RWTH AACHEN UNIVERSITY
Chair of Computer Science 5 - Information Systems and Database Technology
Prof. Dr. Matthias Jarke

Bachelor Thesis

Release PLEASE!

Setting Up an Application Store based on an Agile Release Methodology for Personal Learning Environments

Adam Brunnmeier, Matr.-No. 331624

March 28, 2017

Supervisors: PD Dr. Ralf Klamma
Chair of Computer Science 5

Prof. Dr. Matthias Jarke
Chair of Computer Science 5

Advisors: Petru Nicolaescu, M.Sc.
Chair of Computer Science 5

Peter de Lange, M.Sc.
Chair of Computer Science 5

Declaration

I herewith declare with my signature, that I have written this bachelor thesis

“Release PLEASE!”

on my own, that all reference or assistance received during the writing of the thesis is stated completely, and that any citation is referenced to its source truly.

_____ Aachen, March 28, 2017

Adam Brunnmeier

Contents

1. Introduction

1.1. Motivation

Besides formal education, informal learning plays a role in acquiring profession relevant skills. Informal learning summarizes learning that is e.g. done outside of institutions or courses, often without a defined learning goal. To enhance informal learning by digital means, Personal Learning Environments (PLEs) were developed. Their usage supports self-regulated learning: the user can add his own applications like collections of widgets or Web services, that store, process and present information from multiple sources and enable collaboration. The user must be free to choose any application for his PLE [HeCh10], not depending on a platform or framework.

There are some steps involved in transferring an evolved PLE to another person or to publish it so that other people can find and make use of it. To fulfill this task the relevant parts of the PLE must be identified and, if possible, be digitally preserved. This process would at least require to store associated application logic and data. In a next step, access to a running duplicate of it is necessary, either as a personal instance or used by a group of people. Simplifying the individual steps for the end user by automation would overall simplify PLE sharing.

To meet business needs, there was a shift to agile software development with constant testing and fast release cycles [HiCo01] and its positive effect in multiple use cases was shown [FeLa14]. In conclusion applying agile development for elearning apps is considered.

1.2. Thesis Goal

The aim of the thesis was to develop an agile release cycle for learning related apps, including widgets and widget-based applications for PLEs, in short PLEASE (PLE App StorE), ease their distribution and enable discovery of application settings used for learning. To achieve this aim an app store was developed that does:

- Distribute an app or deploy it as Web service.
- Provide a means of discovering and searching for learning settings. A learning setting is expressed as a set of apps with specific configuration for each app.

- Provide a means of distributing evolved ROLE space setups.
- Store content and run services used by the apps. Apps can use (e.g. http) services and external content to enrich their functionality. These services and content can also be deployed outside of the app store.
- Build an app from sources.
- Store apps and app bundles with their configuration. An app developer can locally develop an app and upload it to PLEASE for usage on any selected platform, including a ROLE space. The configuration eases the app's use with different content and service endpoints. This way the app's functionality can be used in different contexts. Furthermore the apps are versioned to avoid breakage on app updates.
- Provide explanations on the app's usage. Each app and each bundle can include a description of their intended use case, their working and their usage.
- Integrate an issue tracker and a commenting&rating feature. This way the user's burden for feedback is minimized. The feedback can be used for improving the app.

The thesis is divided into four main sections. Section 2 introduces the PLE term, describes example PLE software systems and app stores with some of their general characteristics. In Section 3 two use cases are explained and the implemented app store's requirements are described. Furthermore the implemented system's structure and design are described. The Section 4 describes the used technologies and implementation details . Section 5 tells how the achievement of the thesis goal was assessed and what the outcome was.

2. Related Work

The first Section 2.1 is about Personal Learning Environments and example PLE software systems. The second Section 2.2 summarizes research in the area of app stores, which is relatively young. Furthermore it compares existing app store implementations and their features.

2.1. Personal Learning Environments

To remind, the goal of this thesis is to enhance the development of learning related apps. Therefore it is useful to know relevant information about learning theory.

2.1.1. Definitions

A person that wants to learn something chooses tools and workflows to achieve his learning goals. It may be a notebook with pencil, sticky notes, bookmarks, (...) . However, more and more electronic devices, more specifically laptops, desktop PCs and mobile phones play a major role in learning context. In computer science research, the individual composition of software applications and their use for learning is considered as "Personal Learning Environment". PLE-term

Early work by Olivier and Liber [OlLi01] describes a "Personal Learning Environment" (PLE) as a software system that can:

- store the progress of learning in a data-file
- process and present information, even if not connected to the Internet
- communicate with other users' computers over network

A PLE can be bound to the browser (e.g. when using only ChromeOS) or a single-vendor software suite, but may also include something like an external third-party pdf-viewer or a text-editor. Olivier and Liber create the reference implementation "Colloquia" and elaborate on its communication-technology. There were many ideas from other people modeling a PLE since then, generalizations of these models can be found in [MBJ*06], in a try to create a more concise definition of the term.

The PLE is learner centric and competes with widely adopted course centric solutions, namely Virtual Learning Environment (VLE) / Learning Management System (LMS). Many educational institutions use some kind of VLE (e.g. WebCT, Moodle, L2P) that enables educators to uniformly distribute resources, manage students' sessions and records, and communicate with the students. VLEs evolved in the 90s [MBJ*06] on the basis of the Internet becoming ubiquitous. [OlLi01] mentions their observed trend in 2001 of institutions offering eLearning in terms of a software that relies on an institutional software- and data providing server. The work says that this approach leads to some problems, namely the learner's need to adapt to the user interface every time the institution is changed, somehow managing the learning progress recording across different systems and the obligation of connecting to the server to use the resources. A PLE should solve these problems. VLE

2.1.2. PLE Characteristics

The following paragraphs explain PLE's practical properties and compare it with VLE.

[SLJ*07] compares PLE with traditional VLE stating that the PLE model promotes open resources accessible by everyone every time while VLEs lock access to course visitors, often for a limited time only. Beyond the learning resources also the learner's record can be open for assessment (Open Learner Model / Open Student Model) to improve Self-Regulated Learning (SRL, see below), transparency and learner motivation [BSG*15]. open resources

Even more open is the Open Social Student Model, which enables learners to compare their learning profile with each other. [BSG*15] shows a resulted increased learner motivation in a case study. The used PLE in this thesis promotes open resources.

self-regulated learning

Another theoretical consideration of a PLE is that during its creation and use the learner is responsible for his own learning process and its self-monitoring as well as controlling the achievement of the learning objectives. [KrKl12] classifies this context as "Self-Regulated Learning". The learner must find his way of handling learning materials on the cognitive scale and planning, monitoring and regulating the learning process on a meta-cognitive scale. Self-regulatory competences can be evolved at low age and "have a high impact on successful learning during the whole life" [KrKl12]. The PLE in this thesis is designed to promote SRL. An example of SRL is when the learner wants to learn a new programming language, he may choose to either only learn the subset that is needed to pass an exam or learn it from the perspective of using the new programming language for a private or a professional context. Binding a learner to specific software promotes some learning strategies on the one hand and restrains other strategies on the other hand. This has a positive effect e.g. when providing a more efficient approach, or a negative effect e.g. when the learner has other preferences [Grah07]. This can be considered in the development of apps for a PLE.

skill evidence

A more practical benefit of PLE can be produced in the employment market. Today's education connects visited courses with acquired competences. In contrast, PLE is able to separate the institutional learning program from the task of bringing evidence for having a skill. It does so by managing and exposing an e-Portfolio of the learning process. Such an e-Portfolio can be used by employers; additionally to the evidence of knowledge there is also the possibility to monitor the application of the knowledge in a particular context [Grah07]. The used PLE provides a tool to monitor the learner's activities.

mobile

A challenging part for implementation is the integration of mobile devices. [CGAP13] focuses on the integration of a PLE on a mobile phone with the need to communicate with an institutional VLE and creates a reference implementation. PLEASE was developed with mobile devices in mind.

changing tech

In spite of PLE's advantages, according to [SLJ*07] and [CGAP13], the establishment of the course centered concept as "dominant design" and the difficulty of institutions to adopt new technology in formal contexts is a barrier for the evolution of PLE related technologies.

2.1.3. PLE and Community Theories

CoP term

A major factor influencing a PLE are interactions between learners and teachers. To evaluate and reason on them the theory of "Community of Practice" (CoP) from [WeWe15] can be applied. A CoP as group of participants is in general specified and evaluated

e.g. in terms of problem solving, requesting information, seeking experience, reusing assets, building an argument, growing confidence, discussing developments, documenting projects, visiting members, identifying knowledge gaps. Specifically to the education sector three aspects play an important role: the learning in the institution as participation in CoPs around subject matters, CoPs going beyond the institution (especially for getting insight into industry/service practices), CoPs serving for a time longer than that of the time in school [WeWe15].

The use of PLE can be an instrument to allow the evolution of CoPs around a specific subject, namely PLE has the potential to prove participation for institutional evidence, to go beyond the institutional borders and serve a life long. Furthermore an app store can serve a CoP's need of exchanging learning settings between the members.

PLE promotes CoPs

[PSB*09] tries to measure existing Web-PLE-systems and defines such systems in the six dimensions "Screen, Data, Temporal, Social, Activity, Runtime". The work maps certain software features like "list of friends" and suitable existing Web-standards to these dimensions. Other evaluation methods employ Learning Analytics: [NDKl15] summarizes trends in this field from 2011 to 2014 and lists the used theoretical frameworks and data sources. For example [PKKr15] collected data from 429k posts in an online forum, analyzed the users' interconnections and created a learning community model from that data, that was able to visualize the users' roles and intentions.

measuring

2.1.4. PLE Comparison

One possibility to deploy a learning related app is to deploy it in a PLE software system. Using a PLE system as framework can greatly ease leveraging the above mentioned advantages of a PLE. In the following the PLE and VLE software systems "L2P", "ROLE", "Moodle", "Blackboard", "elgg" are investigated and compared to get an overview of existing PLE and VLE systems. Taken into consideration are the developer of the systems and their philosophy, the target group, the origin of requirements, collaborative features, exposed information and whether the software's source is open available.

The *L2P* system is RWTH Aachen Universitiy's central LMS since 2008[1]. It is developed by the Center for Innovative Learning Technologies (CIL) of the RWTH. It was primarily designed to suit the needs of the lecturers but was redesigned in 2014 with the goal to set the focus on the students. It is a specialized system for use at the RWTH. The design includes feedback from students and teachers on using the software. The core soft requirements of the new design are personalization, mobility (think mobile phones), reflection, collaboration, extendability and usability[2]. The system offers traditional course centric features and its collaborative features are limited to an on course basis. Per course

[1] `http://www.cil.rwth-aachen.de/projekte/re-design-l2p/`, 2014
[2] `https://www.youtube.com/watch?v=f67jnqSE0Ds`, 2014

the system offers document sharing, wiki, question answer forum, creating groups. Per group it offers document sharing and chat. The software exposes course information, obligatory homework, learning material files & links, grades, announcements and surveys. The software's source is not open available.

ROLE is a framework designed to enable the user to create his own learning environment that is suitable for self regulated learning [NKR*14]. The framework is created in the context of a finished IST 7th Framework Programme of the European Commission project, comprised 16 research groups from six EU countries and China [DKK*13]. The currently last software ROLE SDK release is from February 2013[3]. It has extensive theoretical background that was collected and documented during the project-lifetime. The open source framework runs on a server and enables its users to mash up Web widgets in their browsers and share the results or to work with the widgets in a collaborative manner. The sdk's architecutre is shown in Figure 2.1. The DireWolf [KRNK13] extension allows the widgets to be spread over multiple physical devices for the same user.

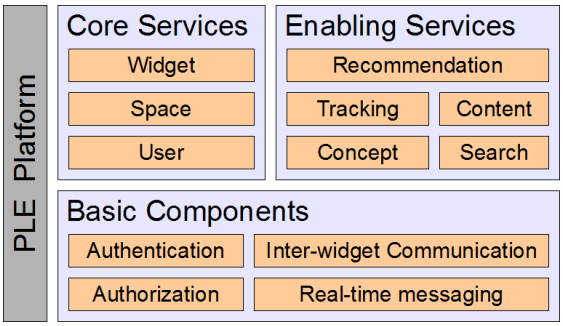


Figure 2.1.: ROLE SDK architecture [AnKi13]

Moodle is an open source learning platform first published in 2001 and now developed by Moodle Pty Ltd. Its core as well as community development team size grew over time. It is designed for use in course context by teachers. Its requirements and development are driven by user feedback (Moodle Tracker). From stock installation, only sending text messages is possible as collaborative feature between students. Further features like chat, forum, wiki or similar can be enabled in a per course context by the course administrators. The Moodle platform is extendable via plugins by core and third party developers. The plugins can only be integrated by the course administrator and are based on the course structure[4].

Blackboard Learn is a widespread closed source learning management system geared towards schools and enterprises. There is no freely accessible version available. The sys-

[3] `https://sourceforge.net/projects/role-project/files/`, 2016
[4] `https://moodle.com/hq/`, 2016

tem is course based and offers per course collaborative features like video sessions, chats, forums, (…). It can be extended by plugins[5].

elgg is an open source project not only targeted at formal educational purposes. It started in 2004 and is currently maintained by the Elgg Foundation. Its goal is to provide a framework that eases development of social aware Web pages (user management, …). The framework is extendable with plugins from the elgg plugin repository. Available plugins can be commented on. Collaborative features include file sharing, forum, sending messages, chat and wiki[6].

The systems are compared in table 2.1. One observation is that most plugin apis are not designed student centered outside of courses. Because of this extending a framework must conform to course centered design as the plugin can only be used in course context. Another observation is that the frameworks use mostly the same technology (e.g. chat, forum, survey, video conference, …), but the student's capabilities to adapt his workspace differ between the systems. In every system except for ROLE, a non-admin user is bound to the functionality an admin has provided.

[5] `http://www.blackboard.com/learning-management-system/blackboard-learn.aspx`, 2016

[6] `https://elgg.org/`, 2016

	developer	philosophy	target group	requirements sources	collaborative features	exposed information	open source
L2P	CIL, RWTH Aachen University (since 2009)	personalization, mobility, reflection, collaboration, extendability, usability	students and teachers	academic research, user feedback	document sharing, wiki, QA-forum, chat (per course)	course, homework, files, links, grades, announcements, surveys (per course)	
ROLE	16 international research groups in a IST 7th Framework Programme of the European Commission (2007-2013)	self-regulated and open	PLE developers	academic research	document sharing, chat, Webinar, programming, 3d tours (per user and per space)	… (per user and per space)	✓
Moodle	Moodle Pty Ltd (since 2001)	support teachers+learners, ensure openness, integrate courses, provide flexible toolset, enable scalability, allow contributions	course teachers	user feedback	chat, forum, Webinar, surveys, … (per course)	course, homework, files, links, grades, announcements, … (per course)	✓
Blackboard Learn	Blackboard Inc (since 1997)		course teachers and educational institutions	user feedback, marketing	chat, forum, Webinar, surveys, … (per course)	course, homework, files, … (per course)	
elgg	Elgg Foundation (since 2004)	open source social networking engine	social aware Web page developers	user feedback	file sharing, forum, sending messages, chat, groups, wiki	latest activities, most popular blogs, …	✓

Table 2.1.: PLE comparison

2.2. PLE Component Collections

A part of a PLE are its software components which a user can interact with. As the system shall serve the user's complex learning needs and enable community aware and pedagogically state of the art features, these components must be versatile. This is achieved by using a modular approach and divide the system's features into many software components (cf.[7][8][9]), called apps, modules, bricks, widgets or alike. PLEASE shall empower ROLE with a place to develop and distribute its widgets and non trivial widget bundles. It shall enable e.g. auto-updating, collaboratively developing an application from multiple places and a method of verifying the apps's quality. The concept of such a store can be found in Apple's AppStore, Google's PlayStore, portableapps.com or more abstractly GitHub, maven/npm's central repository.

[Behr12] enumerates the main benefits of an app store and lists efficiently searching for a most relevant app, automatically and reliably downloading, installing without further user interaction and on multiple platforms, updating apps in a user-friendly way, rating and commenting apps. The developer of an app can earn money per installation or with payments for content and functions during use of the app or with delivering third-party advertisements to the user.

functions

A study from 2013 [PaMa13] evaluates the feedback found in the main application stores from Apple, Google and Microsoft. It analyzed over a million reviews and classified them to different categories like "praise" or "feature request" and evaluated their quality. It concludes that the feedback is a capable source of bug reports, feature requests and documentation start points, and suggests that the feedback should be integrated in the software development process. Furthermore the study encourages simplification of review-submission to gain user-experiences, as users are not willed to take much effort to write the review, less than e.g. writing a review for a movie or hotel. It states that negative feedback mostly lacks a description of the user's experience and proposes to track the user's interaction with the application to enhance the review's value for the developer.

user feedback

2.2.1. App Store Comparison

In the following, app stores from different contexts and with different requirements are compared with regard to:

- The implementation is open source
- The store is targeted at a PLE or is a part of a PLE
- The apps are widget like and can be mashed up

[7] `https://moodle.com/hq/`,
[8] `https://elgg.org/`,
[9] `http://www.blackboard.com/learning-management-system/blackboard-learn.aspx`,

- App search possibilities
- App deployment with minimal user interaction
- Seamless updating of an app
- Platforms the apps are distributed for
- The Store's Features that are offered for an app deployment
- Per app rating and commenting functionality is possible
- Major means of explicitly recommend apps

The *L2P App Store*[10] is the integrated app store of RWTH Aachen's central LMS, L2P. The app store was added in a redesign 2014. Every student can compose his dashboard of manually selected apps, access course material and for example submit homework solutions. The current development focuses on mobility and PLE[11]. The apps are organized into the seven categories organization, information, learning materials, assessment, collaboration, other, third party. They can be additionally searched by name. The users can rate the apps and comment on them. Most apps are available as widgets and are automatically added to the personal dashboard. Other apps can link to distributions other than the Web, e.g. Windows or Linux, these are not automatically installed. New Web app versions are automatically updated. The apps can be sorted by the filters top rated, most used, new, alphabetically.

Moodle plugin directory[12] is Moodle's central repository for functionality and accessible as part of the official website. New plugins can only be installed by site admins, they can be activated/inserted per course by course admins. To install a plugin, they must be downloaded from the website and extracted to the server location. Plugins are of one of 24 base categories like antivirus, user, learning activities, caching, and many subcategories. They can be searched by name and part of description. Users can mark an app as favorite and add comments or reviews. The apps can be filtered by download-, usage-, favorite-count, recency and featured, that means preselected by a special reviewing group.

The *ROLE Widget Store* was developed together with the ROLE SDK and hosted widgets for the PLE environment. It gives users 100 coins to spend on feature requests so that widget developers can prioritize them on this basis. The original prototype with theoretical background is documented in [Dahr12]. The widget store can be used by every user. It stores OpenSocial compatible widgets, which can be added by uploading or referencing from another location. Each widget is associated with a set of metadata. A major feature is the possibility of creating bundles, which contain widgets, content, rich metadata and learning activities describing what should be done using the widgets and content. Widgets and bundles can be rated and commented on by every logged in user. Such a bundle can also be used by a teacher for a course context. The widgets are

[10] `http://www.cil.rwth-aachen.de/projekte/re-design-l2p/app-konzept/l2p-appstore/`, 2016
[11] `http://www.cil.rwth-aachen.de/projekte/re-design-l2p/`, 2014
[12] `https://moodle.org/plugins/`, 2016

categorized into Plan&Organize, Search&Recommendations, Reflect&Evaluate, Collaborate&Communicate, Explore&View, Create&Manipulate, Train&Test, based on a pedagogical model. They can be tagged by functionalities like "text editing" or "video chat", defined in an ontology, and may be limited to a learning domain. Bundles cover multiple categories, based on the widgets it contains. To publish a widget, it is uploaded, then revised by the ROLE developers for quality and licensing issues. The store also includes its own documentation. The store integrates with the ROLE Requirements Bazaar[13] for requirements engineering. Widget and bundle recommendation by a special recommender widget. The latter recommends widgets and bundles based on manually created templates.

The *Layers App Store (LAPPS)*[14] was developed at the RWTH and was developed to deliver apps and services developed under the European "Learning Layers" project. It was designed for learners in all contexts. It is open source and allows the upload of apps for different platforms like Linux, Windows, Android. It is divided in a Java backend and a HTML/CSS/Javascript frontend. Apps can be searched by title, tags and author. They can be downloaded and manually installed. Users can comment on the apps and vote with 1 to 5 stars. The store can filter the shown apps by rating and recency.

Steam[15] is a commercial game delivery platform operated by Valve™. Its target group are gamers and game developers, there are about 10M users online at a given time[16]. The games are organized in genres and can be searched by a text query. The search can be further limited by tags, single/multiplayer, offline/online, etc and the results can be sorted by relevance, release date, name, price and user rating. A user can follow other users who are recommending games, and be updated about his recommendations. Another option is to see the list of games played by the user's friends.

The *Playstore*[17] is Google's commercial software distribution platform for Android applications. It offers a chart of best rated apps and displays the least supported Android version. It is usable by smartphone users. There are more than 2 million apps uploaded[18]. Apps are grouped in 48 categories and can be searched by a query text which includes name and description. Users can rate and comment apps. Apps are automatically deployed when installing. The listed apps can be filtered by category. The Playstore also notifies the user if his device is not compatible with the app version.

The *npm Registry*[19] is the central repository for NodeJS's default package manager, operated by npm, Inc. NodeJS is a program to run server side javascript code. The registry is made for Web developers. There are about 3M node/npm developers[20]. The registry's

[13] `https://requirements-bazaar.org/`,
[14] `https://github.com/learning-layers/LAPPS`, 2016
[15] `http://store.steampowered.com/`, 2016
[16] `http://store.steampowered.com/about/`, 2016
[17] `https://play.google.com/store`, 2016
[18] `http://www.appbrain.com/stats`, 2016
[19] `https://www.npmjs.com/`, 2016
[20] `https://www.npmjs.com/npm/open-source`, 2016

frontend is open source, the backend is operated by other (also not open source) parties. A package is installed or updated by a command line command, which downloads it into the local filesystem. Packages can only be searched by name and they are displayed as description and links. There is no direct possibility to rate or comment a package. To do that the package must include a link to a Web page providing that functionality, what is mostly the case.

The *Cytoscape App Store*[21] was developed to replace a two stepped search and install process with a single step. The targeted application Cytoscape is an open source software tool for biological network visualization and analysis with a modular design. The app store started with 150 apps. Its goals were to emphasize the latest version of an app, enable discovering apps, enable developers to promote their app, simplify installation. It implements a featured apps section and hosts app competitions to promote app development and usage [LMD*13]. The apps are grouped in >50 categories and are tagged. Apps, tags and authors can be searched via a text query that considers name, description and author. The apps can be sorted by name, download count, votes and recency. They can be rated but not commented on.

Some observations:

- The app stores differ in the extent they fulfill common app store functionality. E.g. Steam and Playstore appeal to the user's explorative intents, while the npm Registry's explorative features are limited to a simple name search.

The previous *ROLE Widget Store* fulfills almost all tasks targeted in PLEASE. It includes a theoretical background based on pedagogical models. The L2P was initially VLE centric and headed in PLE direction only later on. So the L2P apps and thus the app store are constrained by some design limitations. The other compared systems have other goals so they lack functionality needed in PLEASE, e.g. an app bundling function.

[21] `http://apps.cytoscape.org/`, 2016

	open source	PLE centric	widget based	app search	automatic deployment	updating	app platform (Mobile, Desktop)	deployment feature	rating+commenting	recommendations
ROLE Widget Store	✓	✓	✓	category, functionality	✓	✓	M+D	widgets, bundles	✓	templates (pedagogical models)
L2P App Store	✓			category, name	✓	✓	M+D	widgets	✓	rating, downloads
Moodle plugin directory				category, name, description			M+D	download	✓	rating, downloads, usage, preselection
LAPPS	✓			title, tags, author			M+D	none	✓	rating, recency
Steam				genre, name, description, tag, single/multiplayer, …	✓		M+D	download	✓	following, marketing
Playstore				category, name, description	✓	✓	M	download	✓	rating, downloads, marketing, …
npm Registry	(✓)			name	✓	✓	D	download		
Cytoscape App Store				category, name, description, tag, author	✓	✓	D	download		downloads
PLEASE	✓	✓	✓	name, description, author	✓	✓	M+D	widgets, Web service, download	✓	rating

Table 2.2.: app store comparison

3. Concept and Design

PLEASE is an appstore that hosts learning related apps. It provides a frontend to discover and search for apps, give feedback on them and download or deploy them, depending on their type. It provides continuous integration tooling for app developers. PLEASE integrates with the ROLE SDK and the CAE.

3.1. Use Case

3.1.1. App User

search for app

The first use case is searching for an app in PLEASE, for example a professor wants to use an elearning app to teach his students anatomy and knows that there is such an app in PLEASE. To search for an app, the user will visit PLEASE's website. There he is confronted with PLEASE's app offer and a search field which searches through all apps. For example the anatomy professor would type "anatomy" into the search field and is served with the "Anatomy 2.0" app. The user selects an app and gets its description and deployment methods. To deploy the app as ROLE space, the user clicks on the "deploy to ROLE space" button. In the opening dialog he types the space name, e.g. "anatomy SS17". PLEASE auto creates a new ROLE space with the app's services and widgets running in a new app instance. Then the user clicks on the generated link to be redirected to the newly generated ROLE space and tests the app and/or modifies the instance's data, e.g. a professor may upload a new anatomical model. Then the user shares the link with other users (see Figure 3.1).

3.1.2. App Developer

upload app

The second use case is uploading an app to PLEASE for publication, e.g. a group of people uses the community application editor (see Section 3.9) to develop an elearning application and wants to publish the developed application. To upload the app a user makes the app's sources available to initiate the build process, or in case of CAE, may use CAE's integrated automated build and deployment. If the app is not deployed yet the user can visit the app's website representation in PLEASE. There he clicks on "deploy instance" to instruct PLEASE to create a new instance of the app and run it under a public IP address. After

the developer group made changes to the app, they want to update the app and the running instance. To do that, they again publish the app's sources and let them be built. After that a developer visits the PLEASE website, selects a running instance of the app and clicks on "update to version [new version]" (see Figure 3.2).

Figure 3.1.: Anatomy 2.0 use case

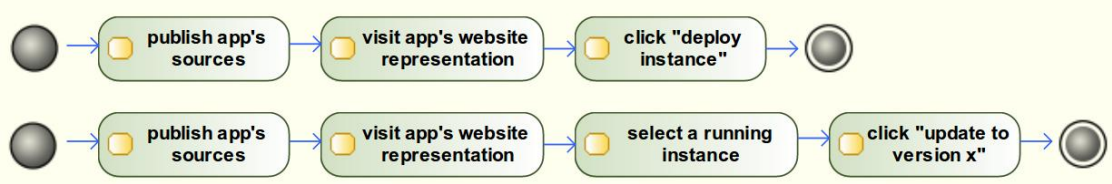


Figure 3.2.: CAE use case

To describe the type of app which can be developed and its needed configuration, some more examples are given:

A developer may choose to create an app that consists of a simple static HTML/JS/CSS webpage. To create the app, he uploads it under a public available url. Then he configures the app to download that content in the build command. In the deploy command a static http server is started. The build (=download) and deployment is started per button click. The deployment is configured to automatically update when a new download is made. — no VCS, simple upload

Another example is to create a small Windows binary. The source again needs to be available under a public url. The app's build is configured to download the source and compile it for multiple platforms. The resulting compiled binaries are configured to be available for download afterwards. — no VCS, local compiler, simple upload

The last example is a simple static ROLE widget hosted on github. The app is configured to automatically build on every github commit. The build consists of downloading the widget source. The app is configured to be deployable to a ROLE space. A deployment is configured to automatically update when a build has finished, thus incorporating every new published commit. — github

3.2. Requirements

The following two tables show the non-functional and functional requirements.

NF1	open source
NF2	target learners in every context the requirements and design is targeted to improve the learner's experience
NF3	PLE centric the store must deliver widgets to a PLE system
NF4	user feedback must be simple and time saving
NF5	app search must be efficient
NF6	high value of provided user feedback for the development process
NF7	app quality assurance
NF8	reliable/stable deployment
NF9	crossplatform PLEASE frontend running on all major OSes
NF10	running on mobile phone responsive design
NF11	compatible with collaborative app development app upload and delivering mechanism must support collaborative development
NF12	test driven development unit tests and other appropriate tests shall cover much of the codebase

F1	widgets storing and delivering Web widgets
F2	mash-up possibility uploading widget bundles to PLEASE and deploying them
F3	automatic ROLE space deployment and updating widget/bundle setup and update with minimal user interaction
F4	user feedback rating, commenting, bug reporting, feature requesting
F5	app search by name, text, category
F6	check browser compatibility warn user when browser incompatibility detected
F7	store apps and their metadata app retrievable as public uri after upload; support for descriptive media
F8	ROLE space functions ROLE space widget search, deploy and upload
F9	recommendation system recommend widgets and bundles to the user
F10	app versions support for multiple app versions

3.3. PLEASE Component Design

PLEASE is built of four modules and integrates two external parts: app metadata module, file distribution module, service runner module, frontend module, ROLE SDK integration and CAE integration (see Figure 3.3). The parts communicate via http interfaces.

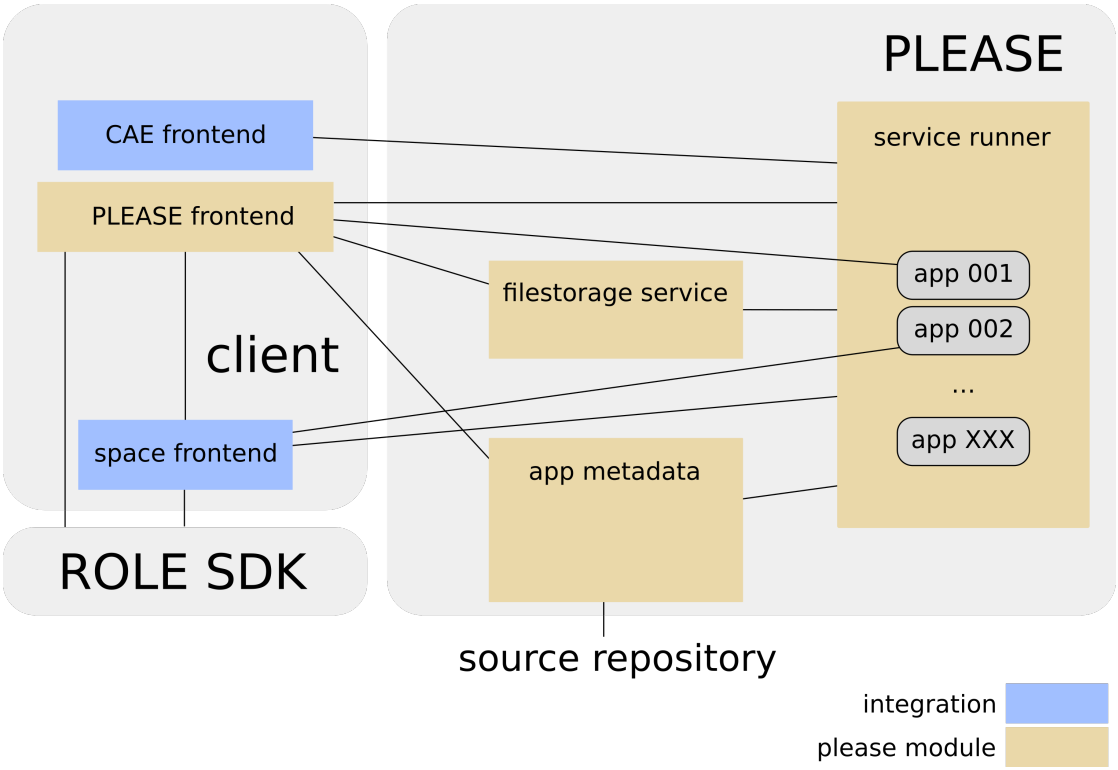


Figure 3.3.: PLEASE system overview

The app metadata service stores the apps' configuration and metadata, including description, rating and comments. It provides hooks for GitHub http hook notifications on source or version change, which trigger configured builds. Furthermore it provides build hooks which can trigger deployment updates. Some resources are user limited, the metadata service provides authorization for these resources.

The file distribution stores downloadable, versioned, per platform exports of an app's build process, e.g. a Windows executable.

The service runner executes app's builds (including testing) and their deployments. It also monitors and limits running apps, exposing resource statistics and sending build notifications.

PLEASE's frontend is an HTML5 Web page. It is based on the *Layers App Store (LAPPS)*. Users are authenticated. The frontend provides the public graphical user in-

terface to the other three modules. It enables the user to discover apps by platform or by search text, give feedback on them, create and maintain apps, manage builds and deployments as well as download app files.

Apps can be configured to be deployable to ROLE and spaces from ROLE can be uploaded as app to PLEASE. The ROLE space frontend extension includes PLEASE's frontend as iframe for widget search, deployment and upload and communicates with the service runner to update running app instances. The ROLE space frontend furthermore has the option to use an app for widget and service hosting.

The CAE frontend extension communicates with the service runner to upload and deploy apps to PLEASE.

3.4. App Metadata

3.4.1. Service State

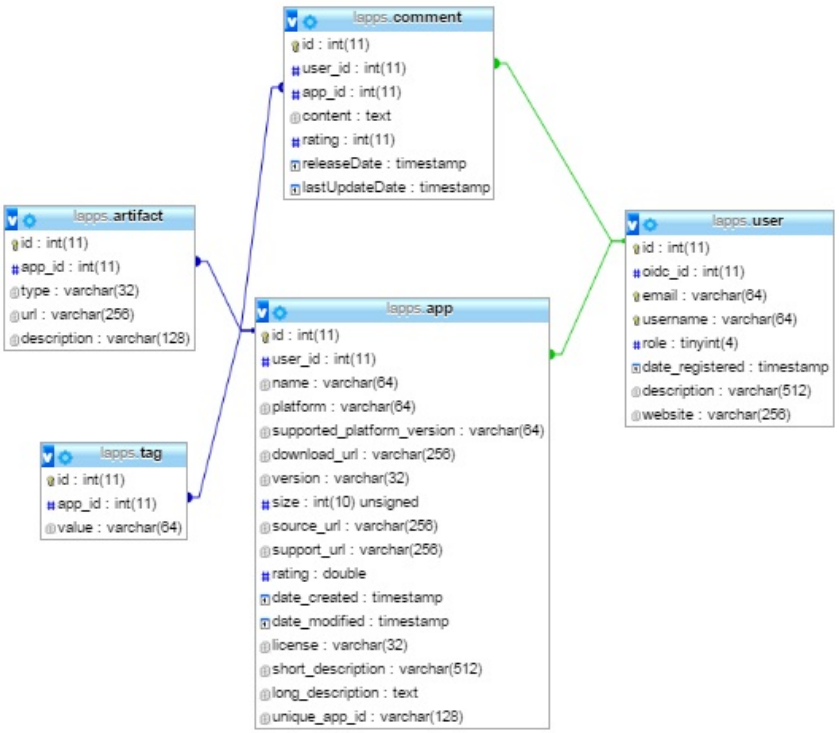


Figure 3.4.: LAPPS data scheme

LAPPS' mysql data scheme can be seen in figure 3.4[1]. It consists of *app* entities with the fields *user_id, name, platform, supported_platform_version, download_url, version, size, source_url, support_url, rating, date_created, date_modified, license*, *short_description*, *long_description unique_app_id*, app related *comment* entities with the fields *user_id, app_id, content, rating, releaseDate, lastUpdateDate*, *artifact* entities with the fields *app_id, type, url, description* and the *tag* entity with the fields *app_id, value*. Furhermore there are *user* entities with the fields *oidc_id, email, username, role, date_registered, description, website*. The data scheme represents the data state of the LAPPS service. The data that is hold by the app metadata service state is derived from that.

The service is able to correctly restart on service or system failure.

3.4.2. App Configuration

An app's configuration consists of the two fields *versions* and *autobuild*. It specifies the app's versions, its build process, its default deploy configuration and the triggers for auto-build and auto-update. An app's build result is used for a potential deployment. An app's deployment always has a version. An example of an app configuration is shown in appendix A.

The *versions* field is a JSON object. Its keys are the app's versions, e.g. "v1.2.1" or "latest". The versions are encouraged to follow semantic versioning[2] and some features only work with the semantic versioning syntax and semantic. For each version, a build can be started and for each version with at least one successfully finished build a deployment can be started (if Web service is configured in the deploy section). A version's value is a configuration object with key-value pairs described in the following. When the versions follow semantic versioning, the configuration of all preceding versions is merged into the current version. Newer versions' values override older ones. versions

The version's *env* value is an object whose key-value pairs are available as environment variables in the build and deploy scripts. Defining variables in this object can make the build and deploy scripts more readable or make them more generic if e.g. another version only differs in a url and this url is defined as environment variable, the build script can be kept the same.

The version's *build* value is an object that defines the build's base. Furthermore the *build* value defines a shell script that is executed as build. The script is responsible for downloading all needed files, compile them if needed, run any tests, log all results and, if all checks pass, prepare the files for either being directly downloadable (e.g. as a Windows executable) or being deployed as a Web service. The build can be configured to have a full

[1] `https://raw.githubusercontent.com/learning-layers/LAPPS/master/LAPPS-backend/doc/db_schema/current_lapps_data_model.jpg`,
[2] `http://semver.org`,

build script that runs the whole build and an additional incremental build script, that is ran on top of the last finished build with the same build script.

The version's *deploy* value is an object that defines how the app can be deployed as service and what files are downloadable for which platform. If the key *service* is defined, the app is deployable as Web service. The value is another object which defines the deployment's base, the command that is executed as deployment (starting some kind of server) and the files that are periodically backuped and used for a restore if the system fails or for a data rollback. If the key *space* if defined, the app is deployable as ROLE space. The value is an object that defines everything that a Web service deployment needs plus the space title, space data and the widget urls that are added to the space. The urls can use the placeholders "$ip6" and "$ip4_http" that get substituted with the deployment's addresses. Other keys than *service* and *space* are treated as platforms and their values are arrays containing the files that are made available for download.

Listing 3.1: versions example

```
{
  "v0.1": {
    "env": {
      "s1": "https://github.com/adabru/PLEASE-sample"
    },
    "build": {
      "base": "node",
      "full": "git clone $s1 && npm install && webpack",
      "inc": "git pull && webpack"
    },
    "deploy": {
      "service": {
        "base": "build",
        "command": "httpd -f",
        "backup_files": ["./*"]
      }
    }
  }
}
```

autobuild

The *autobuild* field is a JSON array with all hooks that trigger a build. An element of this array has the fields *trigger, url, change, prefixes*.

The *trigger* is either "commit" or "release". "commit" triggers fire every time a new commit is pushed to the repository while "release" triggers fire when a new tag is pushed to the source repository. A push is noticed via a notification send by a hook from github. It could also be any other provider.

The *url* specifies the repository that is checked against when a commit hook notification is received. The buildhook is only triggered if the notification originates from this repository.

change must be one of *none, commit, sync*. If it is *none*, a new build will be started without making any configuration changes. The new build will be used as the latest build for the affected versions. If the value is *commit*, the app's latest version is incremented in its prerelease identifier, also increasing the patch (or minor or major) number where necessary to follow semantic versioning. The new version is added to the app's configuration with the commit's sha added as environment variable, so that the build and deploy scripts can refer to the exact version. If *change* is set to *sync*, when a new tag is pushed to the repository, a new version with the same identifier as the git tag is added to the app's configuration, including the sha and tag name as environment variables. This option is only availabe when *trigger* is set to "release".

prefixes is an array of strings that specify which app's versions are affected by this build hook. If e.g. prefixes contains the string "v", then the versions starting with that letter (e.g. "v1.2.3", "v2.4", etc.) are affected. This way there can be e.g. a "latest" version that builds on every commit while the "v" versions only build when a version tag is pushed to a specified source repository.

Listing 3.2: autobuild example

```
[
  trigger: "commit",
  url: "https://github.com/adabru/GitTraining",
  change: "none",
  prefixes: ["latest"]
]
```

3.4.3. App Bundle

An app bundle is an app composite. It includes links to other apps' versions and optionally file snapshots. These apps can be implicit apps that are only accessible by the containing app bundle, e.g. when they are uploaded from a ROLE space. The apps inside the bundle can be updated individually. Due to time constraints this concept was not examined thoroughly enough during the thesis so that further specification about an app bundle's behaviour and interface is not done here.

3.4.4. Interface

The metadata service communicates via http. Some requests need authentication & authorization.

Retrieving the url paths "/search?q=search%20terms" and "/platform/{platform}" returns a JSON array of fitting apps. The search is implemented as a fuzzy full text search on the apps' description fields. An app's platforms are extracted from its deploy configuration which defines for which platform files are available.

The url "/apps/{id}" is used to create, update and delete apps. The creator is automatically assigned all rights for this app. Via the path "/apps/{id}/maintainers" further users can be assigned update rights. The path "/apps/{id}/comments/{timestamp}" is used to create and delete comments on the app. To upload and retrieve media that is used in the app's description, the url "/apps/{id}/media/{name}" is used. An app's rating is set- and retrievable under "/apps/{id}/rating".

The hook endpoint for build and for deployment hooks is "/hook". New deployment hooks are maintained at "/hook/{iid}", where iid specifies a unique deployment. To register a build hook, a Web hook must be added in the according github repository. The notifications send by github will then trigger builds of accordingly configured apps.

3.5. File Distribution

The file distribution service stores and serves app files, e.g. compiled binaries or script files. The files are versioned and correspond to an app's version. At the end of an app's build all files marked for export for a specific platform are uploaded to the file distribution service and this way made available for download.

The service communicates via a http interface. Files can be uploaded and retrieved by the path "/{app}/{platform}/{version}". The files are recoverable after a service restart or a system failure.

3.6. Service Runner

The service runner builds, tests apps and runs deployed apps. It provides a network interface to them, monitors them (see Figure 4.5b) and provides a facility to limit their resource consumption. As an example, las2peer nodes can be run by this service, providing an http-interface to communicate with it.

3.6.1. App Isolation

PLEASE is intended to run on a server. A server has limited resources. To reliably run app builds and deployments, they run in resource limited environments. The reasons to limit an app's build and deployment are following:

- provide a minimum amount of permanent disk storage and memory + additional amount where needed
- hinder malware (especially DOS attacks) and distribute the available network bandwidth fairly
- provide as much CPU power as possible and distribute it fairly

The service runner holds state about deployments and builds. Deployments are available from the public Internet.

3.6.2. Interface

The service runner communicates via http. App builds can be created and retrieved at the url path "/build". Deployments can be created, edited and deleted at "/deployed/{iid}". When a build finishes an http notification is sent to the appmetadata service at the path "/hook".

The containers are connected via ip6 addresses that are calculated from their iid and are taken from a configured prefix. If the host of the service runner is configured to route these addresses, the containers are globally connectable by this address. To support ip6 incompatible hosts, the service forwards all http requests made to "/deployed/{iid}/{port}" to the according container via its ip6. Ip4 connections are thus limited to http.

3.6.3. Security

Running arbitrary applications introduces security issues. The apps that will be exposing an interface to the Internet will be prone to be taken over by malicious software e.g. if the app includes a certain server version which was later discovered to have security holes. Apps themselves can be of malicious nature e.g. when the app developer turns out to have bad intentions. As the app's developer is unknown at present time, some attack scenarios are dealt with. These include maliciously acting apps binding all resources available to them and apps conducting Denial of Service (DoS) attacks. Furthermore implementation specific scenarios must be dealt with to protect the host environment and other apps.

3.7. Frontend

PLEASE's frontend provides the graphical interface to the end user. It is an HTML5 Web page to run on major platforms and its base is the *Layers App Store (LAPPS)*. The frontend provides the possibility to discover an app by terms of search queries and platform filters. Each app can be rated and commented on. Furthermore an app has a description that contains text, layout, media and interactive elements. An app's downloadable files are

accessible via the frontend. An authenticated user can be confirmed as an app's maintainer by the app's creator. Every maintainer can edit the app and start new builds. Furthermore users can manage their builds and deployment and distribute the limited resources that are available to them.

3.8. ROLE SDK Integration

ROLE is a framework designed to enable the user to create his own learning environment that is suitable for self regulated learning [NKR*14]. The framework is created in the context of a finished IST 7th Framework Programme of the European Commission project, comprised 16 research groups from six EU countries and China [DKK*13]. The currently last software ROLE SDK release is from 2014[3]. It has extensive theoretical background that was collected and documented during the project-lifetime. The open source framework runs on a server and enables its users to mash up Web widgets in their browsers and share the results or to work with the widgets in a collaborative manner. The DireWolf [KRNK13] extension allows the widgets to be spread over multiple physical devices for the same user.

To enable PLEASE integration with the ROLE SDK some changes to the SDK are needed. This includes linkage to PLEASE from the ROLE SDK, an auto app deployment from the PLEASE website, an auto in-space app update functionality and a bundle upload functionality from the SDK to PLEASE.

The space upload function creates an app in PLEASE with the space's relevant data (i.e. title, description, layout, data, tool {widget, moduleId}) and an optional file snapshot of the space associated services. When deploying an app to a ROLE space and the space already exists, the space data is merged.

The login between the ROLE sdk and PLEASE is shared. When creating a ROLE space from PLEASE, the user auto logs in to ROLE. When a user logged in to ROLE, a cookie is stored at the oidc provider's website which is used when login is done in PLEASE.

3.9. CAE Integration

The Community Application Editor [LND*16] is a Web application for collaborative model-based application development. To provide an interface to PLEASE, the CAE needs a frontend extension that communicates with the appmetadata service to upload and deploy apps to PLEASE. The functionality is provided by an extra widget.

[3] `https://github.com/rwth-acis/ROLE-SDK/releases/tag/v10.1.0`, 2016

4. Prototype

To evaluate the concept and system design of PLEASE, a prototype was developed. The prototype's sourcecode is publicly hosted on github[1][2][3][4]. It implements the appmetadata service, the service runner, frontend and filestorage service. It runs on a VM with 25GB disk storage, 8GB memory, a global ip address and an Intel Xeon CPU. Docker is installed on the system and configured with the lvm storage driver. The prototype's frontend is available at http://cloud40.dbis.rwth-aachen.de.

The app metadata service, the service runner and the filestorage service are implemented as a las2peer service. Las2peer[5] is developed at the same chair of computer science 5, RWTH university, as this thesis. For their data state, they use an in-process h2 sql database[6] and implement database backup and restore functionality. The services' http functionality is implemented using the jax-rs 2.0 api[7] backed by jersey[8] v2. The services are supported by JUnit tests that validate their conformity to PLEASE's api specification. They are built with apache ant[9].

The prototype's frontend is implemented using the polymer js framework v1[10] which provides tools for developing html+js modules that can be composed using shadow dom for component isolation and data binding. It is served using polymer's build-in http server. User authentication is provided by LearningLayers Open ID Connect provider[11]. Communication with the backend services runs over ajax requests using json as payload.

Table 4.1 shows which parts of PLEASE are implemented by the prototype and which parts are missing.

[1] `https://github.com/rwth-acis/PLEASE-Frontend`,
[2] `https://github.com/rwth-acis/PLEASE-Appmetadata-Service`,
[3] `https://github.com/rwth-acis/PLEASE-Deploy-Service`,
[4] `https://github.com/rwth-acis/las2peer-FileStorage-Service/tree/PLEASE`,
[5] `http://www.dbis.rwth-aachen.de/cms/theses/las2peer`,
[6] `http://h2database.com`,
[7] `https://jax-rs-spec.java.net/`,
[8] `https://jersey.java.net/`,
[9] `https://ant.apache.org/`,
[10] `https://www.polymer-project.org/1.0/docs/`,
[11] `https://api.learning-layers.eu/o/oauth2`,

appmetadata service			
app entity	✓	app bundle	✓
comment	✓	rating	✓
service backup	✗ partly implemented	app search	✓
github hooks	✓ authentication missing	user auth	✓
service runner			
app build	✓	app deployment	✓
app update	✓	app rollback	✓
ip6	✓	ip4 http forwarding	✓/✗ only GET requests
autoremove containers	✗	service backup	✗ partly implemented
container files backup	✗	container files copying	✗
cpu limit	✓	memory limit	✓
disk limit	✓	bandwidth limit	✓/✗ ip6 only
collect resource usage	✗	limited resources per user	✓
incremental app build	✗		
filestorage service			
file downloads	✓	file uploads	✓
auth	✗	service backup	✗ partly implemented
frontend			
responsive	✓	app creation	✓
app edit	✓/✗ only creator	app deletion	✓
app search	✓	app description	✓/✗ many description fields missing
file download	✗	service deployment	✓/✗ hooks missing
deploy ROLE space	✓/✗ bundles, hooks missing	build view	✓/✗ statistics missing
deploy view	✓/✗ statistics missing, update, rollback, delete missing	user auth	✓
upload media files	✗		
ROLE integration			
create space, PLEASE	✓/✗ bundles missing	upload to PLEASE	✗
create space, ROLE	✗	update widget	✗
CAE integration			
upload to PLEASE	✗	update application	✗

Table 4.1.: features implemented by the prototype

4.1. App Metadata

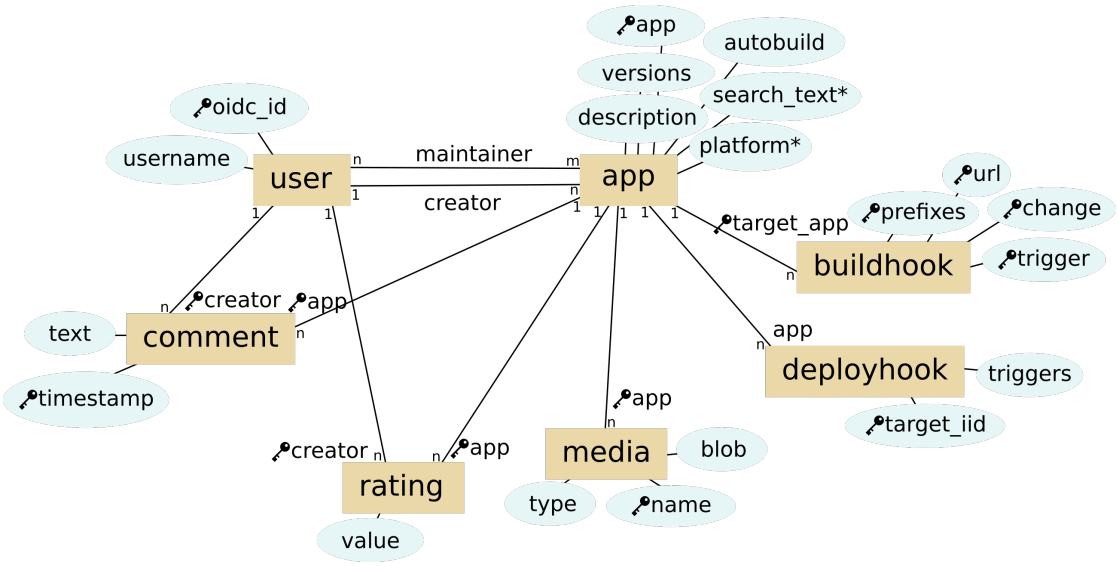


Figure 4.1.: app metadata data scheme (* autogenerated from other fields)

The app metadata's data scheme (shown in figure 4.1) is derived from LAPPS' scheme. The database is periodically backed up to enable a correct service restart on service or system failure.

It strips the *user* entity's fields *email, role, date_registered, description, website*. Some of this information about the user is available from the OIDC provider.

Furthermore the *app* entity will has the fields *description, search_text, platform, versions, autobuild*. The previous *size* entity is dynamically inferred from the downloadable file. *date_created, date_modified* are leaved out for simplicity. The previous attributes *name, platform, supported_platform_version, download_url, version, source_url, support_url, license, short_description, long_description* as well as the *tag* information are inferred from the derived attributes *versions* and *description*. *rating* becomes a separate entity with the additional field *creator*. The *artifact* entity becomes *media* and stores images, videos and other large files containing the fields *name, type, blob*.

There is the *maintainer* relationship that authorizes certain users to edit an app. The *buildhook* entities are autogenerated from an apps *autobuild* attribute. The *deployhook* entities are bound to a deployment.

Not shown in the scheme is the *githubwebhooksecret* entity with the fields *repo, secret*, which stores the secret associated with a github repo's notification hook and is used to verify that the notifications really come from github.

4.2. File Distribution

The service's data scheme is shown in figure 4.2 and only contains the *file* entity with the attributes *app, version, platform, name, size*. The files are stored locally in the filesystem and are periodically backed up together with the database to enable a service restart after a system failure.

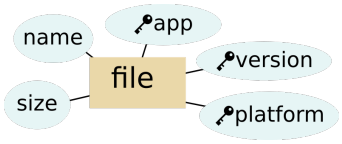


Figure 4.2.: filedistribution data scheme

4.3. Service Runner

4.3.1. Container

PLEASE makes use of linux kernel's cgroup, namespace and traffic control features to isolate app's builds and deployments from each other and from the host (more details in Section 4.3.2). As higher level api docker[12] is used. The baseset of images to build a container from, is provided by Docker Hub[13]. A container is characterized by its private disk storage, memory, cpu share and network adapter. This allows for many builds and deployments running at the same time on the same host without interferring significantly with each and without any significant performance loss. Furthermore resource consumption can be controlled per container so that user policies can be enforced on the resource usage.

A new container is created in following cases:

- a build is started
- a deployment is created
- a deployment is updated

Docker automatically assigns every container a unique container id which is stored by the service and used to control the containers. Additionally every container started with the service runner acquires a unique 32bit interface id (iid). The iid is used to communicate with a container. Build containers loose their iid when they finish and are completely removed when newer builds have finished for the same app version they built. A deployment has a permanent iid and when a deployment is updated, a new container is started

[12] `https://www.docker.com/`,
[13] `https://hub.docker.com/`,

that inherits the iid from its predecessor container. The predecessor is not immediately removed but it is paused and kept a while to enable rollback to it if the update fails for some reason.

Every container is started with the parameters *app, version, base, command, env, limit*. The *base* image a snapshot of a file system. Running a container from an image provides the container with preinstalled binaries. *command* is a shell script that is executed inside the container. For a build the script may download and compile some files, a deployment container's command may start a server. *app* and *version* are assigned to the container so that it can be correctly referenced. *env* is a set that defines environment variables that are set when *command* is executed inside the container. These may include secrets visible only to an app's maintainers. *limit* defines the resource limits for the container. A build container can alternatively be started with an *inc* shell script that will reuse the files of a previous build container to save build time.

When a build finishes, an http notification is sent to the appmetadata service with information about the build's running time and exitcode. While a container is running, its resource usage is perodically measured and stored for assessment. In deployment containers, files that were so configured are periodically backed up to an external storage.

The *base* in an app's build and deploy configuration corresponds to the container's base image (defaults to "alpine"). The images are pulled from Docker Hub[14] so that a custom build environment can be uploaded to the Hub if wished by the developer. Usually it is sufficient to start from an image like "ubuntu" or the more lightweight "alpine" and install all needed dependencies during the build.

4.3.2. Resource Monitoring and Limiting

To achieve the goals described in section 3.6.1 with limited resources, a container has a maximum memory limit and a cpu share prvoided by a linux kernel cgroup[15]. The cpu share works with timeslots that are distributed amongst all competing containers. The less containers run, the more access a single container has to the cpu. The network traffic is scheduled by the stochastic fair queueing algorithm[16] of linux traffic control which allows a container to use the full bandwidth when accessing the network alone and fair distribution between competing containers. Each container is assigned an lvm disk partition from an lvm storage pool. The partition has a fixed size. The pool is thin-provisioned to allow more usable disk space per real storage.

The standard output of the shell script executed in the container can be captured with a directive in the script. The following listing shows an example script that logs its output

[14] `https://hub.docker.com/`,
[15] `https://www.kernel.org/doc/Documentation/cgroup-v1/`,
[16] `http://tldp.org/HOWTO/Traffic-Control-HOWTO/classless-qdiscs.html`,

to a listening udp server. Handling the logging from inside the container ensures that the logging process obeys the container's resource limits.

```
{ command && ... 2>&1; pkill nc; } | nc -u example.com 9999
```

Cpu time is measured by the kernel and written to "/sys/fs/cgroup/cpu/docker/$cid/cpuacct.usage". Specifying a cpu-share to limit cpu usage, a parameter can be given to the docker command:

```
docker run -d --cpu-shares 512 ubuntu bash
```

Memory usage is also logged by the kernel and written to "/sys/fs/cgroup/memory/-docker/$cid/memory.usage_in_bytes". This resourece can also be limited with a docker command argument.

```
docker run -d --memory 100m ubuntu bash
```

Network traffic can be monitored using the pcap library[17]. It is limited with a traffic control stochastic fair queueing discipline, that distributes bandwidth equally between different ip sources.

```
tc qdisc del dev ens3 root
tc qdisc add dev ens3 root handle 10: sfq
tc filter add dev ens3 parent 10: handle 1 \
  protocol ipv6 prio 3 flow hash keys src divisor 1024
```

The disk usage can be seen by examing the container's lvm partition. It's limit is set with another docker argument.

```
# retrieve
devname=$(docker inspect \
  --format='{{.GraphDriver.Data.DeviceName}}' $cid)
sudo df -h --output=avail "/dev/mapper/$devname"

# limit
docker run --storage-opt size=200M ubuntu bash
```

4.3.3. Data Scheme

The service runner holds its state in a database (see Figure 4.3). Each build maps to a *build_container* with *app, version, cid, build_id, exitcode, imageid, runtime*. The *build_id* is the timestamp of when the container was started. Every *deployment* of an app has the fields

[17] https://linux.die.net/man/3/pcap,

iid, app, creator, cid, memory, disk, cpu. The attribute *iid* is unique, the public available interface for the container is derived from it. *memory, disk, cpu* specify the deploment's resource limits. A deployment has one active *deployment_container* which is referenced by the *cid* field. When a deployment is updated or rolled back, a new container for the desired app version is started (if it does not exist yet), all so configured files are copied to the new container, the network interface is transferred and at last the previous deployment container is paused and made available to undo the update/rollback.

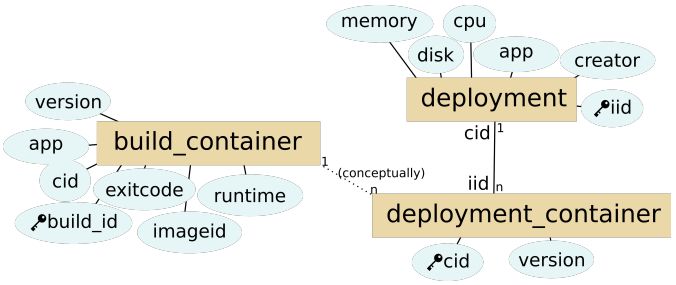


Figure 4.3.: service runner data scheme

4.3.4. Security

Maliciously acting apps can easily bind all resources available to them. Furthermore they can conduct Denial of Service (DoS) attacks as well as exploit linux kernel bugs to compromise the host. To deal with that, PLEASE runs in a VM to prevent kernel bug exploitation of its host. The VM is limited in terms of CPU usage, network traffic, memory and disk usage. Still a malicious app can compromise other apps in PLEASE by kernel bug exploitation (inside the VM). Furthermore it can take away available VM resources from other apps, which is dealt with by fair app-distributed CPU shares, limited disk and memory usage and SFQ distributed network load. DoS attacks will not be impossible, but limited by the VM surrounding firewall. If higher than average network traffic is measured over a certain period of time, the app maintainer is notified with the possibility to restart or stop the app.

4.4. Frontend

To discover an app, the frontend provides a search page. To search for an app, the user can enter query terms into a text field. Editing the text field triggers a full text search in all apps' descriptions and displays the search result as a list (see Figure 4.4). The list can be filtered by the platform the apps are available for.

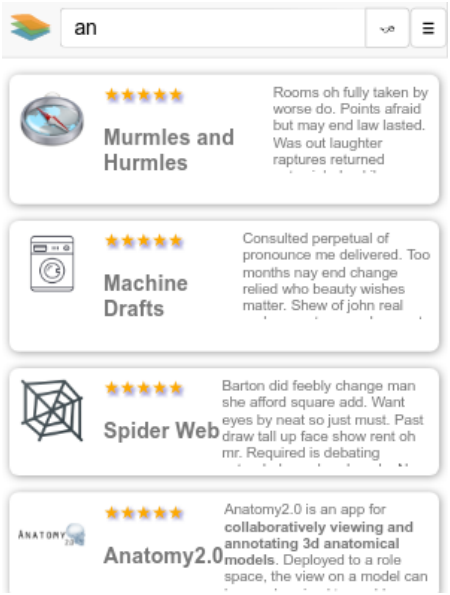


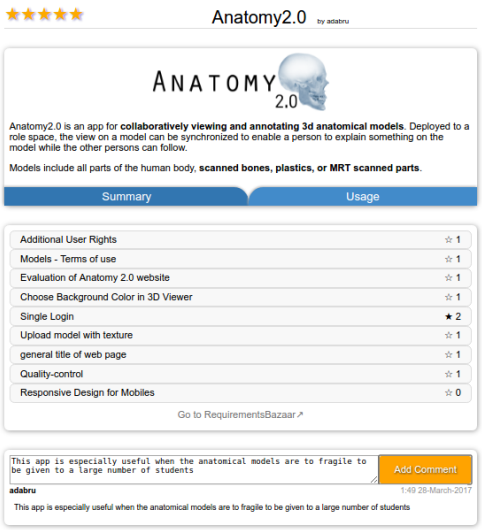
Figure 4.4.: app search frontend

To create and edit an app, the frontend povides a textarea and preview for the description and a json-editor for the configuration. Furthermore each app has a representation that shows its description, rating, comments, downloadable files and deployments (see Figure 4.5a). The app's description is written as markup text for easy editing and extendability. The description may contain the app's title, formatted text, tab views, urls, embedded media, embedded requirements bazaar[18] feature requests, source and support urls, tags, license, external documentation and a thumbnail image. A schematic example is shown in the following listing.

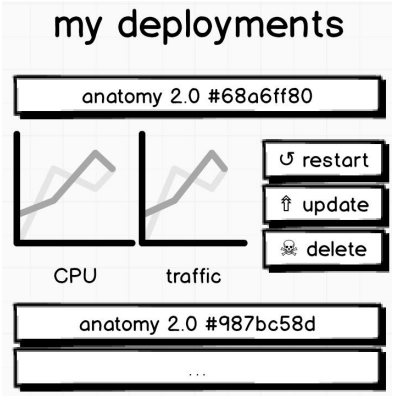
```
# The beautiful app
@source http://github/myapp
@tags learn calender beauty

## usage
come discover me!
...
```

Starting a build and viewing the status of a build can be seen in the app's build view. Starting, restarting, viewing, updating, rolling back, backing up and adding hooks to a deployment can be done via either the app's or the user's deployments view (see Figrue 4.5b).

[18] https://requirements-bazaar.org/,



(a) app details



(b) user app deployments mockup

User authentication is done with Open ID Connect[19] implicit flow. A user needs an account at an oidc provider (e.g. LearningLayers or Google) and logs in with this account. He needs to authorize PLEASE's frontend to use his information which will grant the frontend access to the user's information during a period of time after which the user needs to login again for security reasons.

5. User Evaluation

The purpose of the first part of the user evaluation is to determine if and how much PLEASE simplifies and accelerates the deployment of a widget setup in a ROLE space. The user evaluation is derived from the first use case described in Section 3.1. Therefore a group of people was asked to both conduct the use case with the currently existing/the baseline procedure and with PLEASE, i.e. starting from an open browser and ending with a ready to use ROLE space. They were asked for their user experience.

The baseline procedure is as follows:

- open an ssh terminal and login to the test server
- execute an sql script that fills the (prepared) database with the tables needed for the Anatomy2.0 service

[19] `http://openid.net/connect/`,

- download the service's source code
- change the service's configuration to match the server address and the database
- start a php server
- go to the (prepared) ROLE SDK instance, login
- create a new space and add two widgets served by the previously started php server

For the baseline procedure, every step was described in detail in the instructions given to the participating user. The PLEASE procedure is as follows:

- go to PLEASE's frontend
- search for the Anatomy2.0 app with the integrated app search
- open the deployments view and deploy the service to a ROLE space

For the PLEASE procedure, the only instruction was "Open the PLE App STore at cloud40.dbis.rwth-aachen.de. You want to learn and teach something about the human body, so create a role space with an appropriate app". There were four hints available:

- Hint 1: enter anatomy
- Hint 2: click on Deployments
- Hint 3: select version v0
- Hint 4: enter a random name in the textfield and click on deploy role space

After each procedure the user was asked some questions. The questions for the baseline procedure were as follows:

- How difficult is the setup of Anatomy2.0?
- How reliable do you think is your deployment?
- What do you think of Anatomy2.0's quality without actually using it?

The questions after the PLEASE procedure were as follows:

- How many hints did you need?
- How intuitive was the discovery of the wished app via the search?
- How difficult is the setup of Anatomy2.0?
- How reliable do you think is your deployment?
- What do you think of Anatomy2.0's quality without actually using it?

After completing both procedures and answering their questions, there were some more questions:

- What feature in the app search would result in a better search experience?
- Would you use PLEASE to search for and to setup collaborative learning apps?
- What are reasons to not use PLEASE for the aforementioned task?
- Did you already develop a Web service before?
- If yes, what advantages do you see in using PLEASE to continuously test, build and deploy your service?

- If yes, what disadvantages do you see in using PLEASE to continuously test, build and deploy your service?

To differentiate the effect that working through one procedure will affect the results of the second procedure there were two versions of the evaluation. Version A started with the baseline procedure followed by the PLEASE procedure while version B did it vice versa.

The evaluation was led with five participants.

5.1. Results

The results can be verified at [1] and [2]. Four of the five participants did already setup a Web service on a server before, one did not. The same applies for the number of participants that already developed a Web service.

PLEASE's goal of easing a service's deployment succeded in the case of Anatomy2.0 during the evaluation. While Anamoty2.0's deployment is perceived as mostly *easy* B.5b its deployment is perceived mostly as *difficult* B.5a when using the standard procedure. There is also an improvement in the user's feeling about the deployment B.6b.

PLEASE's goal to ensure an app's quality by means of rating and commenting could not be supported by the evaluation. The user's perceived quality of the Anatomy2.0 app was the same for the baseline procedure and for the PLEASE procedure, although there was a high rating, a description and positive comments given for Anatomy2.0 B.7b.

The app search was seen by every participant as rather *intuitive* B.8b. To improve the search the participants suggested to improve the design and to show more information per app.

The acceptance of using PLEASE for setup and usage of collaborative apps was between neutral and positive B.8c.

In the context of using PLEASE for continuously build, test and deploy a service the participants mentioned a simpler, faster and more flexible workflow on the positive side. On the negative side they mentioned the need to create a test and a build script, and that the build on a server can increase the effort to set up the build process. Only answers are considered of participants that positively answered the question of already having developed a Web service before.

[1] `https://las2peer.dbis.rwth-aachen.de:9098/mobsos-surveys/surveys/58/responses`,
[2] `https://las2peer.dbis.rwth-aachen.de:9098/mobsos-surveys/surveys/59/responses`,

6. Conclusion and Future Work

PLEASE's goal is to contribute to a better PLE experience by providing means of discovering apps and to promote their creation by providing an integrated build & test & deploy service that lets the developer decide which technology he wants to use. The developed prototype only provides basic, demonstrative functionality that is not able to provide a full environment for continuous development, as update/rollback possibility and webhook registration is missing in its frontend. The evaluation suggests that users are accepting PLEASE's provided way of developing and deploying apps. Furthermore the evaluation showed that at least a service's deployment can be simplified with PLEASE.

In the context of CAE integration, little can be said as neither the prototype nor the evaluation provided any practice derived information. This aspect still needs to be evaluated. The same applies to ROLE SDK integration, as neither the app bundle feature nor PLEASE's integration in ROLE were implemented in the prototype.

Regarding PLEASE's concept it can be said that the API parts may change that affect features that are not yet implemented in the prototype. When trying to implement them, new parts may be discovered that must be included to provide the wished functionality.

Regarding app bundles, they can be left out at first. After observing the usage of PLEASE without them a practice derived concept can be found for them.

Implementing more parts of ROLE's concept into the ROLE SDK and integrate them with PLEASE, namely the integration of pedagogical models to recommend apps, can offer a unique feature in the PLE landscape.

Offering a way to experiment with app configurations, e.g. by sshing into the build container or providing an interactive Web GUI to test the build and deploy script, can ease the creation of an app's configuration.

Bibliography

[Behr12] Thomas Behrens. How does an app store / market work? In Georg Carle and Corinna Schmitt, editors, *Proceedings zu den Seminaren Future Internet (FI), Innovative Internettechnologien und Mobilkommunikation (IITM), und Aerospace Networks (AN)*, volume 2012,08,1 of *Network architectures and services*, München, 2012. Chair for Network Architectures and Services, TUM.

[BSG*15] Peter Brusilovsky, Sibel Somyürek, Julio Guerra, Roya Hosseini, and Vladimir Zadorozhny. The value of social: Comparing open student modeling and open social student modeling. In Francesco Ricci, Kalina Bontcheva, Owen Conlan, and Séamus Lawless, editors, *User Modeling, Adaptation and Personalization*, volume 9146 of *Lecture Notes in Computer Science*, pages 44–55. Springer International Publishing, Cham, 2015.

[CGAP13] Miguel Ángel Conde González, Francisco José García Peñalvo, Marc Alier, and Jordi Piguillem. The implementation, deployment and evaluation of a mobile personal learning environment. *Journal of Universal Computer Science*, 19(7):854–872, 2013.

[Dahr12] Daniel Dahrendorf. Service syndication platform. 2012.

[DKK*13] Michael Derntl, Ralf Klamma, István Koren, Milos Kravcik, Petru Nicolaescu, Dominik Renzel, Kiarii Ngua, Jukka Purma, Graham Attwell, Owen Gray, Tobias Ley, Vladimir Tomberg, Christina Henry, Chris Whitehead, Dieter Theiler, Christoph Trattner, Ronald K. Maier, Markus Manhart, Maria Schett, and Stefan Thalmann. Initial architecture for fast small-scale deployment: D6.1, 2013.

[FeLa14] Natasha N. Vitó Ferreira and Josef J. Langerman. Proving that the release management process can enhance throughput in software development projects. In *Computer Science Education (ICCSE), 2014 9th International Conference on*, pages 419–424, 2014.

[Grah07] Attwell Graham. Personal learning environments–the future of elearning? *eLearning Papers*, 2(1), 2007.

[HiCo01] Jim Highsmith and Alistair Cockburn. Agile software development: The business of innovation. *Computer*, 34(9):120–127, 2001.

[HeCh10] France Henri and Bernadette Charlier. Personal learning environment: A concept, an application, or a self-designed instrument? In *Information Technology Based Higher Education and Training (ITHET), 2010 9th International Conference on*, pages 44–51, 2010.

[AnKi13] Andreas Kiefel. Revision of the role theoretical framework. 2013.

[KrKl12] Milos Kravcik and Ralf Klamma. Supporting self-regulation by personal learning environments. In Ignacio Aedo, Rosa Maria Bottina, Niang-Shin Chen, Carlo Giovanella, Kinshuk, and Demetrios G. Sampson, editors, *Proceedings of the 12th IEEE International Conference on Advanced Learning Technologies (ICALT), Rome, 4-6 July, 2012*, pages 710–711, 2012.

[LMD*13] Samad Lotia, Jason Montojo, Yue Dong, Gary D. Bader, and Alexander R. Pico. Cytoscape app store. *Bioinformatics (Oxford, England)*, 29(10):1350–1351, 2013.

[LND*16] Peter de Lange, Petru Nicolaescu, Michael Derntl, Matthias Jarke, and Ralf Klamma. Community application editor: Collaborative near real-time modeling and composition of microservice-based web applications. In *Modellierung 2016 Workshopband*, pages 123–127. Karlsruhe and Germany, 2016.

[MBJ*06] Colin D. Milligan, Phillip Beauvoir, Mark W. Johnson, Paul Sharples, Scott Wilson, and Oleg Liber. Developing a reference model to describe the personal learning environment. In *Innovative Approaches for Learning and Knowledge Sharing*, pages 506–511. Springer, 2006.

[NDKl15] Nicolae Nistor, Michael Derntl, and Ralf Klamma. Learning analytics: Trends and issues of the empirical research of the years 2011-2014. In Gráinne Conole, Tomaž Klobučar, Christoph Rensing, Johannes Konert, and Élise Lavoué, editors, *Design for Teaching and Learning in a Networked World*, volume 9307 of *Lecture Notes in Computer Science*, pages 453–459, Cham, 2015. Springer International Publishing.

[NKR*14] Alexander Nussbaumer, Milos Kravcik, Dominik Renzel, Ralf Klamma, Marcel Berthold, and Dietrich Albert. A framework for facilitating self-regulation in responsive open learning environments. *CoRR*, abs/1407.5891, 2014.

[OlLi01] Bill Olivier and Oleg Liber. Lifelong learning: The need for portable personal learning environments and supporting interoperability standards, 2001.

[PKKr15] Zinayida Petrushyna, Ralf Klamma, and Milos Kravcik. On modeling learning communities. In Gráinne Conole, Tomaž Klobučar, Christoph Rensing, Johannes Konert, and Élise Lavoué, editors, *Design for Teaching and Learning in a Networked World*, volume 9307 of *Lecture Notes in Computer Science*, pages 254–267, Cham, 2015. Springer International Publishing.

[PaMa13] Dennis Pagano and Walid Maalej. User feedback in the appstore: An empirical study. In *21st IEEE International Requirements Engineering Conference: RE 2013*, pages 125–134, 2013.

[PSB*09] Matthias Palmér, Stéphane Sire, Evgeny Bogdanov, Denis Gillet, and Fridolin Wild. Mapping web personal learning environments. In *Proceedings of the 2nd Workshop on Mash-Up Personal Learning Environments (MUPPLE'09)*, 2009.

[SLJ*07] Wilson Scott, Oleg Liber, Mark W. Johnson, Phillip Beauvoir, Mike Sharples, and Colin D. Milligan. Personal learning environments: Challenging the dominant design of educational systems. *Journal of e-Learning and Knowledge Society*, 2, 2007.

[WeWe15] Etienne Wenger-Trayner and Beverly Wenger-Trayner. Introduction to communities of practice, 2015.

A. App Example

Anatomy2.0[1] is a Web service that offers 3dmodel uploads, 3d annotations and collaborative viewing of those in a Web viewer. Courses can be created where selected models are shown and a lecturer can project his view on the students' screens. This service can be added to PLEASE with following configuration:

Listing A.1: Anatomy 2.0 as app

```
{
  "description": "
      # Anatomy2.0

      ## Summary

      Anatomy2.0 is an app for collaboratively viewing and annotating 3d anatomical
      models. Deployed to a role space, the view on a model can be synchronized to
      enable a person to explain something on the model while the other persons can
      follow.

      Models include all parts of the human body, scanned bones, plastics, or
      MRT scanned parts.",
  "versions": {
    "v0.0.1": {
      "build": {
        "base": "alpine",
        "full": "
          {
            apk update
            apk add mysql mysql-client openssl inotify-tools
            chown mysql:root /var/lib/mysql/
            mkdir /run/mysqld
            chown mysql:root /run/mysqld/
            su - mysql -s /bin/sh -c mysql_install_db
            mysqld -u mysql &
            while [ ! -e /run/mysqld/mysqld.sock ]; do
              inotifywait -t 1 -e create /run/mysqld/
            done
            mysql -e \"
              CREATE DATABASE 3dnrt;
              CREATE USER '3dnrt'@'localhost' IDENTIFIED BY 'abcdef';
              GRANT ALL
                ON 3dnrt.* TO '3dnrt'@'localhost';
            \"
            wget -q -O- 'https://raw.githubusercontent.com/'\
```

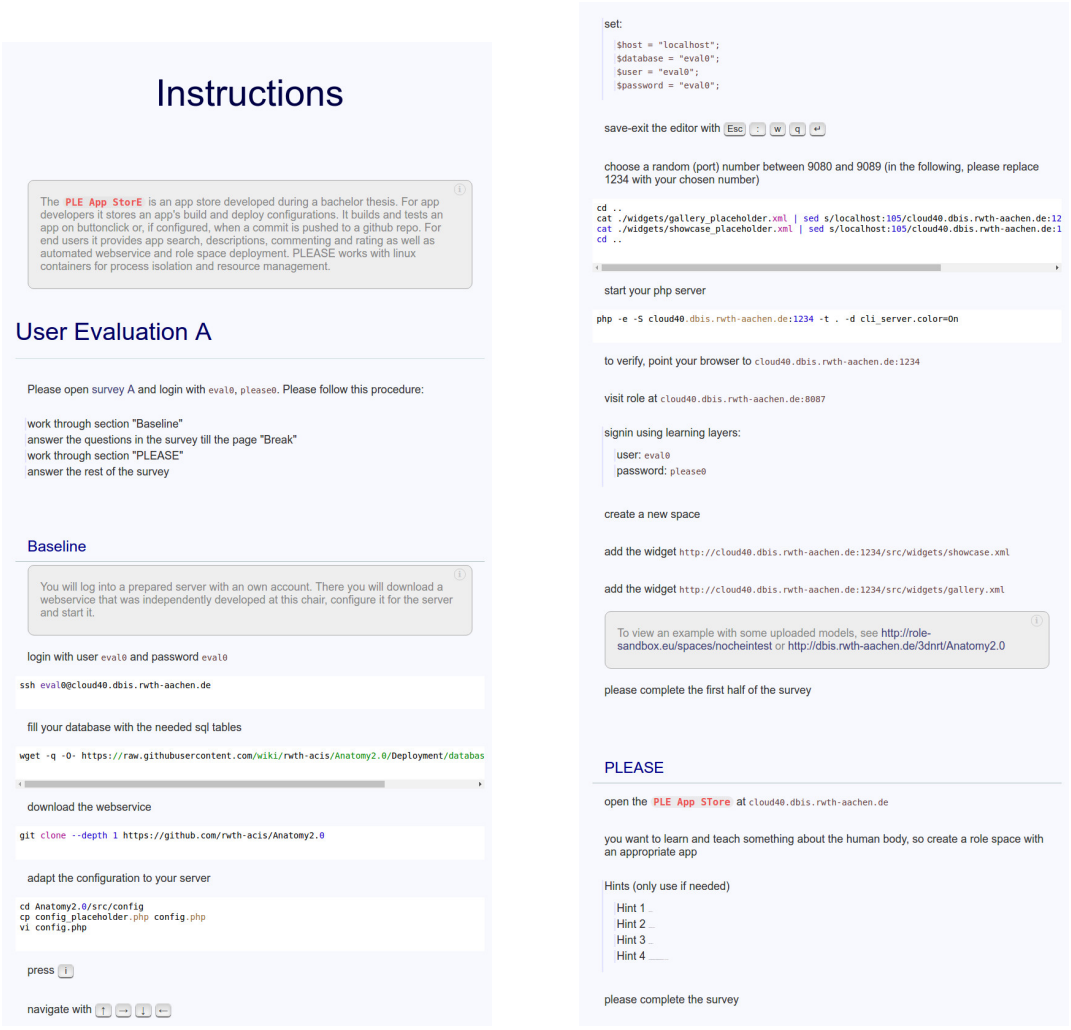
[1] http://dbis.rwth-aachen.de/3dnrt/,

```
            'wiki/rwth-acis/Anatomy2.0/Deployment/database.sql' | mysql 3dnrt

          apk add git
          git clone --depth 1 https://github.com/rwth-acis/Anatomy2.0

          apk add php5 php5-mysql php5-pdo php5-pdo_mysql php5-json sipcalc

          pkill nc
        } 2>&1 | nc -u cloud40.dbis.rwth-aachen.de 3333"
    },
    "deploy": {
      "space": {
        "base": "build",
        "title": "Anatomy2.0",
        "widgets": [
          "$ip4_http/80/src/widgets/showcase.xml",
          "$ip4_http/80/src/widgets/gallery.xml"
        ]
        "command": "
          {
            cd Anatomy2.0/
            ip=$(ip addr show eth0 | awk '/inet6/ && /scope global/ {print $2}' \
              | cut -d '/' -f 1)
            # ip4 compatibility:
            iid=$((0x$(sipcalc $ip | awk '/Expanded Address/ {print $4}' \
              | cut -d: -f 8)))
            cd ./src
            echo \"<?php
              \$baseUrl='';
              \$admins=\cite{'john@doe.com', 'John', 'Doe'};
              \$oidcClientId='342900fa-c91b-4755-9245-1d5392fb7746';
              \$host='localhost';
              \$database='3dnrt';
              \$user='3dnrt';
              \$password='abcdef';
            \" > ./config/config.php
            cat ./widgets/gallery_placeholder.xml  | sed \"s~localhost:105\"\
              \"~cloud40.dbis.rwth-aachen.de:8082/deploy/deployed/$iid/80~\" \
              > ./widgets/gallery.xml
            cat ./widgets/showcase_placeholder.xml | sed \"s~localhost:105\"\
              \"~cloud40.dbis.rwth-aachen.de:8082/deploy/deployed/$iid/80~\" \
              > ./widgets/showcase.xml
            cd ..
            sock=/run/mysqld/mysqld.sock
            php -e -S [$ip]:80 -t . \
              -d cli_server.color=On \
              -d mysql.default_socket=$sock \
              -d pdo_mysql.default_socket=$sock \
              -d 'error_reporting=E_ERROR|E_PARSE'
          } 2>&1 | nc -u cloud40.dbis.rwth-aachen.de 3333"
      }
    }
  }
}
```

B. Evaluation

B.1. Instructions

Figure B.1.: evaluation instructions A



(a) (b)

Figure B.2.: evaluation instructions B

(a) (b)

B.2. Questionnaire

B.2.1. Version A

- Please proceed to the first page.
- yes/no question “Did you already setup a Web service on a server before?”
- ordinal question “How difficult is the setup of Anatomy2.0?”, 0 = very difficult - 10 = very easy
- ordinal question “How reliable do you think is your deployment?”, 0 = very unreliable - 10 = very reliable
- ordinal question “What do you think of Anatomy2.0’s quality without actually using it?”, 0 = low quality - 10 = high quality
- Please continue with the instructions, then finish this survey.

- ordinal question “How many hints did you need?”, 0 - 4
- ordinal question “How intuitive was the discovery of the wished app via the search?”, 0 = not intuitive at all - 10 = very intuitive
- ordinal question “How difficult is the setup of Anatomy2.0?”, 0 = very difficult - 10 = very easy
- ordinal question “How reliable do you think is your deployment?”, 0 = very unreliable - 10 = very reliable
- ordinal question “What do you think of Anatomy2.0’s quality without actually using it?”, 0 = low quality - 10 = high quality
- text comment “What feature in the app search would result in a better search experience?”
- ordinal question “Would you use PLEASE to search for and to setup collaborative learning apps?”, 0 = definitely no - 10 = for sure
- text comment “What are reasons to not use PLEASE for the aforementioned task?”
- yes/no question “Did you already develop a Web service before?”
- text comment “If yes, what advantages do you see in using PLEASE to continuously test, build and deploy your service?”
- text comment “If yes, what disadvantages do you see in using PLEASE to continuously test, build and deploy your service?”

B.2.2. Version B

- Please proceed to the first page.
- yes/no question “Did you already setup a Web service on a server before?”
- ordinal question “How many hints did you need?”, 0 - 4
- ordinal question “How intuitive was the discovery of the wished app via the search?”, 0 = not intuitive at all - 10 = very intuitive
- ordinal question “How difficult is the setup of Anatomy2.0?”, 0 = very difficult - 10 = very easy
- ordinal question “How reliable do you think is your deployment?”, 0 = very unreliable - 10 = very reliable
- ordinal question “What do you think of Anatomy2.0’s quality without actually using it?”, 0 = low quality - 10 = high quality
- Please continue with the instructions, then finish this survey.
- ordinal question “How difficult is the setup of Anatomy2.0?”, 0 = very difficult - 10 = very easy
- ordinal question “How reliable do you think is your deployment?”, 0 = very unreliable - 10 = very reliable
- ordinal question “What do you think of Anatomy2.0’s quality without actually using it?”, 0 = low quality - 10 = high quality

- text comment “What feature in the app search would result in a better search experience?”
- ordinal question “Would you use PLEASE to search for and to setup collaborative learning apps?”, 0 = definitely no - 10 = for sure
- text comment “What are reasons to not use PLEASE for the aforementioned task?”
- yes/no question “Did you already develop a Web service before?”
- text comment “If yes, what advantages do you see in using PLEASE to continuously test, build and deploy your service?”
- text comment “If yes, what disadvantages do you see in using PLEASE to continuously test, build and deploy your service?”

B.3. Survey Results

Listing B.1: What feature in the app search would result in a better search experience?

```
"Nothing regarding functionality, but a more appealing design"

"better design/layout"

"search was fine. But a better explanation of what the Store does is needed"

"1. preview images of the app
2. symbols of provided functionality of the app
  (e.g. anatomy 2.0 : master and multiple viewer)"
(translation)
```

Listing B.2: What are reasons to not use PLEASE for the aforementioned task?

```
"don't know"

"Don't know"

"it makes sense to have a build server for these kinds of apps"
```

Listing B.3: If yes what advantages do you see in using PLEASE to continuously test build and deploy your service?

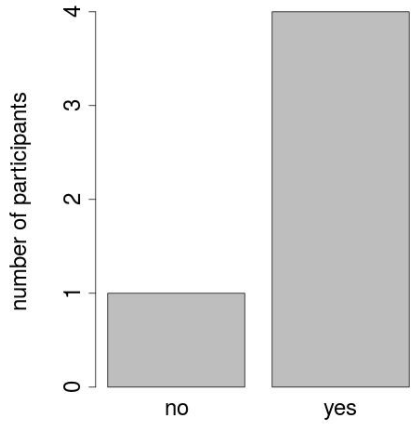
```
"Much simpler workflow and flexibility."

"Saving time"

"My users don't have to set up my stuff"
```

Listing B.4: If yes what disadvantages do you see in using PLEASE to continuously test build and deploy your service?

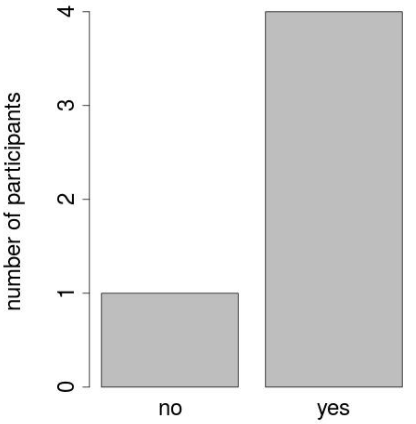
```
"The process happens on a server and thus it may be more effort
to set up the build process."

"Takes time to create test and build script, but this needs to be done only once."
```

```
"none"
```

Figure B.3.: general

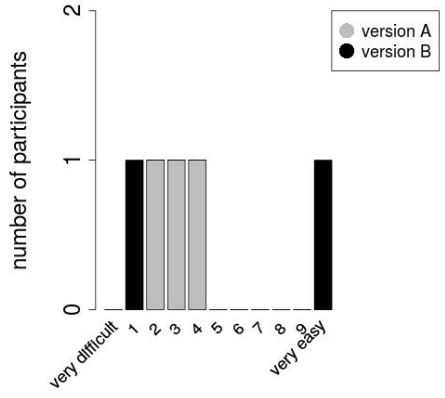


(a) Did you already setup a Web service on a server before?

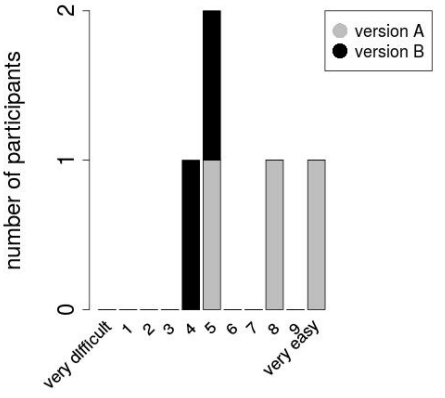


(b) Did you already develop a Web service before?

Figure B.4.: How difficult is the setup of Anatomy2.0?

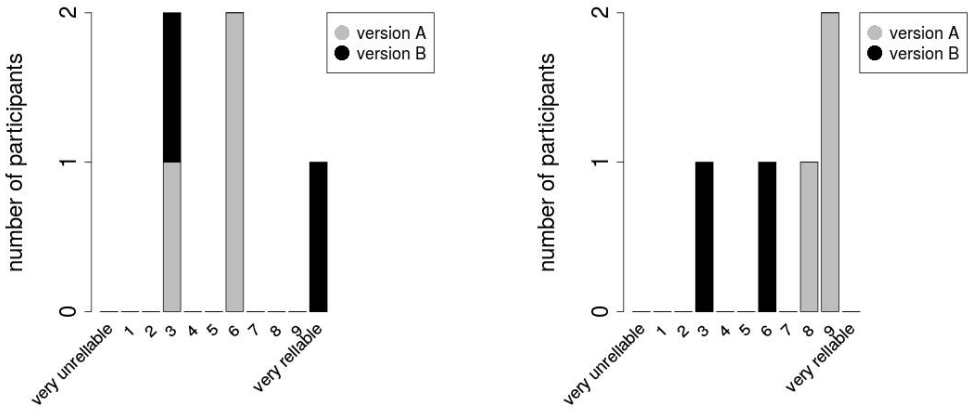


(a) Baseline



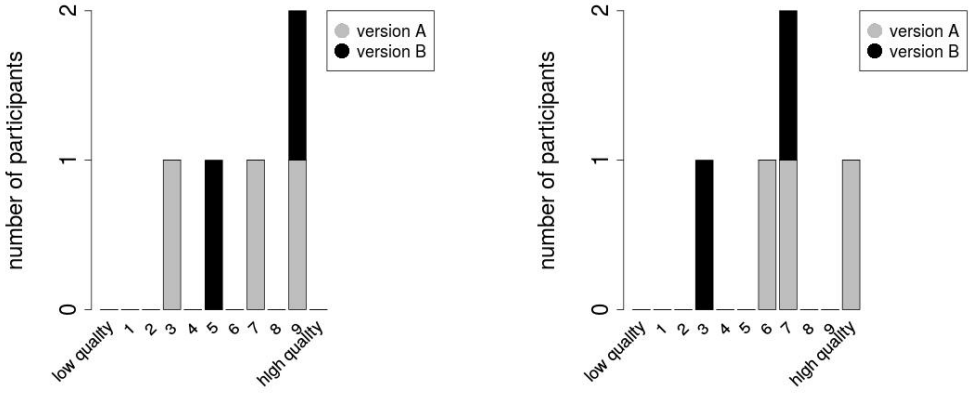
(b) PLEASE

Figure B.5.: How reliable do you think is your deployment?



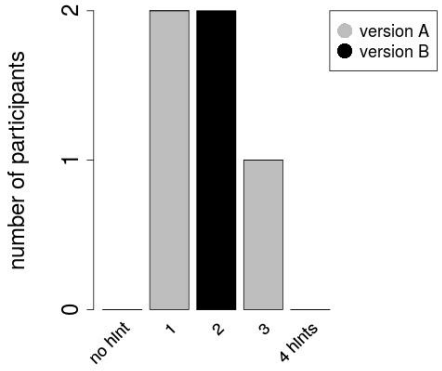
(a) Baseline (b) PLEASE

Figure B.6.: What do you think of Anatomy2.0's quality without actually using it?

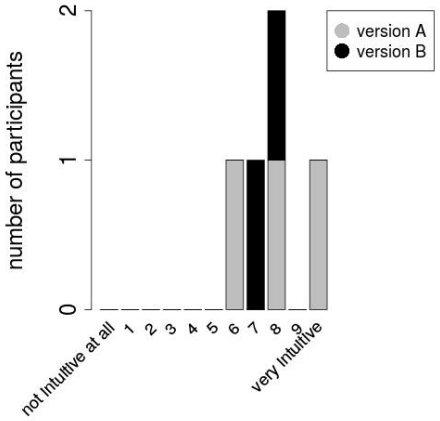


(a) Baseline (b) PLEASE

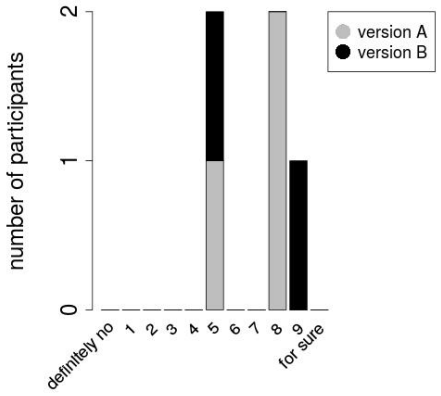
Figure B.7.: PLEASE deployment



(a) How many hints did you need?



(b) How intuitive was the discovery of the wished app via the search?



(c) Would you use PLEASE to search for and to setup collaborative learning apps?