Université du Luxembourg

Faculté des Sciences, de la Technologie et de la Communication
Bachelor en informatique (professionnel)

Année académique 2015 – 2016

Mémoire de fin d'études

A Game-Based Student Response Framework for Interactive Education – Back-End Services and Administration

Auteur
Dren Gashi

Matricule
0130732528

Date
06/06/2016

Etablissement d'accueil
Université du Luxembourg

Responsable local
Dr. Jean Botev

Responsable académique
Prof. Dr. Steffen Rothkugel

Abstract

Lectures may not only be boring but also ineffective. A study found that students enrolled in a traditional lecture are 1.5 times more likely to fail than those enrolled in an active learning environment. The current generation of students has grown up in a technological environment. Students often require constant engagement to preserve the information covered during a lecture. To solve this kind of problems, there exist game-based learning tools that have great potential to motivate and engage students. However, current tools have different limitations and shortcomings in various areas. We propose Yactul, a new web-based learning platform that combines the best features of existing systems, while adding new possibilities. Based on a modular architecture, our tool allows adding new types of questions on the fly without writing a single line of code. Having implemented a specific type of question or activity, it can be exported as a web-archive and imported as a module in the application server where Yactul is running. Next, the module can be paired via well-formed REST-interfaces for communication. Modules may not necessarily be imported due to a language-agnostic REST API and hence, can be written in any programming language and run on any server in any location. We compare our tool against exisiting popular web-based learning platforms. The results show that our tool overcomes the limitations of the competitors while combining the best features and add new possibilities. The backend of Yactul is deployed as a GlassFish JavaEE application server connected to a MySQL database server. The front end is purely written in HTML5 and JavaScript using WebSockets as communication protocol.

Keywords: game-based learning, student response framework, back-end services, gamification, Yactul

Les cours magistraux peuvent non seulement être ennuyeux, mais également inefficaces. Une étude a constaté que les étudiants inscrits dans un cours traditionnel sont 1,5 fois plus probable à échouer que ceux inscrits dans un environnement d'étude actif. La génération actuelle des étudiants a grandi dans un environnement technologique. Les étudiants nécessitent souvent de l'engagement constant, afin de préserver l'information couverte pendant une audience. Pour résoudre ce type de problèmes, il existe des outils d'études basés sur jeux qui ont le grand potentiel de motiver et engager les étudiants. Cependant, les instruments courants ont différentes limitations et défauts dans divers secteurs. Nous proposons Yactul, une nouvelle plate-forme d'étude basée sur le web qui combine les meilleures caractéristiques des systèmes actuels, tout en ajoutant de nouvelles possibilités. Basé sur une architecture modulaire, notre outil a la particularité d'ajouter de nouveaux types de questions en marche sans écrire une seule ligne de code. Après avoir déployé un type spécifique de question ou d'activité, il peut être exporté comme un archive web et être importé comme un module dans le serveur d'applications où Yactul fonctionne. Consécutivement, le module peut être associé par l'intermédiaire des REST-interface bien formés pour établir une communication. Les modules ne sont pas nécessairement importés, grâce à REST, une langue-agnostique API et par conséquent, peut être écrit dans n'importe quelle langue de programmation, démarrant sur n'importe quel serveur dans un emplacement quelconque. Nous comparons notre outil avec des plates-formes populaires existantes, basées sur le web. Les résultats prouvent que notre outil surmonte les limitations des concurrents tout en combinant les meilleures caractéristiques et en ajoutant de nouvelles possibilités. Le back-end de Yactul est déployé comme serveur d'applications de GlassFish JavaEE relié à un serveur de base de données de MySQL. Le frontal de l'application est purement écrit dans HTML5 et Javascript, en utilisant WebSockets comme protocole de transmission.

Mots-clés: student response framework, outil d'études basé sur jeux, services back-end, ludification, Yactul

Declaration of Honor

I hereby declare on my honor that I am the sole author of the present thesis. I have conducted all work connected with the thesis on my own.

I only used those resources that are referenced in the work. All formulations and concepts adopted literally or in their essential content from printed, unprinted or Internet sources have been cited according to the rules for academic work and identified by means of footnotes or other precise indications of source.

This thesis has not been presented to any other examination authority. The work is submitted in printed and electronic form.

Luxembourg, June 2016

Dren Gashi

Acknowledgments

I would like to express my deep gratitude to my academic supervisor Prof. Steffen Rothkugel for his excellent support and guidance during the period of twelve weeks of my work. I am grateful for the constructive criticism which always causes me to rethink and helped me to improve. I especially thank him for his proposal of the topic for my Bachelor thesis, for his great efforts for providing me an office in his work building where I could work and consult him for advices, for organizing recurring meetings to discuss my progress and last but not least for the motivating feedback on the results during live presentations of the tool.

I would like to sincerely thank my local supervisor Dr. Jean Botev who has supported me thoughout my thesis with knowledge and many suggestions to improve my work. I especially thank him for his encouragement, for taking the time to discuss my progress, for helping me to establish the concept for the modular architecture of Yactul and for his detailed comments on my thesis.

Furthermore I would like to kindly thank Christian Grévisse. Despite of not being one of my supervisors, he has always taken the time to help me to investigate my code and to solve problems. Many thanks for providing an admirable template for the thesis and helping me to establish the concept for the modular architecture of Yactul.

Finally I would like to thank the following people for participating at each meeting and providing constructive criticism and suggestions: Prof. Steffen Rothkugel, Dr. Jean Botev, Dr. Johannes Klein, Christian Grévisse, Christian Müller, Mike Pereira Gonçalves, Davide Belpassi, Joe Mayer and Maximilian Biegel.

Contents

List of Figures

1 | Introduction

Interactivity is not only important in classrooms but also for learning in general. Students are interacting all the time but not with the important subjects. During class, they interact on mobile devices, tablets or laptops. Teachers must first capture the attention of the class to make a teacher-student interaction even possible. Students often require such interactions in classrooms in order to stay focused. A study by Scott Freeman *et al.* [1] found that students enrolled in a traditional lecture are 1.5 times more likely to fail than those enrolled in an active learning environment. This is due to our present generation which has grown up in a very technological environment. An active learning environment is therefore crucial to build the basis for interactivity. Current *Student Response Systems* (SRS) or also called *Classroom Response Systems* (CRS) try to apply *Gamification* in classrooms to enhance motivation. Gamification is the application of game-design elements and game principles in non-game contexts [2].

Game-design elements, also known as *game mechanics*, are based on human primary desires.

	Human Desires					
Game mechanics	*Reward*	*Status*	*Achievement*	*Self Expression*	*Competition*	*Altruism*
Points	●					
Levels		●				
Challenges			●			
Virtual Goods				●		
Leaderboards					●	
Gifting & Charity						●

Figure 1.1: Game mechanics - Human desires [3]

As we can see in figure 1.1 there are several aspects a human desires. Each desire has a corresponding game mechanic. The black dots signify the primary desire a certain game mechanic fulfills. The more game mechanics included, the more gamification is applied.

The background for the choice for this theme is that gamification has the potential to change the way of learning in a very positive manner. For instance, an empirical study by Hakulinen *et al.* [4] where 281 students participated in the experiment, shows that using game mechanics has a significant impact on some aspects of students' behavior. Students with more badges spent more time per exercise, which suggests that they thought more about the problem before submitting. Badges are a validated indicator of accomplishment, skill, quality or interest that can be earned in various environments [5]. Trying to combine game-like elements in learning can be the key of motivating people and that's exactly what we students want.

However, the current offer of learning platforms providing gamification is quite small. The main reason for this is that the term gamification itself is new. Current existing tools have different limitations and shortcomings in various areas.

The research questions were:

RQ1 : How to build an *extensible* learning platform?

RQ2 : How to apply game mechanics in a learning platform?

RQ3 : How to overcome the limitations of current solutions?

RQ4 : How to combine the best features of existing systems while add new possibilities?

In order to find answers to the questions, the project was split in three sub-projects. Our solution consists of a web-based learning platform called Yactul, which involves three parts: an iOS offline version, a client web front-end and a server-side part.

This thesis concentrates on the server-side part of our tool. The most important goal for this part was to build an extensible architecture which allows adding new types of questions or *activities* on the fly and without changing or adding a single line of code.

Example of activities which we came up are the following:

1) Simple Question: A question with at least two answers where only one can be correct.

2) Multiple Choice Question: Same as Simple Question with the difference that zero or more answers can be correct.

3) Simple Focus: A question with at least two answers where only one can be correct and where the answers are sequently printed on the screen with a flexible time-interval.

4) Multiple Choice Focus: Same as Simple Focus with the difference that zero or more answers can be correct.

5) Point And Click: A question consisting of a text and a picture where a user must click on specific regions on the picture to validate the question.

6) Ordering: A question consisting of a text and multiple answers where the answers need to be put in a specific order to be correct.

7) Build Pairs: A question with multiple text where two strings belonging together needs to be linked together to successfully validate the question.

8) Discussion: A question with a text containing the description of the exercise where users can write answers.

These are only some examples we invented for our tool. As you can see there is quite some *heterogeneity* among the activities. In order to handle all possible activities, we made Yactul *generic*. The extensible architecture of our tool allows linking of activities in order to be used in quizzes. A quiz can be composed of several different types of activities. Activities are deployed as separate and independent applications which can run either on the same server as Yactul or in another. Due to using REST as language-agnostic API for the communication, it is possible for instance to run Yactul on a server in Luxembourg and the different activities to deploy each on a server in a different location across the globe. We defined a well-formed interface between our tool and the activitiy applications to build a basis for the communication layer.

We evaluated current popular web-based learning platforms to determine the weaknesses or limitations. This evaluation was considered for building our tool. The results of a detailed comparison highlight that our tool overcomes the limitations and combines the best features while adding new possiblities.

The structure of this thesis is as follows. In section 2 the state of the art of current popular CRS are presented. It follows a comparison in terms of restrictions and gamification aspects between the tools. We end the chapter with a conclusion. In section 3 we elaborate our proposed tool Yactul, it's extensible architecture, the message flow and a final comparison where we include our tool. Finally we conclude in chapter 4 and present some details about future work.

2 | State of the Art

The current variety of learning tools providing gamification is quite small since the word gamification itself was not coined until around 2002 by Nick Pelling [6]. Only in the last years it has startet to get public attention.

In this chapter, some of current existing game-based learning platforms are presented. Furthermore the tools will be compared in terms of restrictions and gamification aspects.

2.1 Existing Game-Based Tools

All analyzed tools are web-based solutions. This is due to the most significant advantage over native solutions: one can access the application through any web browser, which makes the tool cross platform compatible. It is not astonishing why most of the tools take the web-approach because a heterogenous variety of devices exist in classrooms. It would be otherwise difficult to create a tool that runs on any device. Among several existing solutions, three major web-based learning platform are presented in the following.

2.1.1 Kahoot

The Kahoot platform is owned and funded from the Norwegian Research Council in 2013. At the moment the tool is fully cost-free. However, in the official Kahoot website they state that in the future Kahoot will offer some added value services. Kahoot is the most popular platform out of the three analyzed tools with more than 8 million [7] public quizzes.

Having created and started a quiz, the creator of the quiz gets a generated six-digit long numerical pin and can forward it to the participants. The participants can next use the pin and use a nickname in order to join the quiz.

During a quiz, the question together with the remaining time and the number of already submitted answers is printed on the teachers' screen/projector and the participants can see the question for five seconds. Next, the possible answers are shown in the screen of the teacher in different colors as illustrated in figure 2.1. The students do not see the answers in textual form but rather in rectangles with different background colors corresponding the one's from the projector view as seen in figure 2.2.

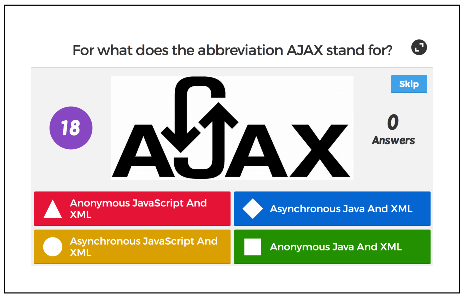

For what does the abbreviation AJAX stand for?
18
AJAX
Skip
0 Answers
Anonymous JavaScript And XML
Asynchronous Java And XML
Asynchronous JavaScript And XML
Anonymous Java And XML

Figure 2.1: Kahoot – Teacher View

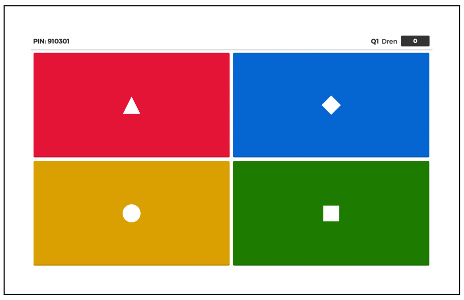


Figure 2.2: Kahoot – Student View

2.1.2 Quizizz

Quizizz was founded in 2015 and hence is in it's initial ages. Less than 10 [8] employees are in charge for the web-based learning platform. Similar to Kahoot, people can create quizzes with multiple questions. In order to access a quiz, students need to have the six-digit pin which is being generated at the moment when the teacher starts the quiz. Unlike Kahoot, the students can see the questions in their screen and don't have to look to the teachers' screen. The UI-View of the teacher is waiting for the responses coming from the student's devices as seen in figure 2.3. In figure 2.4, the view of the student's screen is captured.

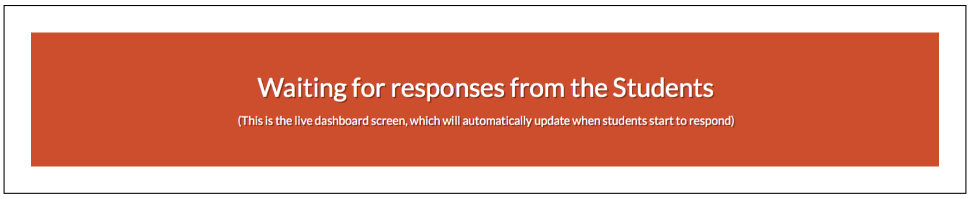

Waiting for responses from the Students
(This is the live dashboard screen, which will automatically update when students start to respond)

Figure 2.3: Quizizz – Teacher View

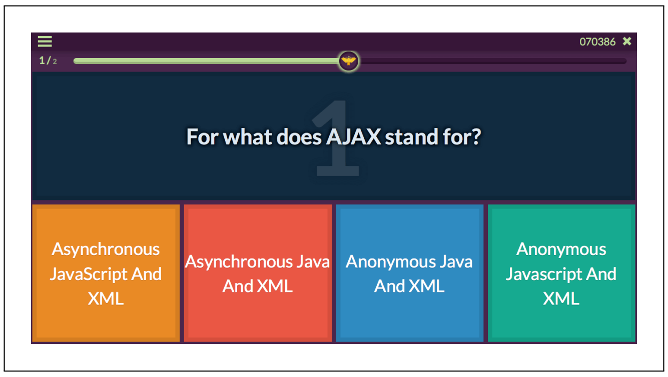
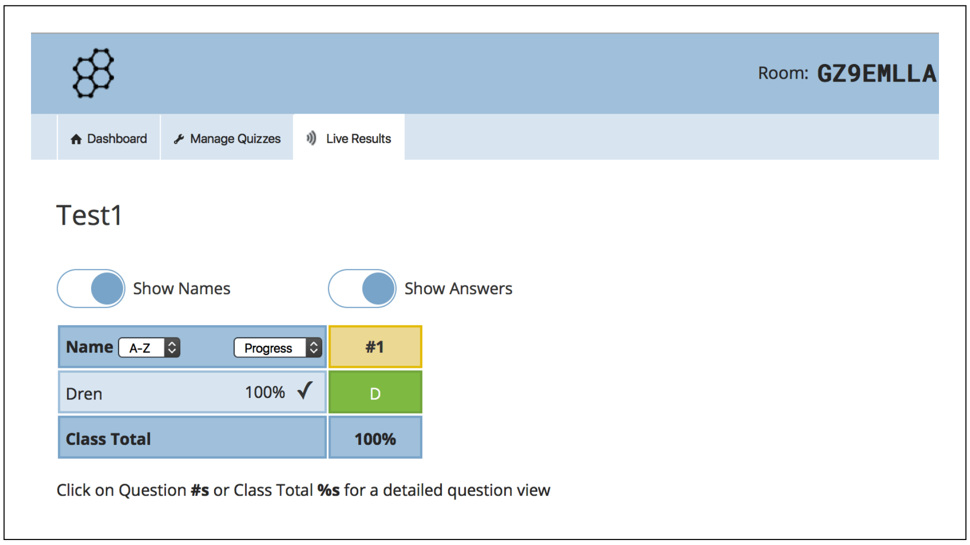

070386
1/2
For what does AJAX stand for?
Asynchronous JavaScript And XML
Asynchronous Java And XML
Anonymous Java And XML
Anonymous Javascript And XML

Figure 2.4: Quizizz – Student View

2.1.3 Socrative

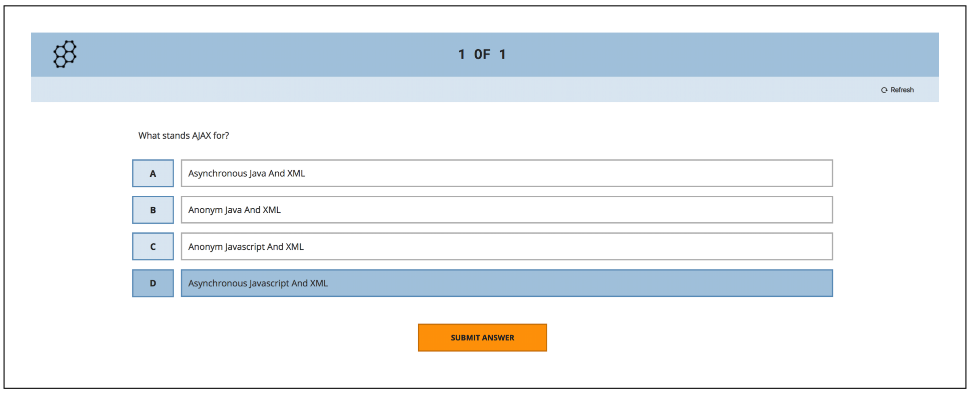
The establishment of Socrative took place in 2011 [9] and therefore it is the oldest out of the three web-based learning tools. The procedure of joining a quiz is exact the same as shown previously for Kahoot and Quizizz with the small difference that the pin does not only consist of a six-digit long numerical number but it consists of a eight-digit long string containing numbers and uppercase letters.



Name A-Z Progress	#1
Dren 100% ✓	D
Class Total	100%

Click on Question #s or Class Total %s for a detailed question view

Figure 2.5: Socrative – Teacher View



What stands AJAX for?

A Asynchronous Java And XML
B Anonym Java And XML
C Anonym Javascript And XML
D Asynchronous Javascript And XML

Figure 2.6: Socrative – Student View

2.2 Comparison

The figure 2.7 lists the differences in terms of features and gamification aspects between the three tools. To keep it simple, we only listed the best features we thought about among the tools and added new functionalities such as featues during a quiz which can be very useful in classrooms. More details about particulars will be discussed in the following.

	Classroom Response Systems		
Criteria	Socrative	Kahoot	Quizizz
Entering room	*pin*	*pin*	*pin*
Multiple Choice	✓	✗	✗
Number of answers	*2..**	*2..4*	*2..4*
Time limit per question	✗	✓	✓
User defined time limit	✗	*fixed range*	*fixed range*
Pause between questions	✗	✓	✗
Offline version	*Mobile App*	✗	✗
Multiple question types	✗	✗	✗
	During quiz		
Modify time	✗	✗	✗
Pause/Resume question	✗	✗	✗
Change order of questions	✗	✗	✗
Modify question	✗	✗	✗
Disable question	✗	✗	✗
	Gamification aspects		
Points	✗	✓	✓
Levels	✗	✗	✗
Challenges	✗	✓	✗
Virtual Goods	✗	✗	✓
Leaderboards	✗	✓	✓
Gifting & Charity	✗	✗	✗

Figure 2.7: Comparison Table

2.2.1 Restrictions

In the table above we can see several restrictions or missing features marked as red crosses. It is surprising that except Socrative, none of the tools support multiple choice questions. The number of answers is also restricted to four in two of the tools. All tools only support one activity, namely a textual question with at least two answers where only one can be correct. This corresponds to the 'Simple Question' activity we listed in section 1.

The table also shows that the most restrictions are given in the criterias listed under the section 'during a quiz'. None of the tools support such functionalities. We believe that teachers need to have control over a running quiz. Imagine a teacher realizes that the time for a particular question he has chosen is not enough. Then, it would be appropriate if he has the possibility to add more time so that students can take advantage of it. Another example is when a teacher realizes that a particular question causes confusion among the students. Pausing and resuming the quiz/question would be very suitable. Thereby the teacher can give additional explanations and then resume the question and start in that state where it stopped. There can be of course many more examples we can image. These are only a few we thought about as most important.

2.2.2 Gamification Aspects

Both Kahoot and Quizizz support three out of six different game mechanics. Kahoot supports a 'team mode' which falls in the category *Challenges* and Quizizz supports custom avatars which we can categorize into *Virtual Goods*. We can also consider the time as a challenge since it produces a slight pressure on the students. However, we think that a time per question is crucial and is not directly a game element. Therefore it is not counted as a gamification-aspect.

It is difficult to implement all six game mechanics while mainting the quiz sensical. A potential hurdle may be a conflict between some game mechanics. For instance, one can implement the *Gifting & Charity* mechanic as a feature such that students can send points to each other in order to stay on a par. However, the human desire *Reward* resulting from the points would suffer since points could be easily obtained when some students work together. Therefore, the implementation of game mechanics must be well choosen.

Socrative does not support any game mechanics. It is designed to be used as a normal quiz system without any gamification aspect.

2.2.3 Conclusion

In this chapter we compared some classroom response systems. We showed some similarities between the tools such as the idea of having a pin to access a particular quiz. Next we compared the tools in terms of gamification mechanics and restrictions. We saw that Socrative does not support any gamification aspects. In the other hand both Kahoot and Quizizz implement three out of six different game mechanics. However, Socrative is the only tool which supports multiple choice questions and more than four answers for a question. Socrative is hence a powerful tool which should be used when gamification is not desired. Nonetheless, in our context, game-based learning plays an important role.

Whether having the question shown on the students screen or not is a more pedagogical question. Kahoot does not show the on the student's devices. In the official website [10] of Kahoot, they state that they are not showing the question and answers on the student screens because they believe that it has a negative impact on social learning. The other two tools, however, are showing both question and answer on the students' devices.

The main drawback is that all compared tools only support 'Simple Question' activities. After all, the tools exist since several years, which might suggest that the underlying architecture does not allow other types of questions and this is the main part which our proposed tool Yactul focuses on. While trying to combine the best features listed above, Yactul also protopically rectifies the restrictions mentioned earlier.

We continue with the main chapter of this thesis which is Yactul and it's extensible architecture.

3 | Yactul

The overall goal of our approach was to build an extensible architecture which allows adding new activities on the fly without writing a single line of code.

Contents

In the next subsection, we describe each of the components used in the modular architecture and the connection between different layers in more detail. In section 3.2 we elaborate on the message flow between students and teachers and in section 3.4 we compare our tool against the presented one's in the State-of-the-Art chapter.

3.1 Modular Architecture

In order to comply with the main research question on how to build an *extensible* architecture to support different activities, we strived for a *modular* approach. Alternatives such as monolithic applications consisting of tightly-coupled code have been analyzed as well. However, due to many advantages of modular programming such as code reusability, program readability, manageable tasks, easier collaboration with othe programmers, etc. we have decided to build a modular architecture. Yactul utilizes a language-agnostic API and is designed as a *microservice* performing small tasks to facilitate a modular approach. We use a *Representational State Transfer* (REST) architecture style as communication interface internally between some components and externally between our tool and an a *module*. A module represents an activity which is a separate and independant application that can be linked with Yactul. The communication is mainly based on *delegation* which we elaborate on section 3.1.3. The process of linking is described in more detail in section 3.1.1. In figure 3.1, a global view of the architecture of our tool is illustrated.

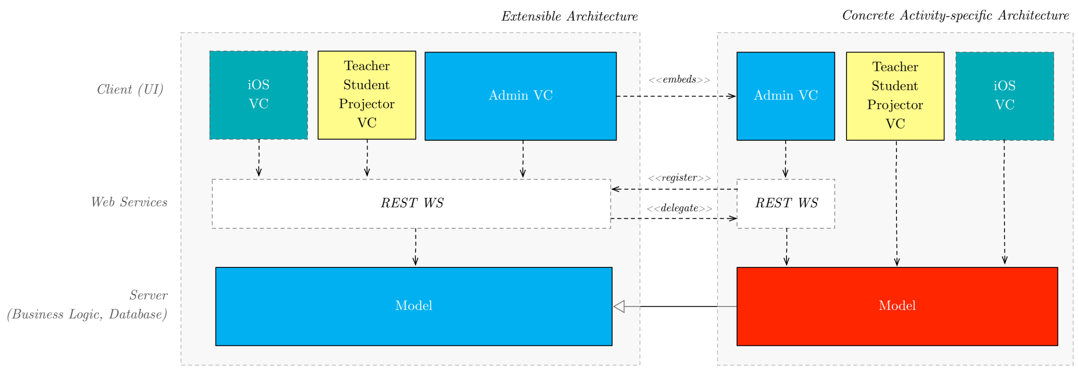


Figure 3.1: Modular Architecture

Our modular architecture is shown on the left-hand side and a general architecture of a module is illustrated on the right-hand side. Each activity must respect this architecture in order to be paired with our application.

3.1.1 Module-Pairing

The linking process between a module and our application is done by *registration*. We store three crucial information about the module in our database. First of all, we store the base URL of the REST of a module for communication purposes. Secondly we also store the name of the activity for identification purposes and in order to fetch it's base REST-URL when given an activity. Lastly the URL of the administration page where the teachers can perform CRUD operations on activities is also stored in the database. The registration process is done by an administrator of our tool. Teachers cannot register new activities but rather use instances of new activities as soon as they have been registred. In the following, an example of a registration entry in our database is listed.

`activityname`	SimpleQuestion
`url`	http://localhost:8080/SimpleQuestion/rest/crud/
`creationurl`	http://localhost:8080/SimpleQuestion/

The `url` value is used for delegating requests to the specific REST web service of a module. The `creationurl` is used for *embedding* the *Administration*

View Controller (Admin VC) of a specific module into the Admin VC in our application.

We can also import the modules in the server where Yactul is running. This can be done either in the admin console of the server or by copying the module application in an auto-deployment folder.

3.1.2 Server-Side Implementation

In this chapter we will focus on the database of the backend of Yactul. The most part of the business logic is spread over the web services and websockets which are described in section 3.1.3 and 3.2 respectively in more detail. We use the world's second most used [11] and the most popular open source relational database *MySQL* as Database Management System (DBMS).

As described in chapter 1, we have invented some activities with their own attributes. However, we noticed that some of the activities have common characteristics but others are very heterogeneous. After some analysis about what are the crucial attributes each activity should or must have and some conceptualization, our solution consists of a model as illustrated in the following UML Class Diagram.

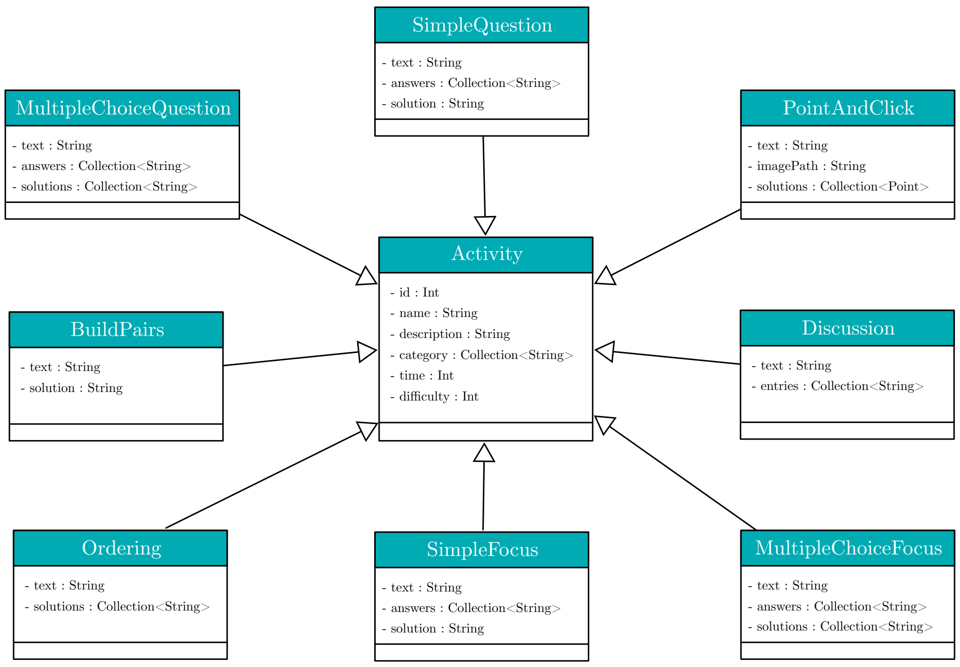


Figure 3.2: Activities – UML Class Diagram

Each new Activity Java class must extend from the parent class Activity. We declare all Activity Java classes as *Java Entities*. Entities can be persistantly stored and represent a model for the tables in the database. Using the Database Schema Creation feature[1] of the persistence provider, we can enable an automated creation of the database tables based on the entities on the deployment of an application. Hence we do not need to write any SQL-commands for creating tables and relations whenever a new activity is created. Note that the activity subclasses are not stored in the server-side of our tool but rather in separate modules. The parent class itself only provides a template to be used in each module. When using Inheritance in combination with entities, an *Inheritance Mapping Strategy*[2] must be defined in the parent class. We use the *Joined Entity Inheritance Mapping Strategy*[3] because it is the most efficient in terms of storage. Using this strategy, the parent class is represented by a single table and each subclass has a separate table that contains only those fields specific to that subclass and a *primary key* which is used as *foreign key* to make a relation to the parent table. The `SINGLE_TABLE` strategy stores all classes into a single table with as many columns or fields as distinct attributes exist. So when an entity is being stored, the columns which are not part of the entity will be filled with `null` values what we certainly wanted to avoid. The `TABLE_PER_CLASS` strategy is similar to the `JOINED` strategy except that the parent class is not created as a table but all properties of the subclasses including the inherited one's are mapped to columns in the class's table in the database.

Due to applying *polymorphism* and when selecting activity-specific data in the database, the *Java Persistance API* needs a way to differentiate between a row representing an object of one class and a row representing an object of another. Therefore, the `JOINED` strategy provides a supplement `DTYPE` property in the parent class. It is a so-called *Discriminator Column* which stores the name of the subclass of an activity and determines what class of object each row in the database table represents.

In the following figure we illustrate a conceptualization for managing quizzes.

[1]Set the property `<property name="eclipselink.ddl-generation" value="create-tables"/>` in the persistance deployment descriptor to enable.

[2]There exists three types of strategies: `SINGLE_TABLE, JOINED or TABLE_PER_CLASS`

[3]Achieved with writing `@Inheritance(strategy=InheritanceType.JOINED)` above the class declaration.

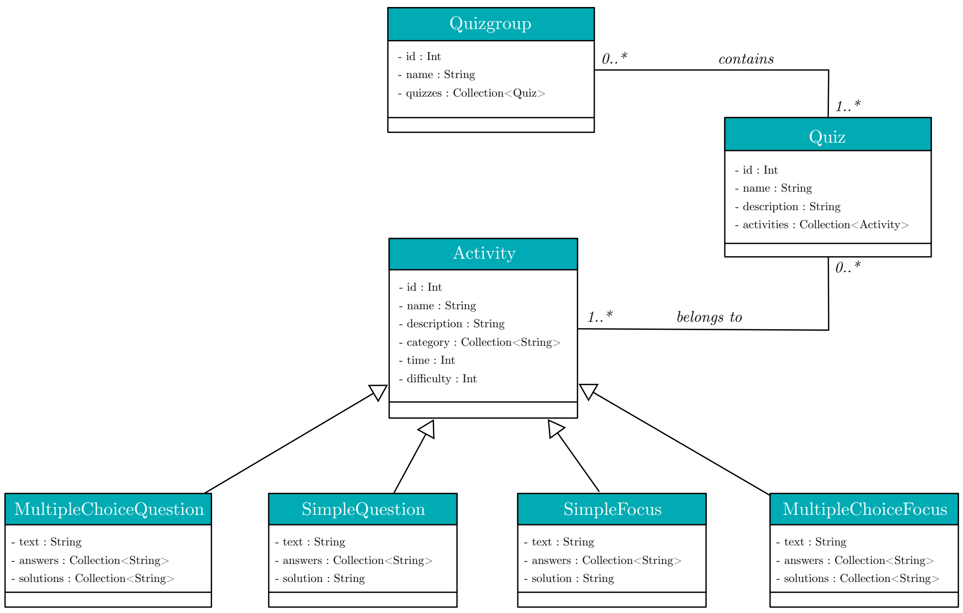


Figure 3.3: Quizzes – UML Class Diagram

We created the Java entities *Quiz* and *Quizgroup*. The Quiz class can be seen as a container for activities and the Quizgroup entity represents a container for quizzes. For example, a teacher creates two quizzes with the name ”Operating Systems 2 - week 1” and ”Operating Systems 2 - week 2”. Of course there can be many more quizzes associated with several lectures. In order to group quizzes associated with the same lecture, the teacher can create a group of quizzes (Quizgroup) called ”Operating Systems 2” and put all the quizzes belonging together in the that Quizgroup.

Whenever a query on the Quiz table is executed, the hole collection of activities will be fetched as well and one can access them via the property activities. However, due to missing subclasses we cannot get instances of concrete activities since they are stored in the modules outside our application. Hence, the collection of activities in a Quiz object will be empty. We solve this problem by *delegation* which is described in following section.

3.1.3 Web Services

The layer of the web services represents the *middleware* between the clients and the server. It also builds an interface for communicating with modules. All local or external requests go through the web services layer. For security reasons, we use *prepared statements* on the parameters coming from the requests to avoid *SQL-injection*. We establish an *injective mapping* from our REST web service to the REST interface of an activity when pairing our tool with a module. Whenever information about a particular activity is required, a request is *delegated* from our REST web service using it's corresponding base REST-URL in the registration entry to the activity application which in turn sends a response in JSON data format containing the required information. We agreed on JSON as a standard data format for all responses in REST-calls. The responses will be forwarded to the student/teacher-view of the module's client-side through websockets. This approach is elaborated in section 3.2. In figure 3.4 a more detailed architecture focused on the web services layer is illustrated.

Well-formed REST Interfaces

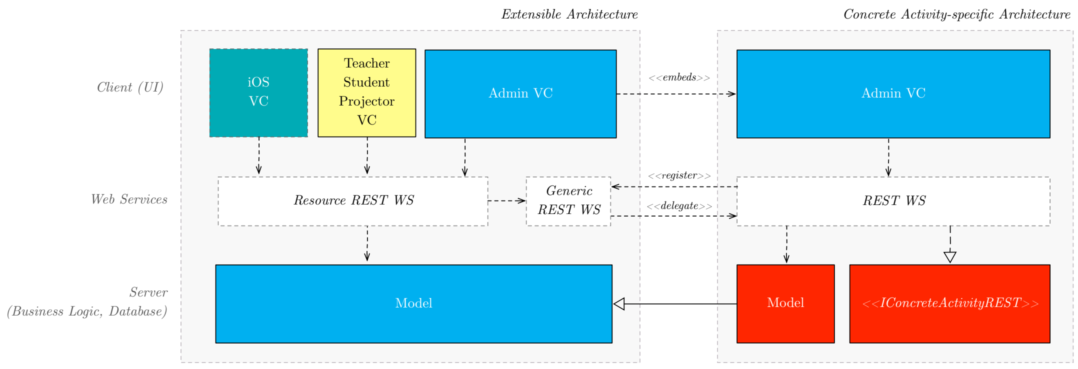
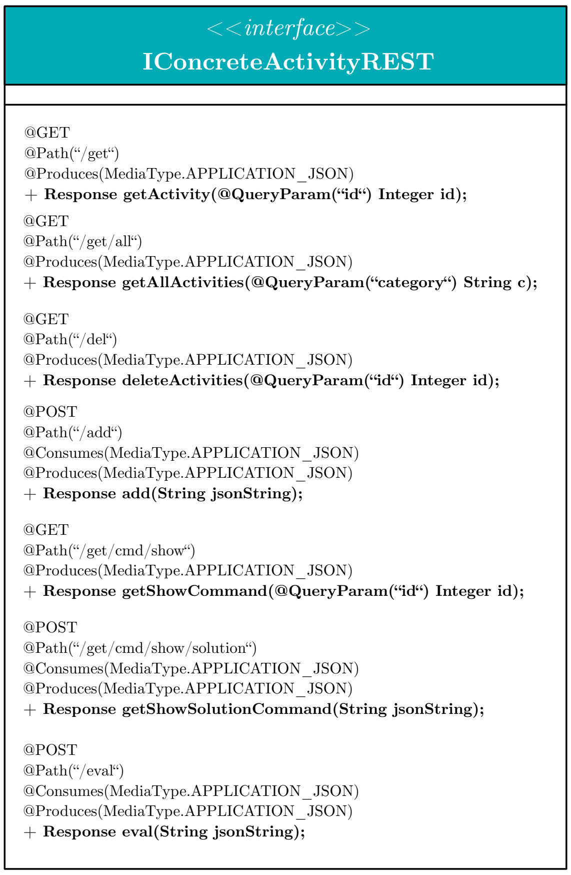


Figure 3.4: Modular Architecture – Web Services

There are many cases when we need to fetch activity-specific data. For instance all activity has another view. This information is also fetched via delegation and the response is directly forwarded to the client-side which then renders the view accordingly. Another example is the evaluation of a question. For each activity the evaluation process might be different and hence is computed on the server-side of the module. Since there are many other purposes for requesting activity-specific data, we need a *well-formed* interface between the REST service of our tool and all the other REST services from the modules. The result of our analysis consists of a Java interface (`IConcreteActivityREST.java`) which is implemented by all REST Java classes of all activities. This interface provides common methods for querying data. At the moment there are several CRUD methods and some methods for communicating with the client-side UI such as showing a particular question, showing the solution together with some statistics and finally evaluating the responses from the students for a specific question. We provide an UML Class Diagram of the interface in the figure 3.5.



```
@GET
@Path("/get")
@Produces(MediaType.APPLICATION_JSON)
+ Response getActivity(@QueryParam("id") Integer id);
@GET
@Path("/get/all")
@Produces(MediaType.APPLICATION_JSON)
+ Response getAllActivities(@QueryParam("category") String c);

@GET
@Path("/del")
@Produces(MediaType.APPLICATION_JSON)
+ Response deleteActivities(@QueryParam("id") Integer id);
@POST
@Path("/add")
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
+ Response add(String jsonString);

@GET
@Path("/get/cmd/show")
@Produces(MediaType.APPLICATION_JSON)
+ Response getShowCommand(@QueryParam("id") Integer id);
@POST
@Path("/get/cmd/show/solution")
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
+ Response getShowSolutionCommand(String jsonString);

@POST
@Path("/eval")
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
+ Response eval(String jsonString);
```

Figure 3.5: IConcreteActivityREST – UML Class Diagram

The first four methods provided in the interface are performing CRUD operations. Reading and deleting an object is done via HTTP GET calls. Creation of an instance takes place via POST method since we do not know the exact parameters for a GET invocation for each activity in the run. Providing simply a JSON String as parameter, allows us to bundle all parameters for a creation of an instance in one object. The parsing into an object and the interpretation of the JSON string is done specificly in each implementation.

The methods `getShowCommand` and `getShowSolutionCommand` are used to build the JSON object to be sent to the clients where the interpretation and rendering of a particular activity is done. Finally, the `eval` method takes a JSON string corresponding to the submission of a student. It is used to evaluate the submission data and to return a JSON containing a local score which will be adjusted in the server side of Yactul.

With regard to the web service layer of our tool, we created two RESTful Java classes to seperate the different tasks between local requests and external requests. The local one's go through the *Resource-REST* and the external requests are forwarded from the *Resource-*[4]*REST* to the *Generic*[5] *REST*. The REST web service in the module implements the `IConcreteActivityREST` interface and therefore the *Generic-REST* knows how to build the URL corresponding to a specific request.

Delegation

The delegation process enables our microservice to concentrate only on performing local tasks whether module-specific tasks are being hand off. Delegation is always done when more data as given in the parent activity class is needed. This is clearly the case during a quiz which is composed of several activities because their subclasses are not stored in the backend of our tool. The most important information which is needed for the delegation is the *id* of an activity and the *purpose of delegation*. The id is used to fetch the needed base URL for the REST-call and eventually to be passed as `GET` parameter. The purpose of delegation is required for contructing the rest of the URL. It follows an example:

`http://localhost:8080/SimpleQuestion/rest/get?id=413`

The id is used to find the activity in the database. We can then read it's type – in this example "SimpleQuestion" – and query the REST-URL corresponding to the activity from the database. The purpose for the request terminates the URL with the corresponding method invocation. Here we want to retrieve an activity of type `SimpleQuestion` with an id of 413.

On the server-side of Yactul, we cannot instantiate the fetched activities in a quiz due to missing subclasses. The subclasses are completely unknown in our server. The *Java Persistance API* (JPA) tries to directly instantiate an object based on a query result. Therefore, we use native SQL-queries to fetch the id's of activities for a certain quiz. For each activity-id, we request the activity instance represented in JSON data format from the REST interface corresponding to the activity. The result is being forwarded to the initial requestor without interpretation. It follows an UML Sequence Diagram for illustration.

[4]We call it *Resource-REST* because it accesses the model which is a representation of a resource.

[5]*Generic* because it can handle any activity-specific data which is completely unknown.

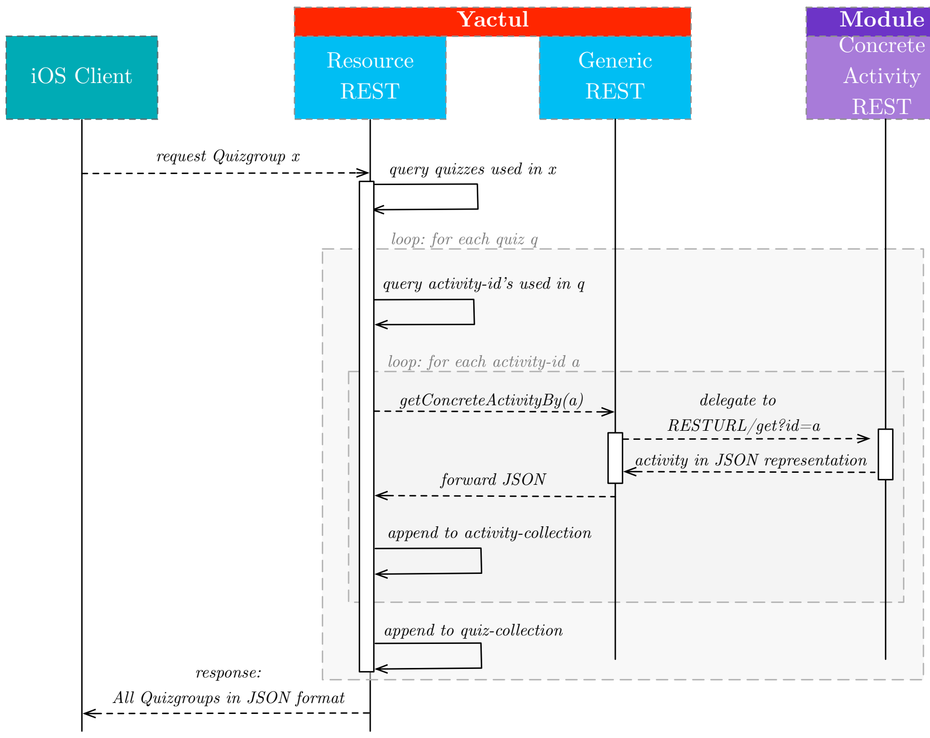

Yactul
Module
iOS Client
Resource REST
Generic REST
Concrete Activity REST
request Quizgroup x
query quizzes used in x
loop: for each quiz q
query activity-id's used in q
loop: for each activity-id a
getConcreteActivityBy(a)
delegate to RESTURL/get?id=a
activity in JSON representation
forward JSON
append to activity-collection
append to quiz-collection
response: All Quizgroups in JSON format

Figure 3.6: Delegation – UML Sequence Diagram

A mobile client requests a specific group of quizzes x via REST. Here x represents an id of a concrete Quizgroup instance. The Resource-REST queries all the quizzes from the database which are used in x. However, the collection of activities in the Quiz instances remains empty. Having the quizzes, we can hence query only the activity-id's which are used in those quizzes to avoid querying instances of activities. For each activity-id, the Generic-REST sends a request to the REST web service of the module associated with the activity. The needed REST-URL is queried by matching the dtype of the activity with the activityname column from the registredrest table. The result consisting of an activity in JSON data format will be stored in a list. This process is repeated as long there are quizzes remaining. Finally the requested Quizgroup with quizzes will be sent in JSON format together with the corresponding activities.

Showing the UI of an Activity

The teacher can start a quiz and this will trigger the rendering of the first activity at client-side. Since the activities are completly unknown, the rendering information is stored in the modules. Because of that, we use delegation to get the rendering information as JSON representation and send the result to the clients which will perform the interpretation and the drawing. The showing procedure is implemented via the method `getShowCommand` we provided in the interface as illustrated in figure 3.5.

Score Evaluation

The evaluation process of an activity is also done via delegation. Each activity-specific REST implements an evaluation method which takes as parameter a JSON in String format. This JSON is coming from the students when finishing a question. The answer will not be interpreted in the server-side of Yactul but rather delegated to the activity-specific REST. The evaluation method returns a *score* where $0 \le score \le 2$. We defined that a correct submission leads to 1 and the other remaining point is calclulated with consideration of some other factors such as the remaining time, the difficulty of the activity, etc. The final score well be, however, computed in our tool by simply multiplying the score coming from the evaluation with a *fixed constant* which can be easily configured in the server. This seperation of the score calculation has the advantage, that one can change the upper-bound for the score. We are using 1000 as fixed score, so the students can get 2000 points at best for each question. If another university wish to use small numbers, they can adjust the fixed score constant.

Showing the Solution of an Activity

Similar to the part of showing the UI of an activity, the procedure of showing the solution also uses delegation to fulfill this purpose. The solution is normally shown at the end of an activity together with a personalised statistics, that is, the chosen answer(s) of the student, the solution(s) and for each answer, the total number of students who selected that answer. Because we cannot interpret the submission, the hole statistics calulcation need to be performed in the part of the modules. We solve this by collecting the JSON submissions from the students and putting them together with the corresponding session-id to identify the student. Having collected all answers for one activity, the answers in form of an JSON array are sent via delegation to the REST of the module corresponding to the activity. The module performs the counting for the statistics and for each JSON answer, it creates individually for each student another JSON object containing the information for the solution. Each JSON solution object is redirected to corresponding student given by the session-id which were collected in the beginning of the submission. The solution procedure is implemented via the method `getShowSolutionCommand` in the interface `IConcreteActivityREST` as illustrated in figure 3.5.

3.1.4 Client-Side Implementation

The client layer of our architecture consists of three parts: the *Administration View Controller*, *The Student/Teacher View Controller* and the *iOS mobile application*. The Student/Teacher-VC is responsible for rendering the UI of a specific activity. It also handles user interaction such as touches, sending answers, etc. The Admin VC is used by the teachers to perform CRUD operations on quizzes and activities. Teachers have the possiblity to manage quizgroups, quizzes and activities. In order to modify a specific activity, the Admin VC from the corresponding module is *embedded*. The teacher has thus the possibility to perform any desired modification on an instance of an activity since he is now at the client-level in the module application as illustrated in the following figure.

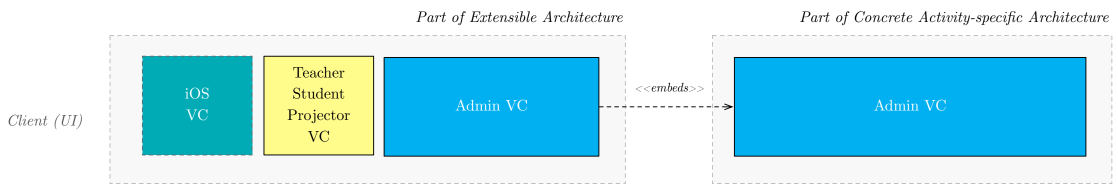

Part of Extensible Architecture
Part of Concrete Activity-specific Architecture
Client (UI)
iOS VC
Teacher Student Projector VC
Admin VC
<<embeds>>
Admin VC

Figure 3.7: Modular Architecture – Client UI

We implemented the *embed* functionality as a redirection. The teacher will hence be redirected to the administration web page of a module when he clicks on an activity that he would like to modify. Since the teacher is not anymore in the Yactul application, he can make use of any functionality provided by the module. When finishing the modification of an activity, the teacher can return to our tool.

The mobile clients access data via REST as shown in the figure 3.8. The REST web service delegates the request to the corresponding module application and forwards the result to the iOS client. The exact Java subclass as used in the module needs to be written in the mobile clients in the corresponding programming language as well in order to be casted from the fetched data using the discriminator value into activity instances for further treatement. Same for the Student/Teacher-VC because it needs to know the model to accordingly render the corresponding UI. Note that the iOS VC represents the same client.

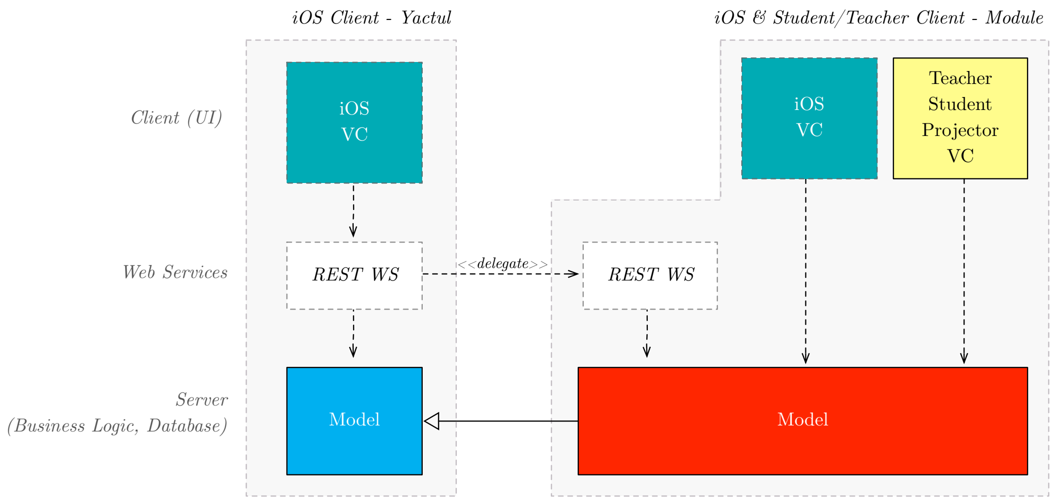
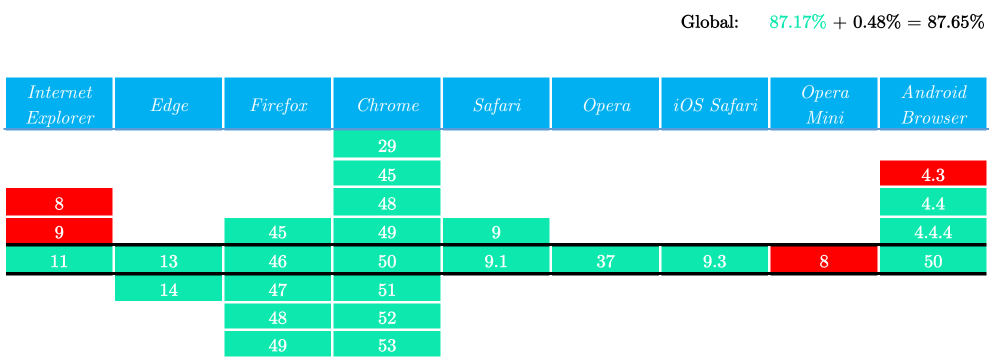


Figure 3.8: Modular Architecture – iOS & Student/Teacher Client

The students and the teachers communicate through websockets instead of REST. This process is elaborated in the following section.

3.2 Message Flow

Besides the modular architecture, the message flow during a quiz is a major aspect in our tool. Both teachers and students are clients and do not communicate directly with each other but rather through the server. We analyzed the tools described in the state of the art and both Kahoot and Quizizz are using *WebSockets* as communication protocol. Socrative in the other hand is using a HTTP persistent connection, also called *HTTP keep-alive*. Both techniques support a persistent connection. However, the key difference is that HTTP keep-alive still follows the HTTP request-response schema but an opening websocket setups a *full-duplex* connection which makes a request-response protocol unnecessary. In order to allow that teachers can push data to the students – which is obviously needed during a quiz – and to reduce the overhead of request-response messages, we decided to use WebSockets as communication protocol which are quite popular and supported by many browsers as seen in figure 3.9 [12].



Global: 87.17% + 0.48% = 87.65%

Internet Explorer	*Edge*	*Firefox*	*Chrome*	*Safari*	*Opera*	*iOS Safari*	*Opera Mini*	*Android Browser*
			29					
			45					🟥 4.3
🟥 8			48					4.4
🟥 9		45	49	9				4.4.4
11	13	46	50	9.1	37	9.3	🟥 8	50
	14	47	51					
		48	52					
		49	53					

Figure 3.9: Websocket – Global Support Overview

3.2.1 Websockets

Both students and teachers communicate through websockets. For simplicity and code management reasons, we separated the endpoint where clients open a websocket connection into two endpoints: one for the teachers and one for the students. The tools presented in the State-of-the-Art chapter require a pin to open a connection and then the system is asking for a nickname. We thought about to combine these steps and therefore when attempting to open a connection, the students must provide a nickname and a pin. The teachers in the other hand only need to provide a username. We differentiate here between nickname and username since teachers have an account and therefore can login into our system and students are anonymous with no account. As mentioned above, the teachers do not communicate directly with the student. The communication flow passes through the server where the messages are being processed and forwarded. The message flow is illustrated in the figure 3.10.

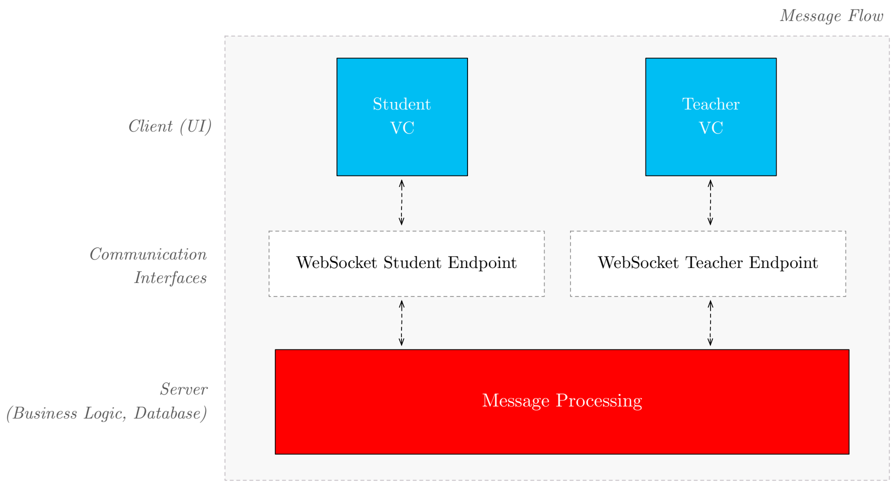

Message Flow
Client (UI)
Student VC
Teacher VC
Communication Interfaces
WebSocket Student Endpoint
WebSocket Teacher Endpoint
Server (Business Logic, Database)
Message Processing

Figure 3.10: Message Flow

Websockets can send objects or literals. We use the same data format as for the REST communication, that is JSON, because it is very lightweight and has build-in support in JavaScript which is heavly used in the Student/Teacher-VC for rendering the UI and for the websocket message handling at client side. At the client-side, a JSON object is parsed as a string literal and is sent through the websocket object. At server-side, the message is captured in the websocket endpoints and a JSON object is contructed or parsed based on the string representation. For security reasons, the incoming inputs are properly escaped to avoid both HTML code and XSS JavaScript attacks.

After providing a pin and a nickname, the websocket enpoint for students processed the message. This includes checking if a quiz corresponding to the entered pin exists, whether the nickname already exists for the quiz, etc. During a quiz, the teacher can send messages such as showing a question, showing the solution of a question, etc. which are being redirected to the students participating to the corresponding quiz. Students also communicate with the teacher for instance by answering questions.

The websocket endpoints can be considered as interfaces between the server and the clients. All incoming messages go through the endpoints. They also handle opening requests and closing procedures. In order to avoid putting all the code for the message processing inside the websocket endpoints, we encapsulated the business logic into several *objects*. The websocket endpoint classes are thus very lightweight and delegate the tasks to specific objects which handle the further treatement of the message. In the following section, we present how those objects work and how to easily add new objects for handling new tasks without changing code in the core application.

3.2.2 Message Processing

During a quiz, there are a lot of messages floating around. Each instruction from the teacher requires a message or even each answer from the students also requires a message to be sent. We thought about the messages as *intructions* for the server. For instance, a teacher can start, end, pause or resume a quiz. Students in the other hand can send answers to questions which also can be seen as an instruction because the server needs to understand the message, evaluate it and notify the teacher. Therefore we came up with a concept consisting of an abstract class called *Command* with a property `cmd` to be used as an unique *key* and a abstract *process* method for processing the message given as parameter. In the figure 3.11 an UML Class Diagram of the presented Command class is illustrated.

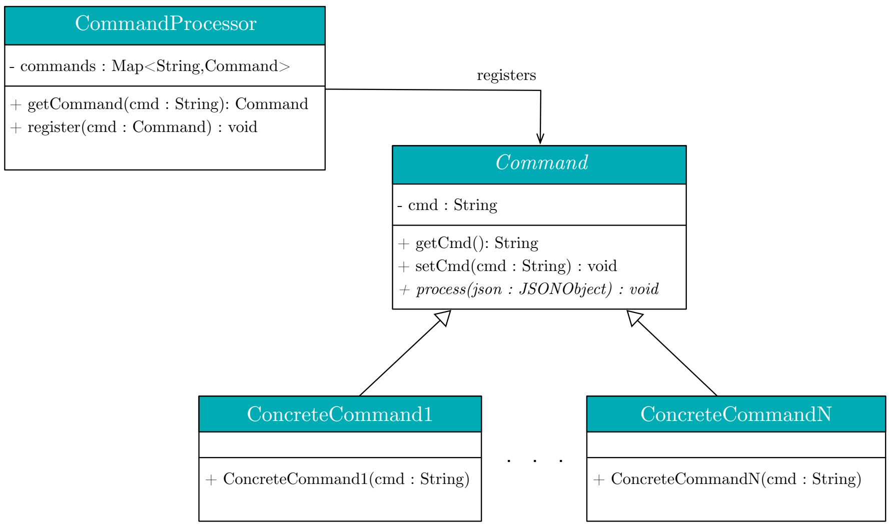

CommandProcessor
- commands : Map<String,Command>
+ getCommand(cmd : String): Command
+ register(cmd : Command) : void
registers
Command
- cmd : String
+ getCmd(): String
+ setCmd(cmd : String) : void
+ process(json : JSONObject) : void
ConcreteCommand1
+ ConcreteCommand1(cmd : String)
. . .
ConcreteCommandN
+ ConcreteCommandN(cmd : String)

Figure 3.11: Abstract Command – UML Class Diagram

The commands are being registrated by the *Command-Processor* as soon as the application starts. The Command-Processor *Singleton* object can be seen as a container for the commands. It uses a *HashMap* of key-value pairs for accessing a command object by it's property `cmd`. The registration allows adding as many distinct commands as needed. The command object will be stored as a value and the String property `cmd` will be used as a key in the HashMap of the Command-Processor. Whenever a message as JSON-String arrives in the websocket endpoint, the *injected* Command-Processor invokes the process method of the command accessed by the `cmd` key which is read from the message. The business logic for the corresponding instruction is outsourced from the websocket endpoint and encapsulated in the `process` method of a command.

In the figure 3.12, some Command classes are listed together with a small description about the business logic in the `process` method.

Command	Business Logic of process method
	From students
WebSocketConnectCommand	This command processes opening connection attempts. It handles some verification routines such as checking if an open quiz exists with the given pin, if the username is not already taken, etc. If all test cases pass, an instance for the student is created and added in the system to maintain it's connection. If not, appropriate messages are sent back.
GroupNameCommand	The process method of this command handles incoming messages when students have chosen a groupname. The groupname will be associated with the student in the system.
SubmitCommand	This command is the most important arriving from the students. The process method is being invoked when students finish an activity. The evaluation process will be done from here via REST using delegation.
	From teachers
StartQuizCommand	When the teacher starts a quiz, this command will be processed. It send a JSON message to all the students which causes to render the first activity.
EndQuizCommand	The processing of this commands ensures that an open quiz closes. This closing implies that all the open websockets of students will be closed as well.
GroupModeEnableCommand	This command enables the group-mode for a particular quiz. Hence, the students are asked to enter a groupname for collaborative playing.
UpdateCommand	The Update-Command is used to perform real-time changes such as adding or removing time or pausing or resuming the activity/quiz.
ShowSolutionCommand	The processing of this command results in showing the solution of the current activity to the students. The solution information is fetched via delegation.
DisableActivityCommand	The teacher has the possibility to disable or enable an particular activity via this command.

Figure 3.12: Commands Table

3.2.3 Playing a Quiz

In the following UML Class Diagram we illustrate some important Java classes which are involved during the procedure of a quiz.

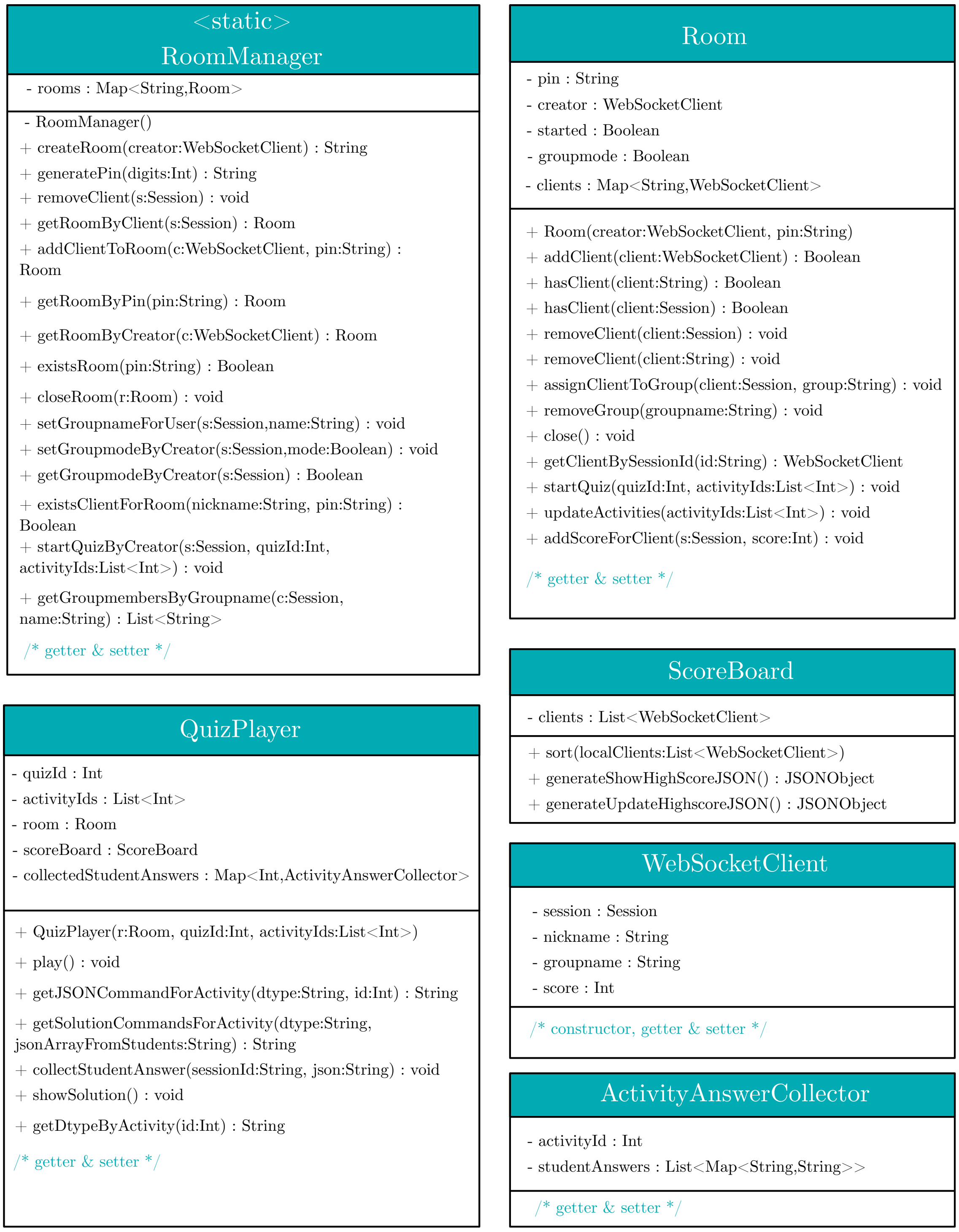

<static>
RoomManager
- rooms : Map<String,Room>
- RoomManager()
+ createRoom(creator:WebSocketClient) : String
+ generatePin(digits:Int) : String
+ removeClient(s:Session) : void
+ getRoomByClient(s:Session) : Room
+ addClientToRoom(c:WebSocketClient, pin:String) : Room
+ getRoomByPin(pin:String) : Room
+ getRoomByCreator(c:WebSocketClient) : Room
+ existsRoom(pin:String) : Boolean
+ closeRoom(r:Room) : void
+ setGroupnameForUser(s:Session,name:String) : void
+ setGroupmodeByCreator(s:Session,mode:Boolean) : void
+ getGroupmodeByCreator(s:Session) : Boolean
+ existsClientForRoom(nickname:String, pin:String) : Boolean
+ startQuizByCreator(s:Session, quizId:Int, activityIds:List<Int>) : void
+ getGroupmembersByGroupname(c:Session, name:String) : List<String>
/* getter & setter */
Room
- pin : String
- creator : WebSocketClient
- started : Boolean
- groupmode : Boolean
- clients : Map<String,WebSocketClient>
+ Room(creator:WebSocketClient, pin:String)
+ addClient(client:WebSocketClient) : Boolean
+ hasClient(client:String) : Boolean
+ hasClient(client:Session) : Boolean
+ removeClient(client:Session) : void
+ removeClient(client:String) : void
+ assignClientToGroup(client:Session, group:String) : void
+ removeGroup(groupname:String) : void
+ close() : void
+ getClientBySessionId(id:String) : WebSocketClient
+ startQuiz(quizId:Int, activityIds:List<Int>) : void
+ updateActivities(activityIds:List<Int>) : void
+ addScoreForClient(s:Session, score:Int) : void
/* getter & setter */
QuizPlayer
- quizId : Int
- activityIds : List<Int>
- room : Room
- scoreBoard : ScoreBoard
- collectedStudentAnswers : Map<Int,ActivityAnswerCollector>
+ QuizPlayer(r:Room, quizId:Int, activityIds:List<Int>)
+ play() : void
+ getJSONCommandForActivity(dtype:String, id:Int) : String
+ getSolutionCommandsForActivity(dtype:String, jsonArrayFromStudents:String) : String
+ collectStudentAnswer(sessionId:String, json:String) : void
+ showSolution() : void
+ getDtypeByActivity(id:Int) : String
/* getter & setter */
ScoreBoard
- clients : List<WebSocketClient>
+ sort(localClients:List<WebSocketClient>)
+ generateShowHighScoreJSON() : JSONObject
+ generateUpdateHighscoreJSON() : JSONObject
WebSocketClient
- session : Session
- nickname : String
- groupname : String
- score : Int
/* constructor, getter & setter */
ActivityAnswerCollector
- activityId : Int
- studentAnswers : List<Map<String,String>>
/* getter & setter */

Figure 3.13: UML Class Diagrams for Playing a Quiz

Students who are joining a quiz with the right pin are stored in *WebSocketClient* instances. This class encapsulates most importantly the websocket session object which is needed for sending and receiving messages. As soon as a teacher opens a quiz, the *static RoomManager* creates an instance of a

Room. A Room instance represents a virtual Room where the quiz takes place. The return value of this procedure is a pin which the teacher can forward to the students in order to be able to join. The RoomManager can be seen as a *facade* for manipulating Room objects. When the teacher starts the quiz, an instance of *QuizPlayer* is created which handles the further process. The QuizPlayer also communicates with the local REST instances to make use of delegation. For instance, the method `getJSONCommandForActivity` sends a request to the Generic-REST to get the corresponding JSON command to trigger the rendering of a specific activity in the students' screen. The *Score-Board* class is used to represent the Highscore in the teacher's screen. It holds a local list of the current students which is being updated and sorted in descending order every time when a student submits an answer. Hence, the highscore is shown in real time. The *ActivityAnswerCollector* stores all the answers from the students for an activity. This is needed for the evaluation and for the statistics. The ActivityAnswerCollector only stores answers for a single activity. The QuizPlayer holds a HashMap of activity-id as a key and an ActivityAnswerCollector instance as value and thus, can store statistics for multiple activities. This HashMap can then be used for persistantly storing the statistics of a quiz.

In the following UML Sequence Diagram, we illustrate an example on how the several classes cooperate when the teacher starts the quiz.

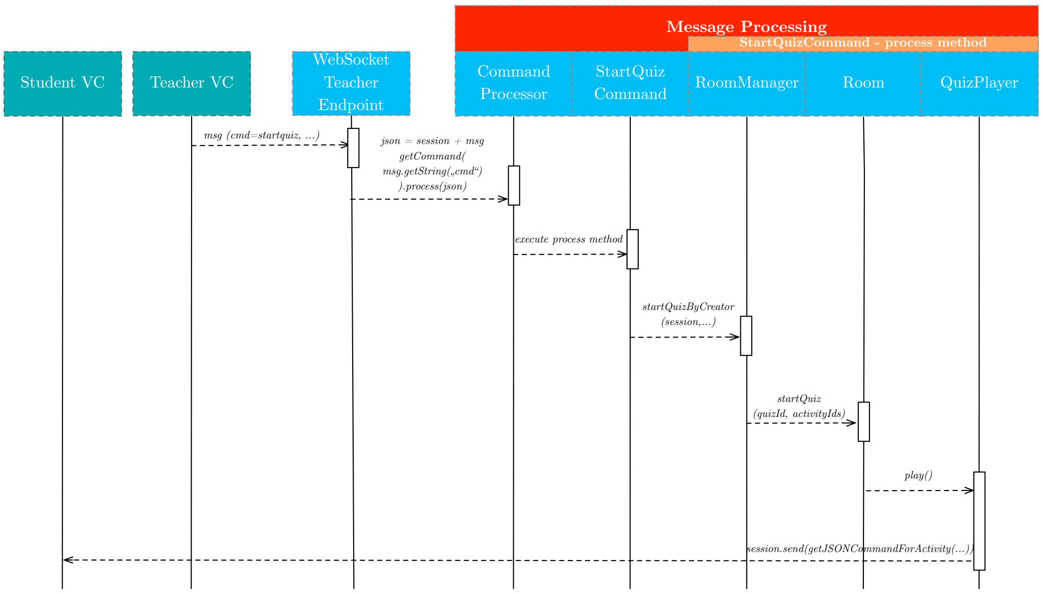

Message Processing
StartQuizCommand - process method
Student VC
Teacher VC
WebSocket Teacher Endpoint
Command Processor
StartQuiz Command
RoomManager
Room
QuizPlayer
msg (cmd=startquiz, ...)
json = session + msg getCommand(msg.getString(„cmd“)).process(json)
execute process method
startQuizByCreator (session,...)
startQuiz (quizId, activityIds)
play()
session.send(getJSONCommandForActivity(...))

Figure 3.14: UML Sequence Diagram – Start Quiz Example

As soon as the teacher clicks on a button for starting the quiz, a JSON message is sent containing the command `cmd=STARTQUIZ`, a *quizId* and an array of `activity-id's`. In the server-side, the message will be received at the teacher's websocket endpoint. The message will not be interpreted but directly packed into another JSON object together with the session object of the teacher in order to identify the sender. The CommandProcessor will read the cmd-value in the message and call the corresponding command's process-method. From here, the message handling is encapsulated in the process method. In this example, the process method of the StartQuizCommand is invoked. This involves searching for the Room object with the teacher's session with the help of the RoomManager and invoking the method `startQuiz` with the parameters given in the initial message. This causes that an instance of a QuizPlayer will be created and it's `play` method will be executed. Since the QuizPlayer has the Room object as property and thus, can access the clients, it can send the appropriate JSON command for rendering the first activity to all the clients of a given Room object. The JSON message is fetched via delegation since the construction of the message is done in the corresponding module.

This is of course only one example on how message processing with several classes work. Other messages are processed in a very similar way. Until the point where the CommandProcessor invokes the corresponding process method, the message processing is same for all other messages.

3.3 Testing and Validation

As soon as our tool reached a stable state, we tested it every week during meetings where we presented our progress. We organized live demos with 8-13 participants. In real-world, this sample is clearly not sufficient since the number of student varies between universities. However, at the moment the tool should be more considered as a *prototype* and a *proof of concept* for modeling a modular CRS. Nevertheless, we plan to use Yactul during lectures to test it with a more realistic sample.

The live demos served not only for progress illustration but also for detecting bugs. This helped a lot to improve our tool and to make it more stable and reliable. Considering security aspects, we analyzed potential *exploits*. The only communication with outside our application are the requests through REST and the messages coming from websockets. On the requests we use prepared statements in order to avoid SQL-injection and on the messages coming from the websockets, we perform sanity checks and escape both HTML as well as JavaScript code to avoid XSS-attacks.

3.4 Comparison with Current Tools

In this section, we want to describe some features that Yactul provides with reference to the research questions and to the comparison table as shown in chapter 2.

The research questions were:

RQ1 : How to build an *extensible* learning platform?

RQ2 : How to apply game mechanics in a learning platform?

RQ3 : How to overcome the limitations of current solutions?

RQ4 : How to combine the best features of existing systems while add new possibilities?

Joining a Quiz

All compared tools provide a mechanism to enter a pin for joining a quiz. There are of course alternatives. One alternative we thought about is to provide an URL. Hence, the students do not need to remember some digits but can access the quiz in a more intuitive way. However, the drawback of this variant is that a certain quiz cannot be played in parallel by two classes which are not in the same room since there is only one teacher projector. In our university, there are sometimes cases, especially in labs, where the classroom is split in two groups within two separate rooms. When using the mechanism with the pin, one can create two pins for the same quiz and thus, the quiz can be played in two separate rooms with two separate teacher projectors. Therefore, we decided to go with the pin. We fixed the length of the pin to four digits, allowing 10^4 simultaneously quizzes (rooms). Socrative is the only tool that generates pins based on numbers and letters. We wanted to avoid using letters because we believe that students should access a quiz as fast as possible and should not lose time due to a complicated pin. The length of the pin can be changed in case that more virtual rooms are needed since the pin generation process takes a number of digit as parameter. So there is no need to change any code in the core application.

General Features

Considering the features for a general question, our tool supports multiple choice questions with a variable number of answers. Note that only Socrative has this feature. Both Kahoot and Quizizz support up to four answers. Yactul supports from 2 to n answers but we suggest to not use more than eight because the rendered UI of the activity might become thereby very small.

Same as Kahoot, our tool supports pauses between questions so that the teacher has the possibility to interpret the results if desired. In terms of time per question, Kahoot and Quizziz support a *fixed range*. Kahoot allows a range from 5 to 120 seconds and Quizziz supports a range from 30 seconds to 15 minutes. Yactul cancels this restriction by storing the time in the base activity class and providing the teacher an input where he can enter the number of seconds needed. Same as socrative, we also created an offline version which consists of an iOS application where students can train and play quizzes without an active internet connection. The first time when the app launches, it fetches all the groups of quizzes via REST and stores them locally for offline use.

Due to the modular architecture, Yactul supports multiple activities. Currently we invented eight different activities as shown in figure 3.2. Four of them has been already implemented as proof of concept. Hence, the variety of activities is the answer to the most important research question **RQ1**. The multiple choice feature together with the possibility for a variable number of answers and a flexible time-range for a question, are answers for **RQ3**. The combination of best features such as the usage of a pin in order to join a quiz and the offer of an offline version are partially dedicated to **RQ4**.

Features during a Quiz

In order to comply with the research question **RQ4** on how to combine best features and add new possibilities, our tool supports – in contrast to all the compared CRS – modifications on quizzes during the life-cycle. This includes modifying the remaining time per question, pausing and resuming a question, changing the order of questions and disabling or enabling a particular question. We leave the modification of a question such as changing the solution, the question, the answers, etc. to future work.

Game mechanics

In terms of game-aspects, we have implemented three out of six possible game mechanics to satisfy **RQ2**. First of all we support points, which are given at each question and cumulated during the quiz. Yactul supports leaderboards, which are represented by the ScoreBoard class and shown in the projector of the teacher. Finally, the challenges game mechanic is implemented by some activities such as ”SimpleFocus” where the difficulty does not only rely on the question and the remaining time but also on the game since the answers are sequently printed on the screen with a flexible time-interval which really represents a challenge.

Student and Teacher View

As described in the State-of-the-Art chapter, Kahoot does not show the question itself in the students' screen due to pedagogical reasons. Also, the answers are not shown in textual form but rather as colors representing the same colors of the answers in the teacher's projector screen. Nevertheless we decided to show both question as well as answer in the student's view. The teacher's UI consists of a highscore with some settings showing the best five participants. The settings are used to change the order of the activities, to disable/enable a particular activity, to modifiy the current time, to stop/resume the current activity and to show the solution of the current activity.

Collaborative Learning

We would like to see students working together to solve problems since collaborative learning is based on the model that knowledge can be created within a population where members actively interact by sharing experiences and take on asymmetry roles [13]. We have implemented the basis data structure for supporting collaborative learning by adding the property groupname in the WebSocketClient class. The RoomManager implements some methods for fetching group members. Teachers can enable a ”groupmode” for a quiz which is set to true in the corresponding Room instance and hence, students will be promted to enter a groupname by using the GroupModeEnableCommand. Other students with the same groupname will be considered as a joint group. However, the evaluation of groups is still missing. We leave this to futue work.

	Classroom Response Systems			
Criteria	Socrative	Kahoot	Quizizz	Yactul
Entering room	*pin*	*pin*	*pin*	*pin*
Multiple Choice	✓	✗	✗	✓
Number of answers	*2..n*	*2..4*	*2..4*	*2..n*
Time limit per question	✗	✓	✓	✓
User defined time limit	✗	*fixed range*	*fixed range*	*flexible range*
Pause between questions	✗	✓	✗	✓
Offline version	*Mobile App*	✗	✗	*iOS App*
Multiple question types	✗	✗	✗	✓
During quiz				
Modify time	✗	✗	✗	✓
Pause/Resume question	✗	✗	✗	✓
Change order of questions	✗	✗	✗	✓
Modify question	✗	✗	✗	✗
Disable question	✗	✗	✗	✓
Gamification aspects				
Points	✗	✓	✓	✓
Levels	✗	✗	✗	✗
Challenges	✗	✓	✗	✓
Virtual Goods	✗	✗	✓	✗
Leaderboards	✗	✓	✓	✓
Gifting & Charity	✗	✗	✗	✗

Figure 3.15: Comparison Table with Yactul

4 | Conclusion

4.1 Summary

Gamification is a new technique for motivating people in classrooms by using game content in non-games. There are some popular web-based learning platforms providing gamification. However, they have several limitations in different areas such as only providing one type of question or restricting the user in many simple settings like non-support for multiple choice or the limitation of the number of answers for a question. Functionalities during a quiz are non-present at all. These restrictions lead us to the research questions we stated in the Introduction section. One of the research questions was how to overcome the limitations of current solutions? Thereby, we presented in the State-of-the-Art chapter some existing tools and we provided a comparison in terms of restrictions and gamification aspects. The presented restrictions are the key points where we wanted to focus on in order to improve. We cancelled a lot of restrictions as listed in the comparison table. Then, we wanted to apply gamification as much as possibly. We came up with eight different activities providing different game elements where we prototypicality implemented four of them. We also asked ourselves how to combine existing features while adding new possibilities? We have achieved this by reimplementing the best known features and providing interactive functionalities during a quiz such as modifying the remaining time or pausing and resuming a question which can be very convenient for teachers.

In this work, we propose Yactul, a web-based learning platform with a modular architecture which is the answer to the first and most important research question on how to build an extensible learning platform to allow multiple types of questions. We deploy our tool as a microservice and each question, as an independent application running as a separate module that can be easily coupled with our tool. The communication is done via REST, a language-agnostic API, that uses delegation as main concept. Hence, our tool can communicate with activities that are programmed in any programming language and that are running in any server on any location.

Yactul is still in it's prototype phase but has shown that it can keep up with current popular learning platforms. The comparison results highlight that our approach outperforms some popular tools as shown in the comparison table.

4.2 Future Work

In the future, we plan to extend the set of activities and the functionalities which can be triggered during a quiz in order to offer more selection to the students and teachers. We also want to try to apply as many game mechanics as possible without conflicting with the resulting human desires. So far, three out of six different game mechanics are implemented. Moreover, we want to implement an user management system which is necessary so that the tool can be used personally by multiple teachers. Another important point which we will work on is the storage of the statistics. We provide a class collecting the results for each student per activity, but we do not yet persistantly store the information and show it to the teacher in a structured way. Furthermore we also want to focus on collaborative learning. Currently our datastructure supports grouping of students and only needs to be expanded in terms of evaluation since the score calculation for a group of students is different from the evaluation for individuals.

Lastly our objective is to make Yactul available as Open Source to enable a free distribution of the source code to drive innovation.

Glossary

Activity	A separate and independent application representing a specific type of question.
API	Set of programming instructions and standards for accessing a Web-based application.
Exploit	An attack on a computer system that takes advantage of a vulnerability.
Full-Duplex Connection	Enables transmition of data in both directions at the same time.
Foreign Key	Field in one table that points to a Primary Key in another table.
Gamification	The application of game-design elements and game principles in non-game context.
Injective mapping	A distinct one-to-one mapping from an element of a set A to an element of a set B.
Microservice	Software applications that run as independently deployable services.
Polymorphism	Provision of a single interface to entities of different types in programming languages.
Primary Key	Constraint that uniquely identifies each record in a database table.
REST	The application of game-design elements and game principles in non-game context.
Sanity Check	Simple test to evaluate rationality on inputs.
SQL-injection	A code injection technique Injection which inserts user- supplied data to an interpreter as part of a command or query.
XSS	Type of computer security vulnerability in web applications. XSS is basically JavaScript-injection.

List of Abbreviations

API	Application Programming Interface
CRS	Classroom Response System
CRUD	Create, Read, Update and Delete
DBMS	Database Management System
HTTP	Hypertext Transfer Protocol
JPA	Java Persistance API
JSON	JavaScript Object Notation
REST	Representational State Transfer
SRS	Student Response System
UI	User Interface
UML	Unified Modeling Language
URL	Uniform Resource Locator
VC	View Controller
WAR	Web-Archive
XSS	Cross-Site-Scripting
Yactul	Yet Another Classroom Tool from the University of Luxembourg

Yactul – User Interface

Figure 1: Yactul – Logo

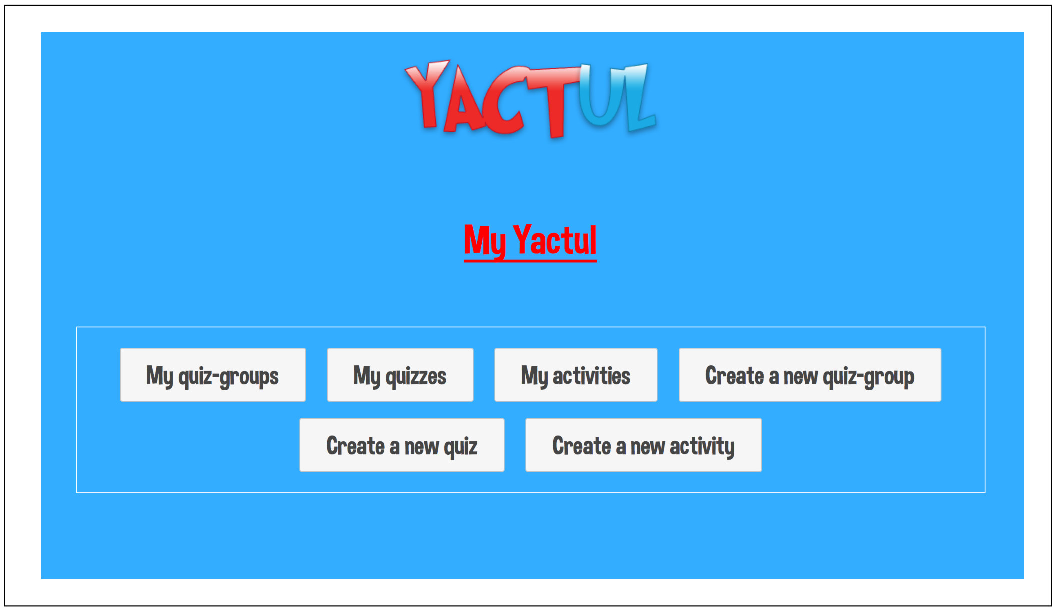
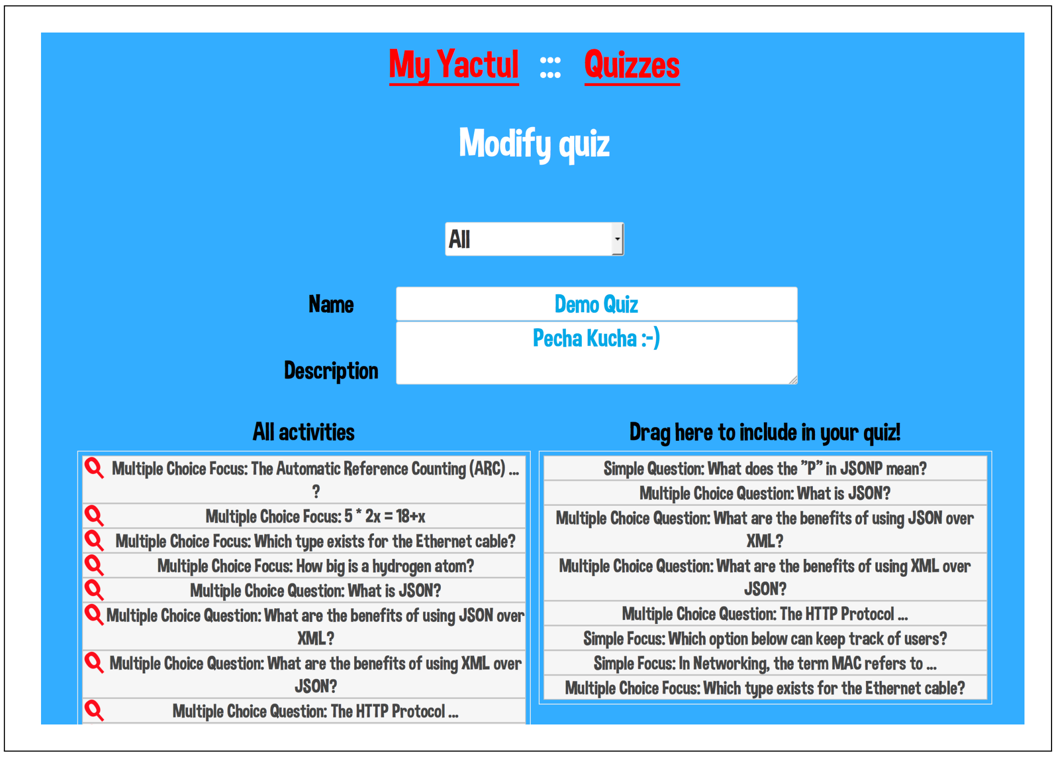


Figure 2: Yactul – Teacher Administration



My Yactul ::: Quizzes

Modify quiz

All

Name: Demo Quiz

Description: Pecha Kucha :-)

All activities	Drag here to include in your quiz!
Multiple Choice Focus: The Automatic Reference Counting (ARC) ... ?	Simple Question: What does the "P" in JSONP mean?
Multiple Choice Focus: 5 * 2x = 18+x	Multiple Choice Question: What is JSON?
Multiple Choice Focus: Which type exists for the Ethernet cable?	Multiple Choice Question: What are the benefits of using JSON over XML?
Multiple Choice Focus: How big is a hydrogen atom?	Multiple Choice Question: What are the benefits of using XML over JSON?
Multiple Choice Question: What is JSON?	Multiple Choice Question: The HTTP Protocol ...
Multiple Choice Question: What are the benefits of using JSON over XML?	Simple Focus: Which option below can keep track of users?
Multiple Choice Question: What are the benefits of using XML over JSON?	Simple Focus: In Networking, the term MAC refers to ...
Multiple Choice Question: The HTTP Protocol ...	Multiple Choice Focus: Which type exists for the Ethernet cable?

Figure 3: Yactul – Teacher Administration - Edit Quiz

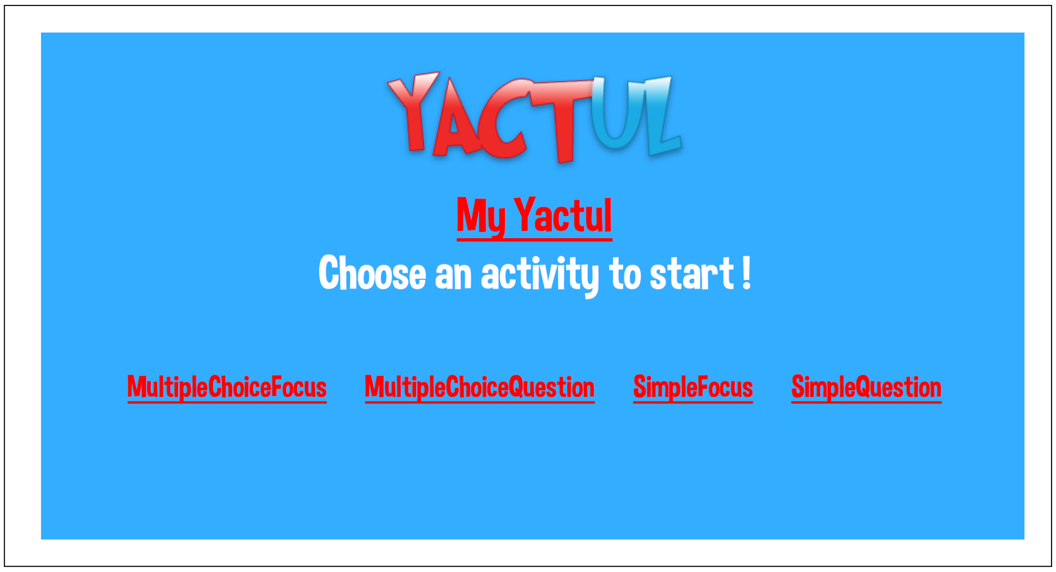
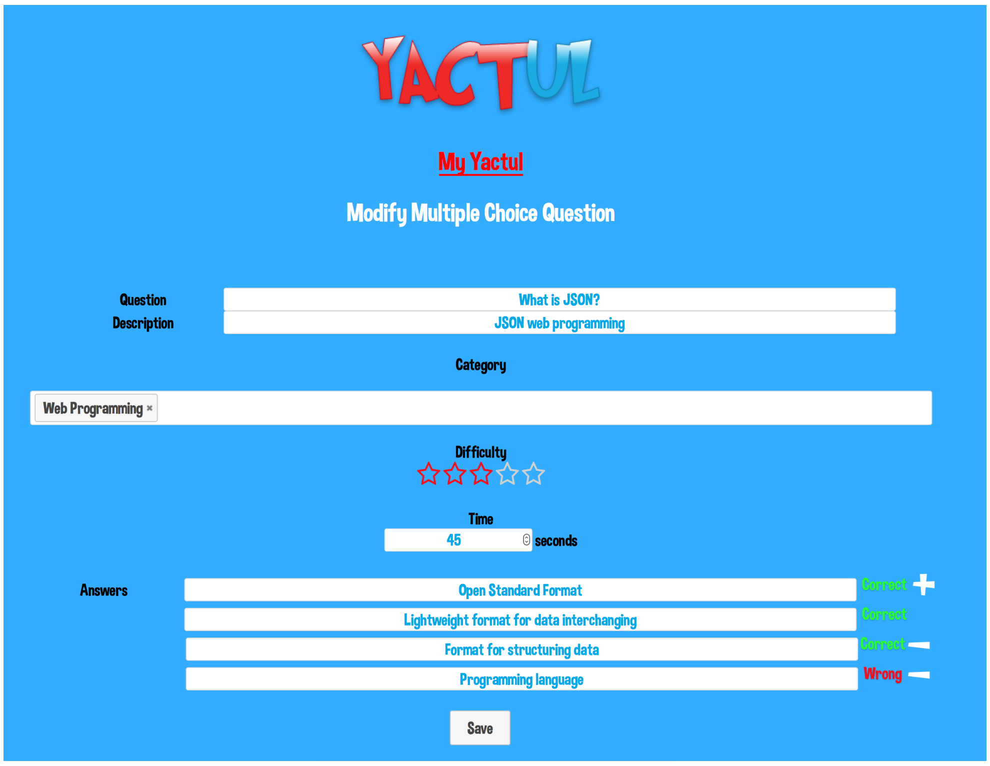


Figure 4: Yactul – Teacher Administration - Activities


My Yactul

Modify Multiple Choice Question

Question: What is JSON?
Description: JSON web programming

Category: Web Programming ×

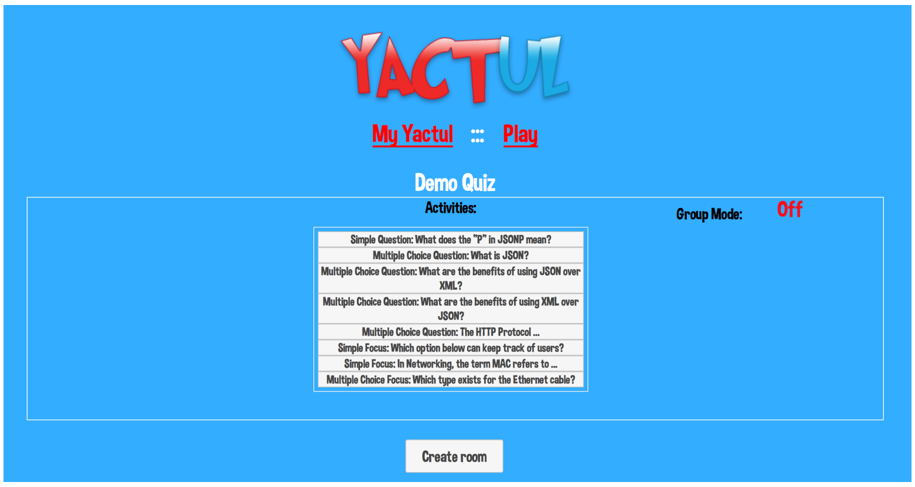
Difficulty

Time: 45 seconds

Answers:
- Open Standard Format — Correct
- Lightweight format for data interchanging — Correct
- Format for structuring data — Correct
- Programming language — Wrong

Save

Figure 5: Yactul – Multiple Choice Question Module - Creation


My Yactul ::: Play

Demo Quiz

Activities:
- Simple Question: What does the "P" in JSONP mean?
- Multiple Choice Question: What is JSON?
- Multiple Choice Question: What are the benefits of using JSON over XML?
- Multiple Choice Question: What are the benefits of using XML over JSON?
- Multiple Choice Question: The HTTP Protocol ...
- Simple Focus: Which option below can keep track of users?
- Simple Focus: In Networking, the term MAC refers to ...
- Multiple Choice Focus: Which type exists for the Ethernet cable?

Group Mode: Off

Create room

Figure 6: Yactul – Teacher Administration - Play Quiz

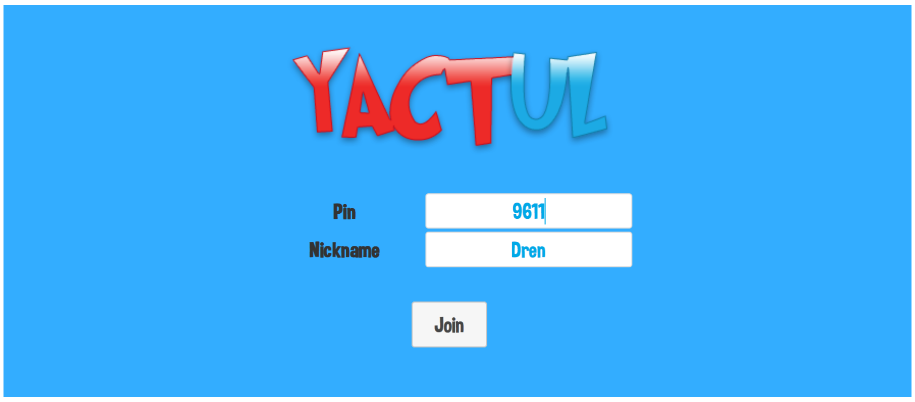


Figure 7: Yactul – Student - Join Quiz

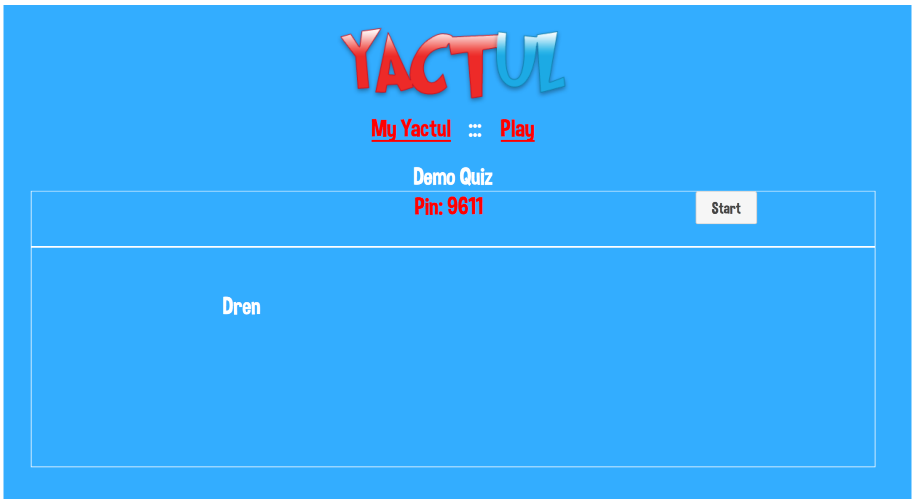


Figure 8: Yactul – Teacher Administration - Start Quiz

Fiches de Suivi de Stage

Université du Luxembourg
Faculté des Sciences, de la Technologie et de la Communication
Bachelor en Informatique (*professionnel*)

TRAVAIL DE FIN D'ETUDES

FICHE DE SUIVI DE STAGE

L'étudiant(e) :

NOM, Prénom	............GASHI Dren.............
Numéro matricule	............0130732528.............

L'établissement d'accueil :

NOM	Université du Luxembourg - FSTC

Responsable local de stage :

NOM, Prénom	............BOTEV Jean............

Responsable académique de stage :

NOM, Prénom	......ROTHKUGEL Steffen.......

Période :

Du	............16/02/2016.............
Au	............06/03/2016.............

Document à compléter et à envoyer (même non encore signé) par email au responsable académique, avec copie au responsable local, le plus tôt possible après la fin de chaque période. Une période correspond à trois semaines de stage.

Ce document complété et signé devra figurer en annexe du mémoire de fin d'études.

Université du Luxembourg
Faculté des Sciences, de la Technologie et de la Communication
Bachelor en Informatique (*professionnel*)

Travaux effectués pendant la période :

— **Research:**

– *Gamification*: Human Desires in games as 'game mechanics'.

– *State-of-the-Art*: Web-based Classroom Response Systems (CRS)

– *Communication of CRS's*: HTTP 'keep-alive' vs HTTP Web Socket

— **Comparison of CRS's:**

– *Candidates*: Socrative vs Kahoot! vs QUIZIZZ

– *Criteria*: features, nr. of game mechanics used, features *during quiz*, etc.

— **Logo creation:**

– YACTUL: Yet Another Classroom-feedback Tool from the University of Luxembourg

– *Tool* : Adobe Photoshop CC 2015

(*Si pertinent*) Travaux prévus pour la période suivante :

— **Design:**

– *Database Architecture:* UML Class Diagram of Relational Database

– *YACTUL Architecture:* Software components / Web services / State-Diagram etc.

Université du Luxembourg
Faculté des Sciences, de la Technologie et de la Communication
Bachelor en Informatique (*professionnel*)

	Le responsable local,	L'étudiant(e),
Date :	10/03/2016	10/03/2016
Signature :	[illegible]	[illegible]

Université du Luxembourg
Faculté des Sciences, de la Technologie et de la Communication
Bachelor en Informatique (*professionnel*)

TRAVAIL DE FIN D'ETUDES

FICHE DE SUIVI DE STAGE

L'étudiant(e) :

NOM, Prénom	…………GASHI Dren………….
Numéro matricule	…………0130732528………….

L'établissement d'accueil :

NOM	Université du Luxembourg - FSTC

Responsable local de stage :

NOM, Prénom	…………BOTEV Jean…………

Responsable académique de stage :

NOM, Prénom	……ROTHKUGEL Steffen…….

Période :

Du	…………07/03/2016………….
Au	…………28/03/2016………….

Document à compléter et à envoyer (même non encore signé) par email au responsable académique, avec copie au responsable local, le plus tôt possible après la fin de chaque période. Une période correspond à trois semaines de stage.

Ce document complété et signé devra figurer en annexe du mémoire de fin d'études.

Université du Luxembourg
Faculté des Sciences, de la Technologie et de la Communication
Bachelor en Informatique (*professionnel*)

Travaux effectués pendant la période :

— **Design:** ....

....

.... ***YFSM:*** A design of a finite state machine for YACTUL...................................

.... ***Architecure***: A design proposal of the architecture (WS, REST, DB, CRUD,...)...

.... ***UML-Class***: A design proposal of a database schema ....

....

....

(*Si pertinent*) Travaux prévus pour la période suivante :

— **Implementation:** ....

– *Server-side:* REST web-service, Web-Socket Java Endpoint............................

– *Client-side:* Web-Socket JavaScript client, Html View for joining a quiz...............

....

....

....

	Le responsable local,	L'étudiant(e),
Date :	06/04/2016	06/04/2016
Signature :	[illegible]	[illegible]

Université du Luxembourg
Faculté des Sciences, de la Technologie et de la Communication
Bachelor en Informatique (*professionnel*)

TRAVAIL DE FIN D'ETUDES

FICHE DE SUIVI DE STAGE

L'étudiant(e) :

NOM, Prénom	............GASHI Dren............
Numéro matricule	............0130732528............

L'établissement d'accueil :

NOM	Université du Luxembourg - FSTC

Responsable local de stage :

NOM, Prénom	............BOTEV Jean............

Responsable académique de stage :

NOM, Prénom	......ROTHKUGEL Steffen.......

Période :

Du	............29/03/2016............
Au	............19/04/2016............

Document à compléter et à envoyer (même non encore signé) par email au responsable académique, avec copie au responsable local, le plus tôt possible après la fin de chaque période. Une période correspond à trois semaines de stage.

Ce document complété et signé devra figurer en annexe du mémoire de fin d'études.

Travaux effectués pendant la période :

— **Implementation:** ....

– ***Client-Side***: ....

- Web-Socket JavaScript Client ....
- View (html) to join a quiz via pin and username (pin hardcoded yet)..
- First (simple) Communication for joining implemented ....

– ***Server-Side***: ....

- Web-Socket Java Endpoint ....
- Accepting clients (pin and username) ....
- Creation of a common Model (Entities) ....
- Generic Framework implementation ....
    - Database (table) creation on start-up based on Model..........
    - Admin (Prof) View for creating an instance of an Activity.......
    - Super Admin View for registering Activity-modules...............
    - Generic REST Web Service for Super Admin View ....

(*Si pertinent*) Travaux prévus pour la période suivante :

Prof-View : possibility to create instances of quizzes composed of activities, possibility to.. "flag" a quiz as a grouped one such that students can build interconnected groups for....... playing...

Server-Side: random generation of pins (4-number digit), grouping web-socket connections for student groups
....

Client-Side: possibility for students to enter group + nickname.. ....convention: groupname**:**nickname...

	Le responsable local,	L'étudiant(e),
Date :	19/04/2016	19/04/2016
Signature :	[illegible]	[illegible]

Université du Luxembourg
Faculté des Sciences, de la Technologie et de la Communication
Bachelor en Informatique (*professionnel*)

TRAVAIL DE FIN D'ETUDES

FICHE DE SUIVI DE STAGE

L'étudiant(e) :

NOM, Prénom	…………GASHI Dren…………
Numéro matricule	…………0130732528…………

L'établissement d'accueil :

NOM	Université du Luxembourg - FSTC

Responsable local de stage :

NOM, Prénom	…………BOTEV Jean…………

Responsable académique de stage :

NOM, Prénom	……ROTHKUGEL Steffen…...

Période :

Du	…………20/04/2016…………
Au	…………11/05/2016…………

Document à compléter et à envoyer (même non encore signé) par email au responsable académique, avec copie au responsable local, le plus tôt possible après la fin de chaque période. Une période correspond à trois semaines de stage.

Ce document complété et signé devra figurer en annexe du mémoire de fin d'études.

Université du Luxembourg

Faculté des Sciences, de la Technologie et de la Communication

Bachelor en Informatique (*professionnel*)

Travaux effectués pendant la période :

— **Implementation:** ....

– ***Other Projects***: ....

- SimpleQuestion, SimpleFocus, MultipleChoiceQuestion, MultipleChoiceFocus with each Client-side code and server-side code (CRUD operations, REST interface, etc.)

– ***Yactul*** : ....

– ***Client-Side***: ....

- Possibility for students to enter group name if quiz is marked as a.... grouped one ....
- Project of Mike embedded (student and teacher).........................
- Possibility for teacher to start a quiz for playing ....
- Showing High-Score in real time (screen of teacher) ....

– ***Server-Side***: ....

- Random generation of pin (n-number digit) ....
- Possibility added for teachers to create quizzes composed of .... activities...
- Abstract Command class and Command-processor created for handling students incoming messages (json format) ....
- Class and Data structure for playing a quiz ....
- Possibility added for teachers to create group of quizzes ....
- Interface created for Concrete Activity REST classes with methods which are being invoked from the Generic REST.........................
- Generic Framework extended...
  - Implemented Logic for interacting with client-side web-socket connections ....
  - Evaluation process added → delegating students answers to concrete activity-specific REST interface and sending response to students web-socket connection ....
  - Generic REST Web Service for Super Admin View ....
- Data structure for storing students scores and answers.................

Université du Luxembourg
Faculté des Sciences, de la Technologie et de la Communication
Bachelor en Informatique (*professionnel*)

(*Si pertinent*) Travaux prévus pour la période suivante :

Writing the thesis.

	Le responsable local,	L'étudiant(e),
Date :	17/05/2016	17/05/2016
Signature :	[illegible]	[illegible]

Université du Luxembourg
Faculté des Sciences, de la Technologie et de la Communication
Bachelor en Informatique (*professionnel*)

TRAVAIL DE FIN D'ETUDES

FICHE DE SUIVI DE STAGE

L'étudiant(e) :

NOM, Prénom	............GASHI Dren............
Numéro matricule	............0130732528............

L'établissement d'accueil :

NOM	Université du Luxembourg - FSTC

Responsable local de stage :

NOM, Prénom	............BOTEV Jean............

Responsable académique de stage :

NOM, Prénom	......ROTHKUGEL Steffen......

Période :

Du	............12/05/2016............
Au	............29/05/2016............

Document à compléter et à envoyer (même non encore signé) par email au responsable académique, avec copie au responsable local, le plus tôt possible après la fin de chaque période. Une période correspond à trois semaines de stage.

Ce document complété et signé devra figurer en annexe du mémoire de fin d'études.

Travaux effectués pendant la période :

Writing the thesis.

	Le responsable local,	L'étudiant(e),
Date :	30.05.2016	30.05.2016
Signature :	[illegible]	[illegible]

Bibliography

[1] S. Freeman, S. L. Eddy, M. McDonough, M. K. Smith, N. Okoroafor, H. Jordt, and M. P. Wenderoth, *Active learning increases student performance in science, engineering and mathematics.* Proceedings of the National Academy of Sciences (PNAS), 2014.

[2] S. Deterding, D. Dixon, R. Khaled, and L. Nacke, "From game design elements to gamefulness: Defining "gamification"," pp. 9–15, Proceedings of the 15th International Academic MindTrek Conference, 2011.

[3] Bunchball, *Gamification 101: An introduction to the use of game dynamics to influence behavior.* Bunchball, Inc., 2010.

[4] L. Hakulinen, T. Auvinen, and A. Korhonen, *The Effect of Achievement Badges on Students' Behavior: An Empirical Study in a University-Level Computer Science Course.* International Journal of Emerging Technologies in Learning (iJET) Int. J. Emerg. Technol. Learn, 2015.

[5] C. Kevin, "A future full of badges," The Chronicle of Higher Education, Apr 2012.

[6] A. Marczewski, "Gamification: A simple introduction," p. 3, Andrzej Marczewski, Apr 2012.

[7] "Public kahoots." `http://create.kahoot.it`. Accessed: 22/05/2016.

[8] "Quizizz linkedin." `https://www.linkedin.com/company/quizizz-inc`. Accessed: 22/05/2016.

[9] "Socrative linkedin." `https://www.linkedin.com/company/socrative`. Accessed: 22/05/2016.

[10] "Kahoot faq." `https://getkahoot.com/support/faq/`. Accessed: 22/05/2016.

[11] "Db-engines.com." `http://db-engines.com/en/ranking/relational+dbms`. Accessed: 29/05/2016.

[12] "Websockets global support overview." `http://caniuse.com/#feat=websockets`. Accessed: 23/05/2016.

[13] M. R., R. M., N. M., and S. A., "Collaborative robotic instruction: A graph teaching experience," pp. 330–342, Computers & Education, 2009.