Ubiquitous Learning Applied to Coding

A set of tools and services to deliver code-intensive learning contexts to student devices

Sachar Paulus
Mannheim University of Applied Sciences
Germany
s.paulus@hs-mannheim.de

Thomas Smits
Mannheim University of Applied Sciences
Germany
t.smits@hs-mannheim.de

Tobias Becht
Mannheim University of Applied Sciences
Germany
t.becht@hs-mannheim.de

Serife Kol
Mannheim University of Applied Sciences
Germany
s.kol@hs-mannheim.de

ABSTRACT

Today programming is a crucial skill in many disciplines, demanding for an adequate education. Unfortunately, programming education requires a dedicated set of tools (editor, compiler, ...), often forcing the students to use the pre-configured machines at their universities. In an ideal setup, students were able to work on their programming assignments anywhere and on any device. This paper presents an infrastructure and tool set for a bring your own device concept in programming education: lecturers are able to provide applications, data and configurations easily and students can install individualized setups for different lectures and programming languages on their clients with one click. Neither student nor lecturer needs detailed knowledge of the installation or configuration process.

CCS CONCEPTS

• **Human-centered computing → Ubiquitous and mobile computing systems and tools**;

KEYWORDS

Ubiquitous Learning, dynamic configuration, bring your own device, virtual environments

ACM Reference Format:
Sachar Paulus, Thomas Smits, Tobias Becht, and Serife Kol. 2018. Ubiquitous Learning Applied to Coding: A set of tools and services to deliver code-intensive learning contexts to student devices. https://doi.org/10.1145/3209087.3209104

1 INTRODUCTION

In 2014, the project "bwLehrpool" [1] was introduced at Mannheim University of Applied Sciences to simplify the setup and usage of the software environment used by lecturers in the university's computer rooms. It provides a substantial improvement of the administrative overhead (prior, each semester a software image was manually created and installed on all machines in the computer rooms), but the solution does not support the deployment of software to the students' computers. Such a deployment improves the learning experience of the students, supports blended learning approaches for programming (and other software intensive) courses and furthermore allows to reduce the number of computer rooms.

The Ministry of Science, Research and Art of the State of Baden-Württemberg funded the project "Überall Programmieren Lernen" (= ubiquitous learning applied to coding), which has the goal to provide a set of tools and services that enable the deployment of software to devices owned by students or the university and to cloud infrastructures. The project and its deliverables are presented in this paper.

The paper is structured as follows: first, the project goals are described. Subsequently, the first attempts that have been envisaged are discussed. Next, the status quo regarding existing solutions attempts is described, followed by a short description of the project results. The conclusion takes a look at potential further improvements.

2 PROJECT GOALS

The basic project idea is to provide an effortless switch between contexts, where students are enabled to use their own devices for the programming courses and lecturers are freed from IT support tasks. Programs and data can be synchronized via cloud services, providing the same set of tools and the same data on all machines. We detail the requirements in more detail in this section.

2.1 Better Learning Experience

In classical programming courses, students will be presented with concepts during lectures and will be asked to translate them into coding during lab courses in computer rooms equipped with preconfigured computers. This limits the learning experience to the lab time. What if students would be able to perform the exercises at home, without needing to set up a separate programming environment? Such an approach would allow for a much more flexible course structure - making use of learning concepts like project-based learning or inverted classroom in the programming context. A solution consists in using student owned devices both at home and in class rooms.

An increasing number of public institutions support "bring your own device" (BYOD) concepts [6], e. g. for email and web usage. Nevertheless, these concepts are often poorly integrated into the

ECSEE'18, June 14–15, 2018, Seeon/ Bavaria, Germany

ACM ISBN 978-1-4503-6383-9/18/06…$15.00
https://doi.org/10.1145/3209087.3209104

learning environment: lecturers have to spend a significant amount of time providing technical support to the students (e. g. installing software, or synchronizing files). To address this problem, one major goal of the described project is to foster the usage of student-owned devices in programming-intensive courses.

If students use their own devices, it will no longer be necessary to tie programming courses to the university's computer rooms. Storing programs and data in the cloud and synchronizing them with the student's machine ensures that the data is always up-to-date on every device – at least as long as it is connected to the learning infrastructure. Source code can be synchronized using version control tools[1] and data via services like OwnCloud or Nextcloud[2]. Doing so would enable the mentioned advanced learning concepts in programming and thus lead to a better learning experience.

2.2 Flexibility and Efficiency

Our ubiquitous learning solution shall provide a flexible learning environment not only for students, but also for universities. Since it is no longer needed to use specific computers for certain courses, the computer rooms can be used much more flexible and eventually less computer rooms are needed at all. Consequently, a programming course can then take place in any room and may even be performed remotely (but, in contrast to pure online learning offerings, they may still being accompanied by professional teachers) or in blended learning setups.

To support this approach, it is necessary that our ubiquitous learning solution can install, change or uninstall applications and services driven by external configurations on any student-owned device. Alternatively, remote terminal solutions could be used to allow students access to the programs from their devices without any local installation.

Lecturers should spend less time supporting students with technical problems. Their main focus should be on the content of a lecture rather than helping to install an integrated development environment or configuring applications. Consequently, a goal of this project is to reduce the time teachers spend on helping students with IT problems so that they will be able to spend more time on governing the actual learning process.

Therefore, one of the main goals of this project is to save time by reducing the workload of the device and application management. Students should be able to instantly switch between workspace configurations. Ideally, synchronizing data should be relatively effortless – sharing files and applications between devices. Students will be able to use the provided learning environment whenever they want to.

3 STATUS QUO

This section shortly describes existing solutions and approaches we identified at the beginning of our project and explores the limitations we observed.

[1]Preferred version control services are GitLab, GitHub or Phabricator.

[2]An open source software, which helps creating file synchronization and hosting services.

3.1 Web Applications

BYOD can easily be realized by using web applications as a main source for application delivery. Most of the devices don't require additional installation of applications or libraries.

To make use of web applications, it is important to ensure that applications required by lecturers are available through a store or an application catalog. On the one hand this solution solves dependency issues and avoids local installation; on the other hand the set of software engineering applications, which are available as a web application (e.g. Eclipse CHE [4]), are rather limited.

Most of the preferred IDE's[3] are not available as a web application (e.g. IntelliJ IDEA, Visual Studio, PyCharm, ...). Some solutions are provided as a web application via a collaboration service like Office 365 by Microsoft.

To sum it up, most IDE's and other development tools require a local installation or cannot easily be converted to a web application.

3.2 Application Servers

Terminal server / remote desktop solutions consist of a server with applications installed and a client. Every application has to be installed on the server. Additionally, this approach depends on an active connection between server and client. If a valid connection is established, the applications can easily be streamed to any device, and specifically to mobile devices.

Furthermore, a serious amount of processing power and storage space is needed to create an application server with a lot of IDE's and their dependencies. NoMachine's Linux Terminal Server works with the latest version of the NX client to connect and stream desktop applications to mobile devices or local environments [5]. This product needs to be licensed and the price range varies depending on the number of CPU cores. The same applies to the approach providing a Windows Desktop through a Windows Terminal Server solution, where a user based licensing is necessary.

An open-source alternative is the Linux Terminal Server Project [2], which provides a terminal server image and clients. This open-source project is driven by the community, however, it is not supported any more, since the news section and the sources are outdated. Furthermore, most of the modern applications do not work at all with the Linux Terminal Server Project and after all a high amount of knowledge is required to get started.

Overall, either the solution is commercial or does not work properly. Both approaches rely on a stable connection for streaming applications, which is in general the case only on site in universities. As a result, this approach cannot be considered since a stable network connection cannot be ensured when students are learning outside of the university.

3.3 Docker

Docker[4] provides a clean and isolated deployment solution for applications designed to run as services in containers. It provides a

[3]IDE is an acronym for **I**ntegrated **D**evelopment **E**nvironment with extended support for programming languages and tools to develop applications, services and/or web based solutions.

[4]Docker is a software which helps to create containerized applications, provides virtual network infrastructure and container isolation as a security concept.

virtual environment for applications and can be configured to use local resources[5].

Whereas Docker is perfectly suited to set up and run services such as web servers or databases for learning environments, it is not designed for running applications that make use of a graphical user interface of the host platform. There are some workarounds to run graphical applications in docker containers [3]. Our experiments shows that some applications still tent to freeze. This solution is not reliable and furthermore cannot be used natively on mobile devices with an Android or iOS operating system. Finally, there is still the limitation of providing support for the local installation of docker on student devices.

3.4 Virtual Machines

Virtual environments and virtual machines can be used to deploy complete sets of applications, data and configurations to lecturers and students. Those applications are bundled with an operating system. Mannheim University of Applied Sciences uses a service called bwLehrpool [1] to start virtual machines over network and to load the desired configuration for almost all computer programming courses. There is a pool of virtual machines available and those environments contain a number of applications, most of them are redundant for every virtual machine image created (e.g. Microsoft Office on Windows machines).

This approach has two drawbacks: first, this solution is not generally available on mobile devices and it cannot be accessed via remote desktop solutions, rather just as a virtual machine image that students need to install locally. Second, those virtual environments are large in size and therefore a high capacity in storage space is required to save them for mobile usage.

To provide context changes on student devices, students need to switch between virtual machines, thereby installing all required virtual machine images on their device. Moreover, a change in configuration by the lecturer might not be recognized by a student and thus he would work with an outdated version of the virtual machine.

Virtual machines may seem the way to go, but storage restrictions and software redundancy need to be solved. Another challenge is how to license software that is formally used outside of the universities infrastructure.

3.5 Remote Desktop Environment

Some classes of devices cannot execute virtual machines, such as smartphones, tablets or netbooks. This can be solved by providing applications and services using a remote desktop environment or through a terminal server. Data and files can be accessed through a file share via SMB or SSHFS (similar to the bwLehrpool approach). This method is a good fall back for "weak" devices, although it is not a good standard solution because it requires constant Internet connectivity and can not be used while traveling.

3.6 Data Synchronization

Data synchronization solutions, such as OwnCloud, Nextcloud, Dropbox, Google Drive, iCloud and OneDrive cannot be used to deploy applications, because these typically rely on local dependencies.

[5]Examples are storage, CPU cores, libraries, files and network.

3.7 Versioning

To create applications in a development environment, the system should provide tools to manage source code and versioning of files. Services like git, svn or mercurial provide those features. To ensure the quality of the infrastructure and to eliminate costs, self-hosting services like GitLab or Phabricator are great resources. Using such an infrastructure, students are capable of syncing their data after modifying code while working on an exercise. This is a major advantage compared to environments like bwLehrpool, where unsaved changes may result in a loss of data, since data is only stored during the runtime of the provided operating system.

4 APPROACH

In this section, we describe our approach finding a solution to our challenge. The concept was developed in the period Jan 2017 - May 2018 and during this phase, numerous discussions took place with representatives from different stakeholders.

4.1 Stakeholder Analysis

Initially, we identified the stakeholders for an ubiquitous learning solution for programming and subsequently, the processes used. The major stakeholders are:

(1) Lecturers
(2) Students
(3) Administrators

whereas lecturers teach programming courses to students, and administrators support lecturers in providing infrastructure and maintaining the courses (in many cases, the lecturers created the learning infrastructure themselves and were assisted by administrators in deploying the solution). We analyzed the processes and identified improvement potential where stakeholders could be better supported in their work by using specifically developed software, e.g. infrastructure elements. Notably, the following areas have been identified:

(1) Currently, lecturers offer full virtual machines to students as an option to working in computer rooms. The size of these virtual environments range between 5 and 40 GB. This is considered as disadvantage, since this in reality only works by using USB sticks. Furthermore, if the lecturer decides to change the learning environment, this results in a complete re-installation, often deleting existing already developed learning artifacts of the student.
(2) An alternative that is often used consists of letting the student install its own integrated development environment (IDE). In this case, the lecturer spends a considerable amount of time in assisting the students during the configuration of the software, often during lecture or lab course times.
(3) Finally, the construction of new virtual machines consumes a lot of time, specifically when installing new software that shall be provided to students. This has led to the effect that VMs are built once and then used for a long period of time - even if new versions of the software used would be available.

4.2 Architecture Design

A system architecture was iteratively designed in several subsequent approaches. The upper part of figure 1 shows the components that are part of the final system architecture. The lower part visualizes the architecture elements that needed to be developed.

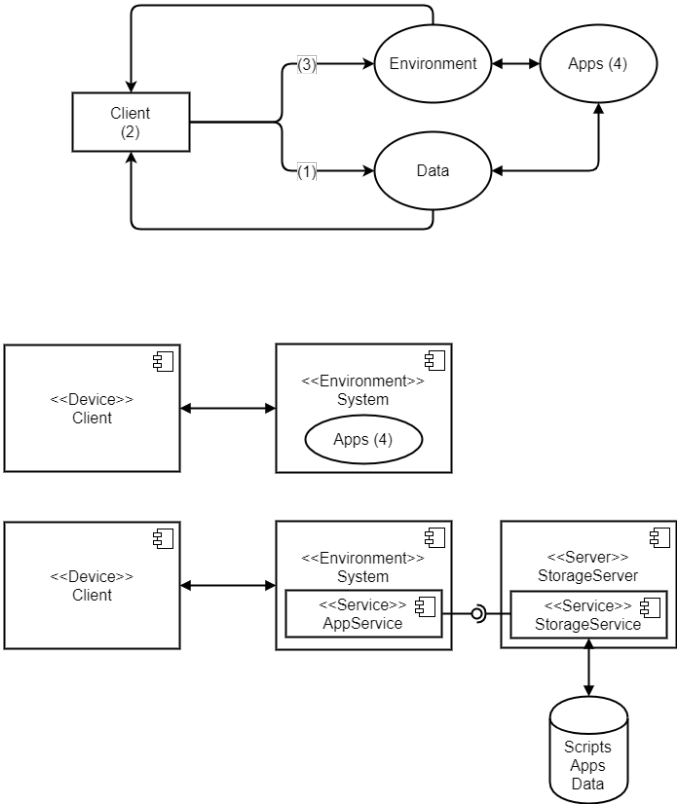

Figure 1: Component view of the system.

4.3 The Concept

The most effective way of accessing data online and offline is to use a service like OwnCloud or Nextcloud. Therefore, the project team integrated this service for data synchronization.

As the discussion in the last chapter shows, some form of virtualization is needed to provide applications to the student devices. State of the are technologies were tested such as Ubuntu Snaps, Flatpak, AppImages, Zero-Install and Docker; these solutions differ by their level of integration into the operating system. Moreover, they are not available for each student's operations system. The software packages available for these technologies are rather limited compared to a local software installation. The best support for a multitude of applications and services is therefore a virtual operating system that allows to install software locally (or remotely).

The next challenge is to identify a (virtual) operating system that fulfills the requirements of the project. Major requirements are the availability of applications, the size of the operating system image and the usability from a student perspective.

Initially, it was planned to use a small virtual environment with a size of approximately 500 MB. This virtual environment consists of a minimal desktop setup and loads applications and services as Docker containers. A number of variants are investigated, among others the linux variant Alpine OS, a system that is based on libmusl. Unfortunately, it is not compatible to many linux applications (web browsers, editors, etc.) that make use of libc. A small test among students shows that they prefer a slightly larger system if a standard level of comfort regarding the desktop environment is offered. Unfortunately, the project team could identify a fully functional desktop environment with the expected small size. Specifically, Microsoft Windows installations as guest OS with no / a reduced set of software applications were much larger than expected (20 GB and more).

After a number of investigations including usability tests with students classes, Linux Mint has been identified to be the system of choice. The size of the system is approximately 4 GB (zipped).

The next step consists of equipping the virtual machine operating system with some management software allowing it to install/change/remove lecturer software, called "applications". It was decided to use existing state-of-the-art tools with a dedicated, easy to use graphical interface. A first mock-up for the student interface is shown in figure 2.

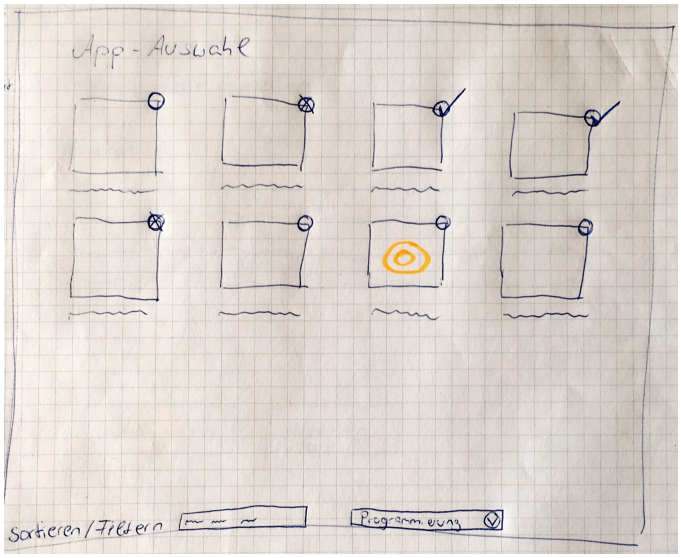

Figure 2: Early scribble of a lecturer's view on the application catalog.

The student interface sends a request to web services that provide software and the corresponding state transition scripts (e.g. install or activate) for specific learning scenarios. The student interface, the web API as well as the configuration inferface for these web services have been developed as part of this project.

5 RESULTS

At the time of writing of this paper, the solution exists as a prototype supporting different student usage scenarios (university computer room, BYOD laptop, BYOD tablet). A full documentation and

the source code is available at
https://github.com/informatik-mannheim/UEPL.

The prototype has been tested in different courses at the Computer Science Department of Mannheim University of Applied Sciences in the timeframe Sept 2017 - Feb 2018, whereas feedback from lecturers and students was collected and taken into account for subsequent versions of the prototype.

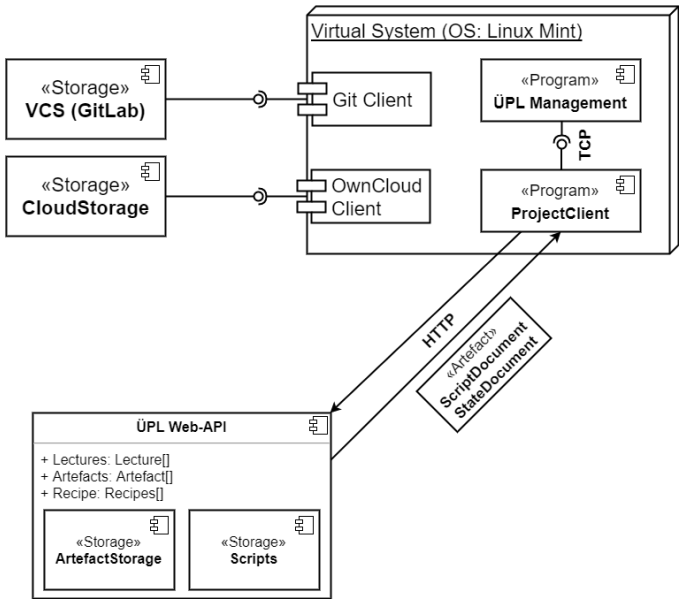

Figure 3: Deployment view of the prototype.

5.1 Server

The web API is split into a web service, which stores scripts, files and manages the relationship between applications[6], lectures and users and a configuration service which controls the content of the corresponding scripts. The following states can be changed:

- (Un-)Install an application
- (De-)Activate the context of an application

To maintain the scripts, administrators make use of the web interface of the service reachable on port 10000 on the hosting server by default. The web front end enables lecturers to create their own lecture for a specific class or to create a virtual image by first installing the context on a virtual environment and creating a snapshot of the image. This snapshot can be used to distribute a certain configuration, e.g. applications and corresponding data.

The web API runs in a docker container (or, alternatively, directly on a custom server). It supports authentication against an LDAP-server as default, but allows to use a custom authentication provider. Corresponding script files are stored at file system level; relations are stored in a SQLite db-file database. Other database providers are supported as well, such as MySQL or SQL server.

Besides these services, an OwnCloud server has been set up to provide synchronization of data between different end points (two or more instances of the virtual operating system, running on different computers (in the university computer room, on the students laptops, at home or alternatively on a Remote Desktop instance). Alternatively or in addition, a GIT server could be used.

By the date of this document, a demo version is running on bwCloud[7] infrastructure.

5.2 Client

Our virtual operating system uses Linux Mint 18 with Mate desktop environment as an operating system. Just one service is configured to run on startup and to manage the operating system resources; there is an icon on the desktop to start the student interface.

Once the VM is installed on a student computer, the students can make use of the student interface, illustrated in figure 4, to download the lecture specific contexts and to (un-)install or (de-)activate them. The service will execute the state transition scripts after checking the integrity of all downloaded programs and data using pre-shared certificates. The student interface makes use of standard packet managers (rpm) and shell scripts.

This allows the following scenario: students can start working on a programming exercise in the computer room of the university. Any work item produced is synchronized to the Cloud. If they want to continue their work (and assuming they have installed the VM on their laptop), they simply can change the lecture in the student interface, and continue to work on their exercises using their laptop. The laptop can be offline during work; online connectivity is needed during the lecture switch and at the end of the work (to synchronize work items).

Furthermore, students can access a VM prepared like described above running in the Cloud using Remote Desktop Protocol (RDP). This way, students can access the same and synchronized learning environments in the Cloud from a tablet computer. The condition is, of course, an online connection.

6 CONCLUSION

This section summarizes the results and experiences we have collected from the project as well as possible extensions of the system.

6.1 Summary

The Ubiquitous Learning approach presented in this paper enables every institution to synchronize data, deliver content and share applications with people, especially students.

With the current prototype it is possible to add lectures and assign applications that will be installed on demand as soon as a student clicks the install button for the corresponding lecture.

Application pools and other artifacts can be managed by operating system experts or administrators. They specify which system to use and ensure the scripts will run without problems.

Our provided tools and infrastructure enable the lecturers to provide dynamic environments without the need of pre-installed software or detailed knowledge of the installation and configuration process.

The software developed is public domain and therefore can be freely used or customized, or even further developed.

[6]In the software, applications are called artifacts, since applications are only one potential use case; another might be e.g. a database server.

[7]BwCloud is a custom hosting service of the state Baden-Württemberg.

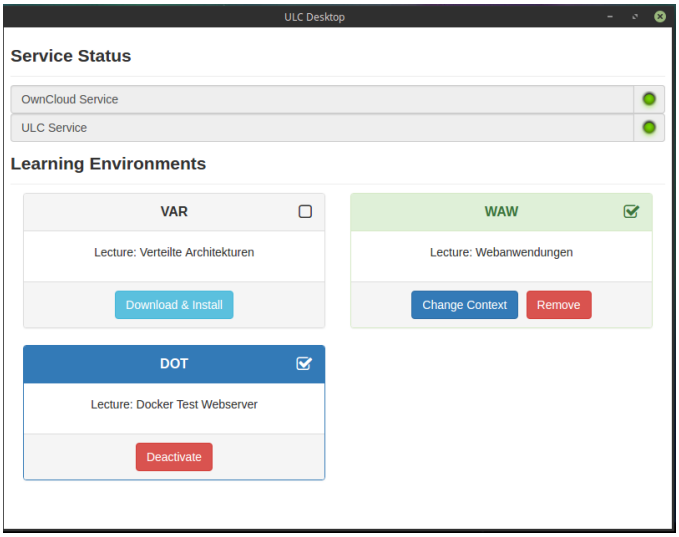

Figure 4: Generic desktop client interface with one available context (white, box unchecked), one context installed (green, box checked), one active context (blue, box checked) and context-sensitive controls to perform state transitions. In the upper area are status indicators, which show the actual status of local services.

6.2 Future Work

There are a lot of improvements to consider. The following list shows some of these topics:

- **More flexible use of artifacts**
  The artifacts section and all scripts could be parameterized to gain more control over installation locations and configuration details. This would allow to specify a docker image name or include generic artifacts to reduce the amount of artifacts. Furthermore, this would support administrators' use of generic templates and customizing the configuration.

- **Context data beyond applications**
  Uploaded data is currently stored in a fixed subfolder of the service. Metadata or additional information could be sent to the OS service, thus it would be possible to copy or move the files. This feature would help creating dynamic projects, inserting files to work folders or adding further information to an existing project.

- **OS Support**
  Currently, the only supported operating system is Linux. Support for Microsoft Windows is in beta status; tests must be run against the Windows build to ensure the same behavior as of the Linux build. Furthermore, to add support for Windows and macOS, it is needed to change some of the existing code of the service to run the state transition scripts without failure and help to diagnose unexpected behavior.

- **Context aware desktop client**
  The desktop client is an electron application with a basic authentication mechanism. Some features like assigning contexts to user profiles could be implemented on the client side; they currently exist as a server side feature only.

ACKNOWLEDGMENTS

The authors would like to thank the Ministry of Science, Research and the Arts of the State of Baden-Württemberg who funded the project "Überall Programmieren Lernen" as part of the program "Smart Teaching - Better Learning".

REFERENCES

[1] bwLehrppol Wiki. 2018. Was ist bwLehrpool? (2018). https://www.bwlehrpool.de/doku.php/allgemein/was_ist_bwlehrpool
[2] LLC DisklessWorkstations.com. [n. d.]. Linux Terminal Server Project. ([n. d.]). Retrieved March 18, 2018 from http://www.ltsp.org
[3] Fabio Rehm. 2018. Running GUI apps with Docker. (2018). http://fabiorehm.com/blog/2014/09/11/running-gui-apps-with-docker
[4] Eclipse Foundation. [n. d.]. Eclipse Che | Eclipse Next-Generation IDE, Cloud IDE, and Workspace Server. ([n. d.]). Retrieved March 19, 2018 from https://www.eclipse.org/che/
[5] NoMachine. [n. d.]. NoMachine - Terminal server. ([n. d.]). https://www.nomachine.com/product&p=NoMachine%20Terminal%20Server
[6] Christoph Pimmer, Magdalena Mateescu, and Urs Gröhbiel. 2016. Mobile and ubiquitous learning in higher education settings. A systematic review of empirical studies. *Computers in Human Behavior* 63 (2016), 490 – 501. https://doi.org/10.1016/j.chb.2016.05.057