Workplace-based e-Assessment Technology for Competency-based Higher Multi-professional Education

Deliverable 3.2: Architecture design

<table>
<thead>
<tr>
<th>Delivery month Annex I</th>
<th>Actual delivery month</th>
<th>Lead participant: UM</th>
<th>Work package:</th>
<th>Nature:</th>
<th>Dissemination level:</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>10</td>
<td>UM</td>
<td>3</td>
<td>R</td>
<td>PU</td>
</tr>
</tbody>
</table>

Version: Final 1.0

19 December 2014

Project coordinator

Dr. Marieke van der Schaaf

Utrecht University

Faculty of Social and Behavioural Sciences
Department of Education
PO Box 80.140
3508TC Utrecht
The Netherlands

Telephone: +31 (0)30 253 4944

Email: M.F.vanderSchaaf@uu.nl

This project has received funding from the European Union’s Seventh Framework Programme for research, technological development and demonstration under grant agreement no 619349.
# Table of Contents

1. Executive Summary ........................................................................................................... 4
2. Introduction ...................................................................................................................... 5
   2.1 Background and goals .................................................................................................... 5
   2.2 Scope of the deliverable ............................................................................................... 5
   2.3 Structure of this deliverable ......................................................................................... 6
3. The WATCHME architecture ............................................................................................ 7
   3.1 The overall architecture ............................................................................................. 7
   3.2 Communication between EPASS and Student Model server ....................................... 8
   3.3 Communication between modules around JIT ............................................................ 8
   3.4 Communication between modules around VIZ ........................................................... 8
4. The EPASS system ............................................................................................................ 9
   4.1 General description of EPASS ..................................................................................... 9
   4.2 EPASS API .................................................................................................................. 10
      4.2.1 Pushing data to SM ............................................................................................. 10
      4.2.2 Offering data to JIT/VIZ ...................................................................................... 10
      4.2.3 Authorisation/Privacy requests: Privacy Manager ................................................ 10
5. The Student Model Module ............................................................................................. 11
   5.1 General description of the module .............................................................................. 11
   5.2 Student Model API (Reading, review UM) .................................................................. 11
      5.2.1 Communication patterns ...................................................................................... 12
      5.2.2 API Specification ................................................................................................. 15
6. JIT and Visualisation ........................................................................................................ 18
   6.1 General description of the modules ............................................................................ 18
      6.1.1 UI level integration .............................................................................................. 20
      6.1.2 API level integration ............................................................................................ 21
      6.1.3 Security & Privacy ............................................................................................... 23
7. External Assessment tools ............................................................................................... 24
8. Data Description .............................................................................................................. 25
   8.1 EPASS Data description .............................................................................................. 25
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Privacy and social ethical design</td>
<td>28</td>
</tr>
<tr>
<td>9.1 Concerning student model issues (Reading, review Mateum)</td>
<td>28</td>
</tr>
<tr>
<td>9.2 EPASS Privacy</td>
<td>28</td>
</tr>
<tr>
<td>9.3 Privacy in communication between WATCHME modules</td>
<td>29</td>
</tr>
<tr>
<td>Conclusion</td>
<td>30</td>
</tr>
<tr>
<td>References</td>
<td>31</td>
</tr>
<tr>
<td>Figures</td>
<td>32</td>
</tr>
<tr>
<td>History of the document</td>
<td>33</td>
</tr>
<tr>
<td>13.1 Document history</td>
<td>33</td>
</tr>
<tr>
<td>13.2 Internal review history</td>
<td>34</td>
</tr>
<tr>
<td>Appendix 1: EPASS CLASS SKELETONS</td>
<td>35</td>
</tr>
<tr>
<td>Appendix 2: EPASS API description</td>
<td>37</td>
</tr>
</tbody>
</table>
1 Executive Summary

In this deliverable we describe the general architecture of the WATCHME system. In this project we investigate how an e-portfolio enriched with student models can improve workplace-based learning by offering Just-in-time feedback and adequate visualisation of relevant information. The WATCHME system will be based on an existing portfolio-system (the EPASS system by Maastricht University and Mateum), but the design needs to be applicable to other e-portfolio systems as well.

The WATCHME system is envisioned as a collection of loosely coupled modules that can be developed and tested relatively independently and deployed in a distributed manner. To allow modules to communicate with each other in order to function as a whole, APIs (application programming interfaces) are defined for each of the modules that act as corridors through which information is exchanged and functionality is invoked. All technical partners have been involved in developing and describing the architecture as laid down in this deliverable. Further details on the new modules are provided in Deliverable 4.1 (on the student model platform) and Deliverable 5.1 (on integration of just-in-time feedback and visualisation in an e-portfolio user interface).

Many requirements on the WATCHME system are still unclear, since the educational context (as defined in WP2) is under construction. Moreover, only during the formal evaluation studies (in WP6) the final needs for visualisation and just-in-time feedback will become clear. It means that the modules, including their APIs cannot yet be specified in complete detail in this deliverable. However, we provide enough detail for the modules and APIs that the hardware platform can be set up and software development within work packages 3, 4 and 5 can commence.

After explaining the general modules and their relations in Chapter 3, we go into the details for each module in Chapters 4 to 6. For each module, a general structure and the basic specification of APIs are provided. These specifications are the product of a series of meetings in workgroups defined per API in which consensus has been reached between the needs of the modules involved. The level of detail in these descriptions reflects the level of consensus that has already been reached between the developing partners, and also reflects the level of uncertainty in requirements still present in this stage of the project. Since at this stage no need for external assessment tools has been articulated, chapter 7 on this subject only describes possible integration of such tools.

In chapter 8 we describe, on a conceptual level, the data as currently available in the EPASS system for use in student modelling and visualisation. Finally in Chapter 9 we lay down how the WATCHME project deals with privacy and ethics aspects at the level of the system architecture.
2 Introduction

In this chapter we explain the background and goals of the deliverable and specify its scope as provided in the description of work (DOW, Annex 1) of WATCHME, updated with the changes as described in Deliverable 1.5 (ICT and IPR management plan).

2.1 Background and goals

In the WATCHME project, we investigate how an e-portfolio enriched with student models can improve workplace-based learning by offering Just-in-time feedback and adequate visualisation of relevant information. The WATCHME system will be based on an existing portfolio-system (the ePASS system by Maastricht University and Mateum), but the design needs to be applicable to other e-portfolio systems as well.

The choice for a loosely coupled modular approach to the WATCHME architecture has several reasons. First, a modular approach allows the system to be distributed over multiple systems. This not only gives more freedom to select different deployment and development platforms per module, but also allows for better distribution of computational resources. Second, the development of these modules is distributed over three different work packages (WP3, 4 and 5) and five technical partners. A modular approach facilitates relatively independent and simultaneous development and testing of modules.

The modules need to be able to communicate with each other, in order to function as a whole in delivering the required learning analytics functionality. To this end, APIs need to be defined for each of the modules. The APIs act as corridors through which information is exchanged between modules and functionality is invoked. All technical partners have been involved in developing and describing the architecture as laid down in this deliverable. Further details on new modules are provided in Deliverable 4.1 (on the student model platform) and Deliverable 5.1 (on integration of just-in-time feedback and visualisation in an e-portfolio user interface).

The WATCHME system will be dealing with personal data as collected in the e-portfolio. Special attention is being paid to incorporate privacy protection into the architecture design.

2.2 Scope of the deliverable

According to the DOW, this deliverable is a design document that specifies the system architecture of the learning-analytics enhanced portfolio system. This system will consist of the following elements:

1. An existing electronic portfolio system (EPASS by Maastricht University and Mateum)
2. External assessment tools (such as a game or simulator) – if applicable
3. A student model engine to be developed in WP4
4. Just-in-Time feedback module to be developed in WP5
5. Visualisation module to be developed in WP5

The document should be described in such way that the modules in Work Packages WP4 and WP5 can develop independently from and in parallel to WP3 and each other. This document defines the
requirements for system architecture and will become part of the requirement document (deliverable 3.1). The existing electronic portfolio system will only be described as far as needed to support the integration into the system. This description will be put in a generic way such that another electronic portfolio system could take its role provided that it adheres to the requirements and architectural design. The document will describe the following aspects:

- Description of the above mentioned modules
- A conceptual description of communication protocols (e.g. web services) or APIs that link the modules between which information needs to be exchanged
- The Privacy protective and socio-ethical design features to provide for (cf. Deliverable 1.5):
  - Informed Consent Process implementation within the user interface prior to the point of registration by a Workplace Trainee into the Portfolio System (this is part of the current EPASS system)
  - Allowing profile elements to be updated by the user (this is part of the current EPASS system)
  - Server-side generation of the instantiated models
  - Ensuring personal student models are deleted on request by users

For the modules being developed within the project the following aspects need to be described if applicable:

- Deployment platform, operating system
- Development platform
- Database platform
- Software modules, libraries, used

The system architecture document needs to be approved by the WP-leaders of WP3, WP4 and WP5. Subsequent updates might be necessary when requirements are changed. New approval is requested in such case.

2.3 Structure of this deliverable

In chapter 3 the overall architecture and a general description of the modules are provided. Chapter 4 will explain the EPASS system and its new API for connection to the WATCHME modules. Chapter 5 is dedicated to the Student Model module and its API, and Chapter 6 to the just-in-time feedback and visualisation modules. In Chapter 7, we spend some words on possible external assessment tools. Data that will be passed between the modules in WATCHME is described at a conceptual level in Chapter 8. Last, but not least, privacy and security aspects of the architecture and its modules are detailed in Chapter 9.
3 The WATCHME architecture

In this chapter we first give a conceptual overview of the whole architecture, indicating the modules and their relationships to each other (3.1). Next we describe the current e-portfolio system EPASS (3.2). Then we provide the descriptions of the new student-model module to be developed in WP 4 (3.3) and the new modules for just-in-time feedback and visualisation to be developed in WP 5 (3.4).

3.1 The overall architecture

The main architecture of the WATCHME- system as envisioned is indicated on a conceptual level in Figure 1.

In general, the three modules: EPASS, the Student Model module and the JIT/VIZ module all have their own REST API [1] that each communicate with their peer APIs. The details of these communications are explained below. A general data language is developed that describes how portfolio-information is translated in order to be transported between APIs.

Next to the three modules, a central authentication server (based on OAuth) will be included that arranges secure authentication between the three modules.
The student model and JIT/VIZ modules are envisioned here to reside on one physical sever separate from the EPASS server, but other deployment schemas are possible (see 3.3).

The architecture of the student model module is explained below and further detailed in deliverable 4.1. The user interface of the JIT/VIZ module is, from the perspective of the user integrated into the EPASS user interface (as explained below and in Deliverable 5.1).

3.2 Communication between EPASS and Student Model server

In principle, EPASS will send portfolio updates to the student model server (SM) on a nightly basis. These updates will be anonymized (as much as is possible in an automated way) and a system for privacy enhanced user identities will be applied.

The data processed by the Student Model Server will be erased immediately after the update was taken into account. Whenever EPASS signals the SM that a submission has been invalidated, the SM will erase the existing models and request a new image of the student protocol. This sequence is described in more detail in the Student Model API section (5.2).

3.3 Communication between modules around JIT

Users request a just-in-time feedback message through the EPASS user interface. To this end, EPASS either offers widgets (provided by the JIT/VIZ UI) within the EPASS GUI or forward requests to the JIT/VIZ API. This module communicates with the SM API to collect an appropriate feedback message for the user and delivers this data inside a JIT/VIZ UI widget that is integrated into the EPASS UI.

3.4 Communication between modules around VIZ

Similarly, users request visualisations through the EPASS interface, via widgets provided by the JIT/VIZ UI. Whenever the user engages a visualisation session, the JIT/VIZ module gathers aggregated data as needed from EPASS portfolio database via the EPASS API and from the SM via the SM API.

The SM module in combination with the EPASS API take care that users are not allowed to visualize data for which they do not have access rights. This should include statistical intractability (e.g. small numbers).
4 The EPASS system

In this chapter we first provide a general description of the EPASS system as far as is needed to understand its role in the WATCHME architecture. Then we provide some specifications of the API for EPASS needed to communicate and interact with the WATCHME modules.

4.1 General description of EPASS

EPASS is an online web-based portfolio system which supports competency-based education and training programmes. EPASS offers learners (students, trainees, professionals) and their educators/assessors an electronic portfolio to monitor and provide evidence of their learning and competency development.

At the moment, the following features are available in EPASS:

- Workplace-based assessment forms;
- Multisource feedback module;
- Support for themes, critical professional situations and EPAs;
- Several feedback and reflection forms (including self-evaluations) on (the development of) competencies;
- Forms for entering attendance of educational activities, performance of teaching tasks and other (academic) activities;
- Adjustable competency models.

Relevant summaries of feedback and workplace-based assessments are automatically presented in different types of overviews (spider web diagrams, line charts, tables, bar charts), which are based on scores (on the different competencies) from the inserted forms. Other overviews are also displayed, for example numerical overviews of the total inserted forms and an overview of the progress within themes or EPAs, based on all activities, forms and procedures linked to it.

EPASS is developed and owned by University Maastricht and Mateum. For more information see the EPASS website: www.epass.eu

On an architecture level we can describe EPASS as Software as a Service (SAAS) built on the PHP/MySQL stack using HTML/CSS/JavaScript for the UI. The EPASS system uses separate databases and servers for client domains. EPASS makes use of a shared identity management system that allows users access to several roles in different configurations.

EPASS offers a dedicated SSL-based Single-SignOn service. Several dedicated procedures for uploading portfolio materials have been developed as well as procedures to export portfolios in PDF and XML format. Recently EPASS started offering an OAuth-based REST API [1] to allow applications to access and update portfolios.
4.2 EPASS API

The EPASS API will contain three different services: pushing data to the SM, offering specific data to VIZ and JIT on request and dealing with authorisation/privacy requests. When other e-portfolio systems are intended to implement the WATCHME modules, this API has to be implemented for these e-portfolio systems by the e-portfolio provider.

4.2.1 Pushing data to SM

Strictly speaking, this part is not a restful API but a module that communicates with the SM API. This module will, every night, collect changes in portfolios, export them to JSON format (cf. Appendix 1 for a conceptual description) and upload these files to the SM API as described below.

Whenever validated assessments are changed for any reasons, the SM will be notified and data will be resent (Rollback/Replace) after a request by the SM.

4.2.2 Offering data to JIT/VIZ

This is a REST API that offers (aggregated) data on request. See Appendix 2 for more details on this API. In principle, the API will follow a structure that is based on the class skeletons in Appendix 1.

The EPASS API will use hash codes with a limited life span to replace user IDs in communication with SM and JIT/VIZ.

4.2.3 Authorisation/Privacy requests: Privacy Manager

The EPASS API will contain an area (the Privacy Manager) that deals with authorisation/privacy requests such as: is user xxx allowed to receive JIT messages about user yyy? Is user xxx allowed to see aggregated data over group xxx?

For xxx, yyy the user hash codes are used, a description of groups for aggregated views is to be defined, maybe restricted to: curriculum phase, location, and discipline. Alternatively a list of student-hashes needs to be provided.
5 The Student Model Module

Although the Student Model Module is developed and described in work package 5, it’s necessary to give some insight in the module in order to understand its role in the overall architecture and to specify aspects of the API.

5.1 General description of the module

According to Deliverable 4.1, the preliminary ideas regarding the Student Model Module include several elements:

- **External Student Model API** – which is used to link the SM to the EPASS and JIT/VIZ API. This API is described in the next sections of this deliverable.
- **Data (pre- and post-) Processing Module** – used to communicate with the student model external API and convert the data from/into internal student model structures. This module will be described in more detail in Deliverable 4.2.
- **Bayesian Student Model** – comprises the domain and individual student models. The structure of this module will be described in more details in Deliverable 4.2.
- **Student Model Database** – could potentially store the student models and different parameters required at runtime. The module will be described in more details in Deliverable 4.2.

Figure 2 presents the interaction between the Student Model API and different internal and external components.

![Student Model API component diagram](image)

5.2 Student Model API (Reading, review UM)

Before we go into details of the student model API we start with some preliminary notes:

- The Student Model External API will use a RESTful API [1]
- EPASS can push and query data from the SM: HTTP GET, POST and DELETE
- VIZ module can query, filter or aggregate student model output: HTTP GET
- JIT module can query, filter, aggregate or post data: HTTP GET and POST
- JIT/VIZ modules use a common layer to communicate with the Student Model
The SM would expose 2 methods corresponding to different operations:

1. Query data, which can be used to retrieve the latest image of a given data type for a certain student or cohort of students. The query API specifies the student and/or group, domain and the required data.

2. Post data, support several operations (i.e. Update, Replace, Rollback or Delete). These operations correspond to different functions supported by the SM.

5.2.1 Communication patterns

Authorisation and Privacy Management will be kept by EPASS, by using existing APIs. The Student Model uses a Post-Query Pattern, presented in Figure 3. The Publisher in this scenario can either be the EPASS system or the JIT module. The Privacy Manager is a component that manages the access rights for a given user (e.g. which supervisor has the rights to see private student data). The clients in this scenario are all the modules who can benefit from SM data: JIT, VIZ and EPASS.

The information flow in this scenario initiates from the publisher (e.g. EPASS or JIT) and, once processed, it is made available to the clients. While the query is computed, a preliminary privacy query is done with the existing privacy manager implemented in EPASS.

The client does not necessarily need to be informed right after the data is available. The SM will keep an updated image of the latest data available. Once ready, the clients will query the Student Model to retrieve this image.
For the delete operation, shown in Figure 4, the SM receives the request from EPASS, usually when the student demands it. After this operation is performed, both the student data and last image of the student model are deleted and any future query is replied with an empty dataset.

![Data Flow Diagram](image)

**Figure 4:** The data flow used by the Student Model for the delete operation

Figure 5 shows the invalidate operation. The SM also receives the request from EPASS, usually when certain forms were invalidated in the portfolio. After this operation is performed, the student data is deleted but the last image of the student model is still available for future queries. EPASS will continue by re-sending all the valid data and once these have been processed, the new SM image will be available.
From the API perspective, the delete and invalidate methods differ in only one parameter describing which operation should be executed by the SM.

The information flow, in this scenario, initiates from the publisher (e.g. EPASS or JIT) and, once processed, it is made available to the clients. While the query is computed, a preliminary privacy query is done with the existing privacy manager implemented in EPASS.
5.2.2 API Specification

As stated above, the Student Model API exposes only two methods; query the latest data image for a given student and post to send update data for a given student. The SM exposes two generic structures: Query and Post. These structures contain (or return) some abstract Data structures, which are either coming from EPASS or JIT/VIZ modules.

**POST Method**

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
<td>/SM/API/</td>
</tr>
<tr>
<td>HTTP Method</td>
<td>POST</td>
</tr>
<tr>
<td>Input type</td>
<td>TPost</td>
</tr>
<tr>
<td>Input format</td>
<td>JSON</td>
</tr>
</tbody>
</table>
| Output return codes | HTTP 200 (OK)  
                       | HTTP 401/403 [Reject] |
| Output type   | TData*    |
| Output format | JSON      |

**TPost:**

<table>
<thead>
<tr>
<th>Field name</th>
<th>Data type</th>
<th>Necessity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
<td>URL</td>
<td>M</td>
<td>The source of the call (e.g. URL of EPASS)</td>
</tr>
<tr>
<td>Topic</td>
<td>String[ ]</td>
<td>M</td>
<td>The list of selected topics to be queried. Topics are directly linked</td>
</tr>
<tr>
<td>Student ID</td>
<td>UUID</td>
<td>O</td>
<td>The unique ID is in the format proposed by the Privacy Manager.</td>
</tr>
<tr>
<td>Group ID</td>
<td>UUID</td>
<td>O</td>
<td>This unique ID represents the student cohort ID to be queried</td>
</tr>
<tr>
<td>Model ID</td>
<td>UUID</td>
<td>M</td>
<td>This unique ID represents the domain model: undergraduate medical education and anaesthesiology, veterinary training, teacher training</td>
</tr>
<tr>
<td>Content</td>
<td>TData[ ]</td>
<td>M</td>
<td>The content of data to be posted. This data is linked with the selected topic.</td>
</tr>
</tbody>
</table>

1 The necessity can be either mandatory (M) or optional (O)
QUERY Method

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
<td>/SM/API</td>
</tr>
<tr>
<td>HTTP Method</td>
<td>GET</td>
</tr>
<tr>
<td>Input type</td>
<td>TQuery</td>
</tr>
<tr>
<td>Input format</td>
<td>JSON</td>
</tr>
<tr>
<td>Output return codes</td>
<td>HTTP 200 (OK )</td>
</tr>
<tr>
<td></td>
<td>HTTP 401/403 [Reject]</td>
</tr>
<tr>
<td>Output type</td>
<td>TData*</td>
</tr>
<tr>
<td>Output format</td>
<td>JSON</td>
</tr>
</tbody>
</table>

TQuery:

<table>
<thead>
<tr>
<th>Field name</th>
<th>Data type</th>
<th>Necessity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
<td>URL</td>
<td>M</td>
<td>The source of the call (e.g. URL of the JIT/VIZ module)</td>
</tr>
<tr>
<td>Topic</td>
<td>String[ ]</td>
<td>M</td>
<td>The list of selected topics to be queried. Topics are directly linked</td>
</tr>
<tr>
<td>AuthToken</td>
<td>String</td>
<td>M</td>
<td>The authorisation token required by the Privacy Manager to authorize the access to student data</td>
</tr>
<tr>
<td>Student ID</td>
<td>UUID</td>
<td>O</td>
<td>The unique ID is in the format proposed by the Privacy Manager.</td>
</tr>
<tr>
<td>Group ID</td>
<td>UUID</td>
<td>O</td>
<td>This unique ID represents the student cohort ID to be queried</td>
</tr>
<tr>
<td>Model ID</td>
<td>UUID</td>
<td>M</td>
<td>This unique ID represents the domain model: undergraduate medical education and anaesthesiology, veterinary training, teacher training</td>
</tr>
<tr>
<td>Parameters</td>
<td>TData</td>
<td>O</td>
<td>Optional query parameters, required by some topics</td>
</tr>
</tbody>
</table>

DELETE Method

2 The necessity can be either mandatory (M) or optional (O)
**Workplace-based e-Assessment Technology for Competency-based Higher Multi-professional Education**

### Field Value

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
<td>/SM/API</td>
</tr>
<tr>
<td>HTTP Method</td>
<td>DELETE</td>
</tr>
<tr>
<td>Input type</td>
<td>TDelete</td>
</tr>
<tr>
<td>Input format</td>
<td>JSON</td>
</tr>
<tr>
<td>Output return codes</td>
<td>HTTP 200 (OK)</td>
</tr>
<tr>
<td></td>
<td>HTTP 401/403 (Reject)</td>
</tr>
<tr>
<td>Output type</td>
<td>-</td>
</tr>
<tr>
<td>Output format</td>
<td>-</td>
</tr>
</tbody>
</table>

#### TDelete:

<table>
<thead>
<tr>
<th>Field name</th>
<th>Data type</th>
<th>Necessity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
<td>URL</td>
<td>M</td>
<td>The source of the call (e.g. URL of the JIT/VIZ module)</td>
</tr>
<tr>
<td>Student ID</td>
<td>UUID</td>
<td>O</td>
<td>The unique ID is in the format proposed by the Privacy Manager.</td>
</tr>
<tr>
<td>Group ID</td>
<td>UUID</td>
<td>O</td>
<td>This unique ID represents the student cohort ID to be queried</td>
</tr>
<tr>
<td>Model ID</td>
<td>UUID</td>
<td>M</td>
<td>This unique ID represents the domain model: anaesthesiology, veterinary training, pedagogical training</td>
</tr>
<tr>
<td>Invalidate</td>
<td>boolean</td>
<td>O</td>
<td>Selects the operation to be executed by the Student Model, either invalidate (true) or delete (false). By default this parameter should be false</td>
</tr>
</tbody>
</table>

*) TData is a generic data type. For post methods, the EPASS data structures are used. For Query purpose, the data will be defined later (e.g., in Deliverable 4.2).

3 The necessity can be either mandatory (M) or optional (O)
6 Just-in-time feedback and Visualisation

Similar to the Student Model Module, the Just-in-time feedback and Visualisation modules are described in more detail within work package 5 (cf. Deliverable 5.1).

6.1 General description of the modules

This section outlines the proposed high-level architecture for integrating the just-in-time feedback (JIT) and visualisation (VIZ) modules within the rest of the WATCHME system landscape. For a detailed description of the integration architecture, including EPASS specifics, refer to deliverable 5.5.1: Description of integration of systems and modules.

How the JIT and VIZ modules fit within the overall system architecture is illustrated in Figure 6, below:

![Figure 6: Roles of JIT and VIZ modules within the WATCHME system landscape](image)

The main integration points of JIT and VIZ are:

- The electronic portfolio and assessment support system (e.g. EPASS) which provides the system’s user interface and captures/persists core data.
- The Student Model (SM) from which new information relevant to VIZ and, in particular, JIT is synthesised, based on data from the electronic portfolio.

As the JIT and VIZ modules are both stand-alone modules embedded into a hosting system, these subsystems will be integrated using a unified architecture supporting both modules. This is illustrated, from a logical perspective, in Figure 7 below:
A concrete implementation based on this architecture will encounter constraints in the choice of implementation technology and deployment. Specifically, the implementation of the modules embedded within the hosting portfolio system must support close interaction with that host, e.g. be based on web technologies such as JavaScript and HTML if the host itself is a web-based system.

This is highlighted in the logical deployment perspective of Figure 8 below, in which the artefacts of each tier of Figure 7 are grouped by (potentially) independent technology stacks:
Figure 8: Deployment view of JIT/VIZ integration architecture

Note that Figure 8 is a logical deployment view, a physical view of a concrete implementation based on this architecture may choose to host modules such as SM API and JIT/VIZ API on a single web server node, subject to technology stack compatibility and performance considerations.

As evident from Figure 7, integration takes place at two system levels: at the UI level (integrating both visualisations from VIZ and feedback from JIT into the portfolio system UI by embedding JIT/VIZ specific artefacts into the host) and at the API level (facilitating communication and transfer of data between the involved subsystems).

6.1.1 UI level integration

The approach to integrating both visualisations from VIZ and feedback from JIT into the EPASS UI is to embed JIT/VIZ specific artefacts into the hosting portfolio system UI.

From a structural perspective, this means:

- Referencing JIT/VIZ specific code from the host
- Embedding JIT/VIZ content placeholders in the host UI
For interactive content, embedding JIT/VIZ within the host UI allows obtaining user input to visualisations or feedback. This is illustrated in Figure 9 below:

Parameters supply contextual information such as user identification, session token, data sources and any instance specific configuration.

6.1.2 API level integration

The JIT/VIZ APIs use a network-based application architecture, and is specifically based on the REST architectural style [1]. HTTP is the applied application protocol, transported over TCP/IP, with JSON being used for resource representations [2].
The API handles two separate concerns:

- Providing a uniform datasource abstraction for, and at the same abstraction level as, the JIT and VIZ modules, mapping those to concrete repositories of data in a given deployment, such as portfolio system and SM.
- Providing server side rendering of visualisations for use on constrained clients that cannot handle client-side rendering.

This is illustrated in the following component diagram:

![Diagram](image)

**Figure 10: JIT/VIZ API endpoints**

<table>
<thead>
<tr>
<th>Interface</th>
<th>Nature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIZ DataSource</td>
<td>Read</td>
<td>Provides data series for consumption by concrete visualisations. Data sources do not directly reflect the underlying data sources (e.g. portfolio system repository) but expose data series relevant to a particular class of visualisation. Data sources must be configured for a particular deployment of the system.</td>
</tr>
<tr>
<td>JIT DataSource</td>
<td>Read/Write</td>
<td>Provides action sets (e.g. &quot;Take assessment&quot;) for consumption by concrete feedback implementations. Data sources (e.g. &quot;What should I do next?&quot;) must be configured for a particular deployment of the system. Optionally supports receiving notification from a JIT controller about a particular action taken by the user, in the event that the SM integration needs to be notified outside the SM-portfolio system integration.</td>
</tr>
<tr>
<td>VIZ Rendering</td>
<td>Read</td>
<td>Provides optional off-client rendering support for visualisations that cannot be handled natively for a particular client type.</td>
</tr>
</tbody>
</table>
6.1.3 Security & Privacy

Authentication and authorisation are already concerns on the portfolio system. With JIT and VIZ being embedded into the portfolio system’s user interface, it is natural to propagate authentication and authorisation concerns for the JIT/VIZ modules to the portfolio system as well.

After authenticating a user in the hosting portfolio system, it will pass an opaque session context to the components embedded within it. This context, validated by the portfolio system, will propagate, as a token passed to the JIT/VIZ controllers, to all requests made against the JIT/VIZ API interfaces. From there, propagation continues to the underlying portfolio system and SM APIs (in the latter case, propagation continues to the portfolio system), where the token is ultimately verified to ensure that the requests being made against them are allowed in the given context (authorisation).

Information privacy is handled at the transport level (i.e. by TLS/SSL)
7 External Assessment Tools

The use of external assessment tools has been discussed with end users and the technical partners in the project. At the moment, there is no need for linking external assessment tools to EPASS.

However, several internal assessment tools will be used, as is described in deliverable 3.3. For the different domains, three tracks have been defined: (1) undergraduate medical education and anaesthesiology; (2) veterinary medicine; (3) teacher training. All tracks use different assessment tools (feedback markers) in their education plans; these different tools will be incorporated in EPASS internally. In deliverable D3.3, the different tools are described per track. For the anaesthesiology track, the assessment tools ‘Mini-CEX’, ‘OSATS’ and ‘Multisource Feedback’ are indicated. In teacher training and veterinary medicine comparable tools are being used.

When, during the project, external assessment tools need to be included, the provider of the tool need to take care that the assessment information is included in the EPASS portfolio. EPASS offers several APIs to upload information from an external assessment instrument to the portfolio of a given student (also see Appendix 2 for one of the APIs).
8 Data Description

In this chapter we describe the data as contained in EPASS portfolios. This data is used to populate student models and feed visualisations. Other data to be exchanged between WATCHME modules has been described in chapters 5 and 6.

8.1 EPASS Data description

This section gives an overview and description of the different types of portfolio data that is available in EPASS and can be requested by the student model, JIT and VIZ. The description is at a conceptual, external level (see Figure 11). More detailed class skeletons are provided in Appendix 1.

The portfolio of a learner (student/resident) consists of several parts:

• General portfolio info
• Submissions
• Competency scores
• Activities
• Entrustabilities
• Themes

**General portfolio info** - The general portfolio information contains data about the curriculum phase of the student (year, semester), the current location (e.g., hospital) of the student, and the discipline (specialty). In EPASS, several client domains are grouped in configurations that share a database, competency models, forms and other common assets.

**Submissions** - Submissions can be divided into two categories: those filled in by the portfolio owner without any validation and those (filled in and) validated by an assessor. These are called assessments.

Each submission refers to a form that has been used to create in the submission. Some forms allow multiple rounds, each having their own form items and answers. A submission header contains contextual information on the location of the submission, the discipline, the submission, change and validation dates and on the person who submitted. A submission has a status field concerning validation. The header might also contain a reference to a theme in which the submission is placed.
Submissions contain the answers to the items in the form. These answers can either be ordered values (Likert scales [3]), symbolic values, numerical or text. Finally, each submission can contain a series of file attachments, with meta-information (date, title, remark).

**Assessments** - An assessment contains all this data but also has information on the assessor or validator, the date on which the assessed activity took place (reference date) and a context description. Assessments can be linked to specific EPAs. In case of a video-assessment, a set of annotation texts will be part of the data.

**Forms and Competency scores** - The items in a form can be linked to (one or more) competencies. In that case, the answers to these items are used to compute competency scores, which are the average score per competency in a given assessment.

Based on the competency scores per assessment, aggregated scores for a student are calculated and stored per (sub-) competency for each curriculum phase. Each time an assessment is
(de)validated or removed the competency scores are recalculated. The competency scores of a portfolio are the competency scores for the current curriculum phase of the portfolio owner.

**Activities** - Activities that are performed by a user during their training (for example performing a hip surgery) can be stored in activity logs. The activity itself has a name and group/category to which it belongs. An activity log entry contains the following data: date, context, Miller level [4] and an optional assessment linked to the activity.

**Entrustability** - When a user has performed a certain activity several times on a specific level, he/she can request the supervisor for an entrustability level for that activity. An entrustability status consists of: a reference to the activity, a Miller level [4], a date of the decision and the status of the request.

**Themes** - Within an educational programme, themes might be defined that group learning goals and activities into logical units. Such themes can be represented in the portfolio as a collection of portfolio items that are relevant for the theme and give an insight in the progress of the portfolio owner for a theme. A theme status can be requested by the portfolio owner thinks to be capable for the theme. The data of a theme status are a reference to the theme, a status field (“completed”) and a signature date. The portfolio owner and the assessors decide which items are linked to a theme.

**EPA** - An EPA as defined in WP2 has similarities with a theme in EPASS. The difference is that an EPA has levels of entrustability and is more predefined in the required items to approve a certain level of entrustability. It is possible to implement these levels of entrustability into the EPASS theme status. Another concept in EPASS that is strongly related to EPA is the “KBS”, i.e. critical professional situation which is implemented in EPASS and does already allow the registration of entrustability levels. Both themes and KBS can be used to implement an EPA, depending on the context.

**Portfolio Exports** - The portfolio data will be uploaded to the SM module using a JSON-based format [2]. The structure of this format is specified at a conceptual level in appendix 1.
9 Privacy and social ethical design

This chapter deals with the description of the architectural design addressing privacy, security and social ethical aspects. The first section deals with the Student Model Module, then we discuss privacy and security in the EPASS system and finally we go into the communication between the WATCHME modules.

9.1 Concerning Student Model issues (Reading, review Mateum)

Within an e-portfolio system, measure should be taken to ensure privacy and data security. In the case of the EPASS system, this is already the case. Since the Student Models will reside on a separate system outside the e-portfolio environment, the Student Model Module must also take care of these issues. In order to achieve the requested performance level, it is needed to store the student-specific parts of the Student Models. These will be updated whenever the portfolio-system sends new information on that trainee.

The first concern is that the Student Model Module should store as little privacy-related information as possible. There are a number of measures connected to this: (a) privacy enhanced keys should be used to identify individual trainees, (b) any information that is received by the Student Model module from the e-portfolio-system (such as narrative fields) should be erased after it has been processed, (c) models or model parts that have become obsolete should be erased.

The last concern is the protection of privacy outside the servers: (a) all communication between e-portfolio system, the Student Model Module and client stations should be encrypted; (b) any data that is used off-line to generate and train the Student Model should be treated as privacy data and accordingly local measures concerning storage, access control and removal of data should be arranged.

9.2 EPASS Privacy

Security and safety
EPASS has multiple security levels.

1. Each user receives a personal password, which can be changed by the user.
2. Data is transmitted between the user’s PC and the central system via a secure connection (secure-socket-layer). This is visible from ‘https’ in the address bar and the little lock at the bottom right of the browser screen.
3. Data is stored on a secure server in a data centre by staff specialised in internet security.
4. Access to the server and the database for maintenance is strictly secured.
5. A strict back-up regime with encrypted and saved back-ups.
Protection of privacy
Users are the owners of their portfolios. They determine who can access their portfolio and when. The different roles in relation to the portfolio, i.e. residency programme director, assessor and mentor, are assigned by the user. Once access is granted, the person in question can access the entire portfolio at any time, enabling close observation of what is and what is not going well in the learner’s development. Assessors who have not been granted access to the portfolio can view only their own assessments and not those of colleagues. In this way, access to the portfolio is restricted to people to whom access has been granted by the user, with some of them having limited access only. The secure connection with the system prevents interception of portfolio data. Both central and local system management are bound to strict secrecy. The consent that all users sing when starting with EPASS states that data can be used for research purposes provided it is anonymised.

Information Security Policy
The EPASS team follows an active information security policy according to ISO27001/2 and the Dutch standard NEN 7510 for information security in Health Care.

9.3 Privacy in communication between WATCHME modules

All communication between physical systems (the e-portfolio system, the Student Model server (including the JIT/VIZ module)) and client stations will be encrypted using up-to-date SSL encryption.

A state of the art Authentication server will be implemented that arranges dynamic secure authentication between the modules EPASS, SM, JIT/VIZ.

Moreover, a method for dynamic enhanced privacy ID (still to be decided, but based on OAuth2) will be applied in communication between the API, provided by the EPASS API.
10 Conclusion

The specification of the WATCHME system architecture as laid down in Chapters 3 – 6 of this deliverable is the product of a series of meetings in workgroups defined per API in which consensus has been reached about the needs of the modules involved. The level of detail in these descriptions reflects the level of consensus that has already been reached between the developing partners, and also reflects the level of uncertainty in requirements still present in this stage of the project.

Since at this stage no need for external assessment tools has been articulated, chapter 7 on this subject only had described possible integration of such tools. In chapter 8 we described, on a conceptual level, the data as currently available in the EPASS system for use in student modelling and visualisation. Finally in Chapter 9 has explained how the WATCHME project deals with privacy and ethics aspects at the level of the system architecture.

Many requirements of the WATCHME system are still unclear, since the educational context (as defined in WP2) is under construction. Moreover, only during the formal evaluation studies (in WP6) the final needs for visualisation and just-in-time feedback will become clear. It means that the modules, including their APIs cannot yet be specified in complete detail in this deliverable. However, we have provided enough detail for the modules and APIs that the hardware platform can be set up and software development within work packages 3, 4 and 5 can commence.
11 References


12 Figures

Figure 1: Conceptual overview of the WATCHME architecture. .......................................................... 7
Figure 2: Student Model API component diagram ............................................................................... 11
Figure 3: The data flow used by the Student Model ........................................................................... 12
Figure 4: The data flow used by the Student Model for the delete operation ................................. 13
Figure 5: The data flow used by the Student Model for the invalidate operation ......................... 14
Figure 6: Roles of JIT and VIZ modules within the WATCHME system landscape ..................... 18
Figure 7: Logical view of JIT/VIZ integration architecture ............................................................. 19
Figure 8: Deployment view of JIT/VIZ integration architecture ....................................................... 20
Figure 9: Structural view of JIT/VIZ UI integration architecture .................................................. 21
Figure 10: JIT/VIZ API endpoints .................................................................................................... 22
Figure 11: External conceptual data model EPASS .......................................................................... 26
## 13 History of the document

### 13.1 Document history

<table>
<thead>
<tr>
<th>Version</th>
<th>Author(s)</th>
<th>Date</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Draft 1.0</td>
<td>All WP3</td>
<td>23 October 2014</td>
<td>Initial version</td>
</tr>
<tr>
<td></td>
<td>All WP3</td>
<td>5 November 2014</td>
<td>Updates, additions</td>
</tr>
<tr>
<td>Draft 2.0</td>
<td>Inge Duimel, Jeroen Donkers</td>
<td>14 November 2014</td>
<td>General restructure, redaction, EPASS info</td>
</tr>
<tr>
<td>Draft 2.1</td>
<td>Ovidiu Serban, Jaime Costa, Atta Badii, Daniel Thiemert</td>
<td>17 November 2014</td>
<td>Contributions on the Student Model and Student Model API</td>
</tr>
<tr>
<td>Draft 2.2</td>
<td>Mads Troest, Steen Lehmann</td>
<td>18 – 25 November 2014</td>
<td>JIZ/VIZ improvements</td>
</tr>
<tr>
<td>Draft 2.3</td>
<td>SM API subgroup</td>
<td>3 December 2014</td>
<td>Student model API</td>
</tr>
<tr>
<td>Draft 3.0</td>
<td>Jeroen Donkers</td>
<td>3 December 2014</td>
<td>General redaction, introductions</td>
</tr>
<tr>
<td>Draft 3.0</td>
<td>Ovidiu Serban, Mads Troest, Rik Wijnen</td>
<td>3-10 December 2014</td>
<td>Corrections, comments</td>
</tr>
<tr>
<td>Draft 4.0</td>
<td>Jeroen Donkers</td>
<td>10 December 2014</td>
<td>General redaction, Summary</td>
</tr>
<tr>
<td>Final 1.0</td>
<td>Jeroen Donkers</td>
<td>19 December 2014</td>
<td>Processed remarks from internal review</td>
</tr>
</tbody>
</table>
## 13.2 Internal review history

<table>
<thead>
<tr>
<th>Internal Reviewer</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rik Wijnen (Mateum)</td>
<td>10 December 2014</td>
<td>Comments on Figures 2 and 5</td>
</tr>
<tr>
<td>Ovidiu Serban (UoR)</td>
<td>11 December 2014</td>
<td>Corrections in Figures</td>
</tr>
<tr>
<td>Daniel Thiemert (UoR)</td>
<td>12 December 2014</td>
<td>Reduce redundancy/repetition from other deliverables, make sure you are using the same terms throughout, possibly introduce a glossary (e.g. make clear whether there is a difference between SM module and SM server – not clear at the moment)</td>
</tr>
<tr>
<td>Mads Troest (Jayway)</td>
<td>16 December 2014</td>
<td>Reviewed sections 2 and 8. Minor corrections needed.</td>
</tr>
</tbody>
</table>
Appendix 1: EPASS CLASS SKELETONS

(informal syntax: underline = (partial) key; { } = set)

Class Configuration:
- Name: string
- Specialties: {Specialty*}
- Formtypes: {Formtype*}

Competency:
- Within: Configuration
- Code: string
- Name: string

Subcompetency (subclass of Competency):
- Parent: Competency
- Code: string
- Name: string

Specialty:
- Within: Configuration
- Name: string

Themes: {Theme*}
- Forms: {Form*}
- Activitytypes: {Activitytype*}
- EPAs: {EPA*}

Theme:
- Within: specialty
- Name: string

Activity:
- Within: specialty
- Name: string
- Type: string

Portfolio:
- Within: Configuration
- Trainee: User
- Name: string
- Status: string
- Parts: {Subportfolio*}
- Submissions: {Submission*}
- Scores: {Competencyscore*}
- Activitylogs: {Activitylog*}
- Entrustabilities: {Entrustabilitystatus*}
- Themes: {Themestatus*}

Subportfolio (subclass of Portfolio):
- Partof: Portfolio
- Name: string

Formtype:
- Within: configuration
- Name: string
- Properties: {string*}

Form:
- Type: Formtype
- Within: specialty
- Name: string
- Version: string
- Items: {Formitem*}

Formitem:
- Within: Form
- Name: string
- Type: string

Submission:
- Within: Portfolio
- Form: Form
- Number: integer
- Round: integer (optional)
- Submissiondate: date
- changedate: date (optional)
- validationdate: date (optional)
- Submitter: user
- Status: string
- Answers: {Answer*}
- Attachment: {Attachment*}
- InTheme: Theme (optional)

Assessment (subclass of Submission):
- Assessor: user
- Validator: user
- Referencedate: date (when was the assessed activity)
- Curriculum sequence: string
- Videoannotations: {string*}
- Context: {string*}
- Formscores: {Competencyscore*}
- ForEPA: EPA (optional)

MultisourceFeedback (subclass of submission):
- External_assessors: {externaluser*}
- Selfscores: {Competencyscore*}

Answer:
- Within: Submission
- For: Formitem
- Value: string / integer (dependent on formitem type)

CompetencyScore:
- For: Competency
Value: integer/double

Attachment:
Within: Submission
Name: string
Filetype: string
Content: ?

Activitylog:
Within: portfolio
For: activity
At: date
Context: string
Level: integer (Miller level)
Assessed: assessment (optional)

Entrustabilitystatus:
Within: portfolio
For: activity
Level: integer (Miller level)
At: date
DecisionBy: user

Theme status:
Within: portfolio
For: theme
Status: Symbol
Dateassigned: date
Approvedby: user

EPA:
Within: specialty
Name: string
Intheme: Theme (optional)

User:
Within: configuration
Fullname: string
Organisation: string
Function: string
Appendix 2: EPASS API description

Authorization
The EPASS API allows for authentication using OAuth2. Gaining access is via the Authorization Code Grant flow. At the time that a client application wants to access a number of steps need to be followed:

In EPASS a one-time client_id and secret along with a redirect URL. This client_id and writing must also be recorded on the client side. For debugging API access, the client needs to redirect to using a web browser http://api.test.epass.eu/v1/authorize (GET method) with some query string parameters, for example:


* response_type = code: Indicates an Authorization Code Grant
* client_id = test client: The client_id issued ePass which can identify which client it is
* state = xyz: State parameter to protect against CSRF. This can eg a hash of the session cookie.

The EPASS user (resource owner) then logs in (still available via the web browser) and authorizes the client application for one of his / her portfolio. EPASS then redirects back to the client via the predefined URL along with an authorization code (valid for 30 seconds) in the query string:

   http://fakeurl/?code=87d2a47fe2b259d944911ead5ed97c25bccba175&state=xyz

Or error, if the user did not grant access:

   http://fakeurl/?error=access_denied&error_description=The+user+denied+access+to+your+application&state=xyz

Using this authorization code, the client then requests an access token from the http://api.test.epass.eu/v1/token endpoint (POST method).

The POST request to the token endpoint is as follows for the authorization code Grant:
A basic authorization header consisting client_id and writing, for example,

   Authorization Basic dGVzdGNsaWVudDp0ZXN0cGFzcw==
   post field "grant_type" with the value "authorization_code"
   post field "redirect_uri" with the value previously specified redirect uri: " http:// fakeurl / "
   post field "code" with the previously obtained authorization code "87d2a47fe2b259d944911ead5ed97c25bccba175"

The EPASS API then displays an access token and a refresh token back. Access Tokens have an expiration time of 5 hours:

   {"Access_token": "a2eb2e087d3e85a30dc1b3b0e417aa1bf3ea0f71", "expires_in": 18000, "token_type": "bearer", "scope": null, "refresh_token": "818d4999777e9ac94bdbd4abfd552d19962e3de6"}
After the expiration of an access token, a new access token can be retrieved with the refresh token via the http://api.test.epass.eu/v1/token endpoint. The POST request to the token endpoint is as follows for the refresh grant:

A basic authorization header consisting client_id and writing, for example,

```
Authorization Basic dGVzdGNsaWVudDp0ZXN0cGFzcw ==
```

post field "grant_type" with the value "refresh_token"
post field "redirect_uri" with the value previously specified redirect uri: " http: // fakeurl / "
post field "refresh_token" with the refresh previously obtained token "818d499977e9ac94bdbd4abfd552d19962e3de6" (valid for 14 days)

The ePass API then displays a new access token and a refresh token back:

```
{   
  "Access_token": "ed6b63d71c8a5ec789f486f827095141c55542dd",  
  "Expires_in": 18000,  
  "Token_type": "bearer",  
  "Scope": null,  
  "Refresh_token": "0160b28c1a62035c8eb5234c2efc936cdaefe21a"
}
```

If the refresh token also has expired, the end user must pass the authentication flow again.

**Resources endpoints**

**Example endpoint**

http://api.test.epass.eu/v1/@responseformat/@class(/@id)?access_token=ed6b63d71c8a5ec789f486f827095141c55542dd

responseformat: This can be JSON or JSONP
class: The resource which is invoked
id: Optional resource id
access_token: querystring parameter with access_token previously obtained
The access_token should be sent along with each function call as a query string parameter!

**Adding an upload:**

POST / v1 / @ response format / uploads
enctype: multipart / form-data
post field "file upload" the file data

**Example response (JSON):**

```json
{   "File upload": [     {         "Name": "test.docx",         "Size": 13755,         "Type": "application/vnd.openxmlformats-officedocument.wordprocessingml.document",         "Uploadid": "42",         "Error": "",         "Title": "",         "Summary": ""     }   ]}
```

**Updating an upload:**
POST / v1 / @ response format / uploads / @ id
encype: multipart / form-data
post field "file upload" the file data
query string param "id": uploadid in applicatio

Example response (JSON):
{
    "File upload": [
        {
            "Name": "update.docx",
            "Size": 14261,
            "Type": "application/vnd.openxmlformats-officedocument.wordprocessingml.document",
            "Uploadid": "42",
            "Error": "",
            "Title": "",
            "Summary": ""
        }
    ]
}

Retrieving all uploads:
GET / v1 / @ response format / uploads /

Example response (JSON):
[{
    "Uploadid": "42",
    "Title": "",
    "Summary": "",
    "Filename": "test.docx",
    "File size": "13755",
    "File type": "application/vnd.openxmlformats-officedocument.word"
    "Stored_at": "2014-08-21 12:56:58"
}]

Deleting an upload:
DELETE / v1 / @ response format / uploads / @ id
query string param "id": uploadid in applicatio

Example response (JSON):
true