larger driving school.
- **Lessons**: for the material management system, different lessons might require different types of materials like videos, slides, links...The system has variability in the sense that it is thought to support multiple file types and be flexible in how files are organized or accessed for different needs of the educational institution.
- **Calendar**: depending on the user's role, the scheduling system acts differently. Students can sign up for classes and see the material, while teachers manage time slots and administrators have even more options. The calendar is flexible to add more features that could be toggled depending on the user's role.

4. *Possible variability extensions*

Even though right now the project may not have that many variability elements due to the lack of implementation, I can imagine several ways of extending the project.

I'll start with variability and generalization of the current features:

- **Authentication management**: instead of hardcoding the three user roles and manage everything with conditional statements, a more dynamic permission system could be implemented. Using a configuration file with permissions marked with "enabled" or "disabled" could be useful. This way, new roles and exceptions are way easier to handle and defined at runtime.
  This would allow a lot of variability for the different roles you can find in specific organizations. For example, if we had specialized teachers for theoretical and practical classes, if we wanted to add personal (cleaners, mechanics for the cars, maintenance personal) ... A generalized role-permission system would allow more diverse product configurations.

- **Chat system**: the current chat system allows strictly direct communication between 2 people and limited to text messages. Rather than having these restrictions, the system could support different communication models like group chats, general notifications, forums, role-specific chats or even lesson-specific chats. This could follow a plugin-based architecture, so the system could allow administrators to enable the desired communication methods for the institution. This could make the system more suitable for many different needs.

- **Testing system**: in its current form, the system only allows one kind of theoretical test. It also allows to re-do them and check previous mistakes to review, but this can be generalized. Instead of being limited to tests of 30 driving-related questions, the system could be designed to support any type of test or quiz. Different kinds of categories of tests, different kinds of questions (true/false, short answer, multiple choice...) with specific scoring and grading rules. With this, the testing system could be more complex and could be applied to many other educational-related contexts and institutions.

- **Calendar system**: right now, the system is limited to scheduling driving lessons (both theoretical and practical) and blocking hours, but this could go further. The system could be turned into a general event management system. Instead of being limited to lessons, the system could manage any kind of events related to the institution. It could even allow integration with external calendar systems. Abstracting scheduling rules in a configuration file so administrators across various institutions decide their functionality could be useful. It could allow them to define institution-specific scheduling policies (like lesson intervals, exam periods and holidays).

  Also, in this system the car management was included. This could be further generalized by allowing booking other materials for the lessons as classrooms, vehicles, tools or equipment rentals if the domain was broadened. A **room booking system** and **physical material lending system** could be useful. Teachers could then book classrooms or shared and students could also book rooms for studying or group projects.

  And lastly, the learning material feature was implemented here. Even though right now it has quite a lot of variability by letting it support various types of learning materials (as analyzed in part 3), it could also allow to structure the contents, add content versioning and collaborative editing with different permissions depending on the users.

- **Request system**: currently, the application allows students to submit exam requests and teachers to submit car maintenance requests. This could be generalized by letting it handle more different tasks like lesson changes, material review requests or general administrative requests. Different types of workflows could be configured by administrators based on the type of request, like who can submit, what can be requested, who can approve and how notifications are handled.

5. *Software Product Line and new features*

I believe that creating a Software Product Line based on the app could be achieved efficiently with the right resources. Educational apps are everywhere, and they are useful. The set of features described in the beginning are simple and useful for everyday use in the educational context. It could even develop into company management applications or organizations and institutions management. With everything properly generalized, we have fundamental core assets for any management related topic: time management (originally calendar), exams and formularies (originally tests and requests), security (originally authentication system) and contact between workers and/or clients (originally personal profile information and chat system).

In this case, I would be following a reactive approach. Since I already had the design of one variant (the specific driving school app) without knowing it would lead to more, I would have to refactor many features. All the generalization process I did in the last point would take a lot of time and effort, but less resources were needed at the beginning and at least I had the first results in less time.

For domain scoping, even though I explained before that it could expand to a big domain (any management related area), I would stick to the educational domain for the SPL. Middle-schools, universities, language academies, music conservatories, particular classes… Just educational institutions that need a software, as big or as small as they are.

Following a feature model, I should define the fundamental abstractions of the domain and their relationships. Educational institutions basic elements that I believe a complete software should cover are:

Resource management: payment system, financial management, physical material management for lectures and infrastructure

Personal & Social management: information of all the people involved in the institution, their roles, permissions, relationships and contact information. For students, record of results or any other academic information. Communication between all of them.

Scheduling management: distribution of workload and timetable management.

Examination & Official management: both internal and external exams and storage of results, bureaucratic processes and documentation.

Security & privacy management: internal documents and confidential information

All of this is just partially covered by the features of the current system (part 2 of his assignment). With the generalized features described in the beginning of part 4 even more of these are covered, but not entirely.

Now I'll describe some possible new features that would complete covering the domain or that I believe they could be interesting for some educational institutions. The system could be modularized to allow selective feature inclusion to increase variability for the different institutions' needs.

- First of all, a more **secure authentication system** could be implemented. If all that relevant information is going to be in the system, at least a two-factor authentication could be required in some educational institutions.

- More information on profiles and a **personal document storage system** could be added and allow users to modify the visibility of these documents. This could be useful in education centers like universities. Users could then have their own personal space for important documents, so students could even create their own academic portfolio with grades, assignments and certificates.

- A **payment system** could be implemented instead of using an external one. This way, generated documents and bank accounts don't need to go through an external system. All the information will be in the same place, accessible only by the correct people. This could be useful for private academies or systems like the original (driving schools, need to pay per practical lesson).

- An **attendance tracking system** could be also added. In the current system it was limited to practical lessons for later ease the payments for them, but now the domain has broadened. This feature would allow teachers to mark attendance for classes and students would be able to view their attendance records. The system could also notify administrators or parents when the attendance falls below a set threshold. This could be connected to the sign-up feature and the notification feature explained in part 2.

- Another feature that could be implemented is an **assignment and homework management**. It could allow teachers to hand out homework or projects with deadlines and students could submit files and text. This could reuse aspects of the learning material uploading system for lessons already in the original project.

- Adding a **parent portal** could also be useful for educational institutions for younger students. It could allow parents to view their children's academic progress, attendance, teachers' comments and receive notifications for important dates, exams or events. This would then be linked to the chat system (notifications), calendar and attendance.

- A **course management system** could also be applicable to all educational institutions, whether it's universities with full degree programs or smaller academies offering small courses and classes. This feature would allow administrators and teachers to manage course creation, assigning teachers and managing its course materials. This has a lot of variability due to the wide range of types or courses or events in the whole educational domain.

With all of this, the summary of the feature model for the educational application SPL could be as follows:

Core features: scheduling system, authentication, lesson material support, basic communication and notifications, attendance tracking.

Optional features: payment system, assignment and homework management, booking system, lending system, parent portal, tests system, personal storage system, request system, course management system.

With this, variants of the SPL would look different. A more basic and smaller academy could use a software with only the core features and one or two optional, while a large university could use also most of the optional features. All these features could be toggled by admins for their own institution needs.

In conclusion, I believe that the project can be generalized in many ways. The thorough analysis from an abstract and generalized point of view has led me to detecting some details that would be inconvenient in the actual implementation, even more if more variability was wanted. I think that the current version of the software design lacks some important features and doesn’t have that much variability, but with the generalization and new features in the chosen domain it could lead to a well-designed SPL.