České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra počítačové grafiky a interakce

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **Nikita Silin**

Studijní program: Softwarové technologie a management
Obor: Web a multimedia

Název tématu: **Webová aplikace pro správu databáze 3D modelů**

Pokyny pro vypracování:

Navrhněte a implementujte webovou aplikaci pro správu databáze 3D modelů s touto funkcionalitou:
- Nahrávání nových modelů a automatická kontrola jejich kompletnosti.
- Generování informačního souboru k modelu.
- Vytvoření náhledu modelu a uložení modelu do samostatného podadresáře.
- Nahrané modely budou nabízeny ke stažení po schválení cvičícími.
- Modely budou z databáze stahovány jako zip.
- Neschválené modely budou vráceny uživatelům k doplnění.
- Správa uživatelů.

Databáze bude doplněna o webové prezentační rozhraní s náhledy jednotlivých modelů a následující funkcionalitou:
- Galerie bude členěna na sekce, které bude možné přidávat/odebírat.
- Prohlížeč modelů ve WebGL umožní modely otáčet a měnit stínovací model.
- Galerie bude vytvářena online průchodem adresářů s modely.

Implementaci proveďte pomocí technologií PHP, XML a WebGL (bez použití databázového serveru). Funkčnost aplikace ověřte vložením alespoň 25 modelů z existující databáze předmětu A7B39PGR.

Seznam odborné literatury:

[1] Matt Zandstra: PHP 5 Objects, Patterns, and Practice. Apress, 1st edition, 2005, ISBN 978-15905938.
[2] Kouichi Matsuda, Rodger Lea: WebGL Programming Guide: Interactive 3D Graphics Programming with WebGL. Addison-Wesley Professional, 1st edition, 2013, ISBN 978-0321902924.

Vedoucí: Ing. Jaroslav Sloup

Platnost zadání: do konce letního semestru 2015/2016

L.S.

prof. Ing. Jiří Zára, CSc.
vedoucí katedry

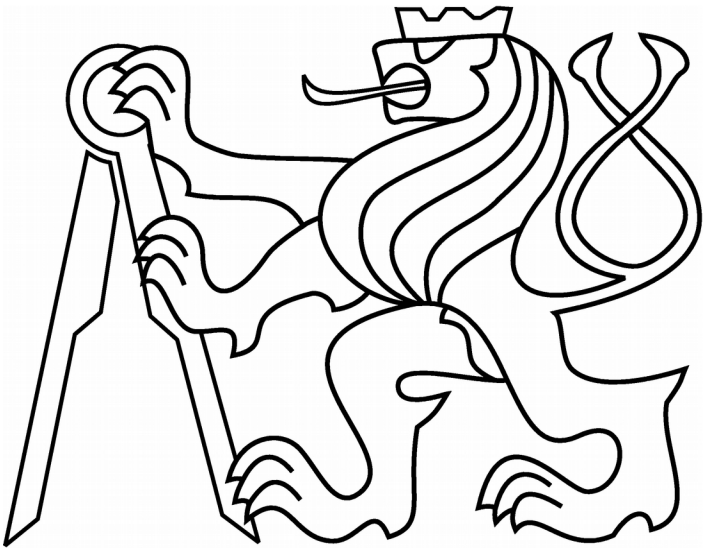
prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 24. 3. 2015

Czech Technical University in Prague

Faculty of Electrical Engineering

Department of Computer Graphics and Interaction



Web application for 3D models database management

Bachelor Thesis

Nikita Silin

Bc. programme: Software Technologies and Management

Branch of study: Web and Multimedia

Supervisor: Ing. Jaroslav Sloup

Prague, May 2015

Prohlašení

„Prohlašuji, že jsem předloženou práci vypracoval samotně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací."

V Praze dne podpis

Declaration

"I hereby declare I have written this bachelor thesis independently and quoted all the sources of information used in accordance with methodological instructions on ethical principles for writing an academic thesis."

In Prague signature

Abstract

The purpose of this thesis is to design and implement a web application for managing a database of 3D geometry. Following the methodology defined in this paper, architecture is created for the desired system using a set of models. Additionally, application is split into several layers. Obtained models and layers are then used to implement the system. Revealed problems are described along with their solutions. The technology set includes: PHP, C++ and WebGL. Documentation and user guide are then constructed for the system maintenance.

Abstrakt

Táto bakalářská práce se zabývá návrhem a implementací webové aplikace pro správu databází 3D modelů. Architektura systému která je definována jako sada modelů, vytvořena pomoci metodologie, popsane v této prací. Aplikace je rozdělena do několika vrstev. Výsledné modely a vrstvy jsou pak použité k implementací systému. Práce popisuje problémy které vznikají během vývoje spolu s jejích řešením. Projekt využívá moderní softwarové technologie, ke kterým patří PHP, C++ a WebGL. Součástí práci je uživatelský manual a dokumentace pro správu systému.

Illustration Index

Illustration 1: Use Case Model Diagram.....17
Illustration 2: Components Model.....26
Illustration 3: Unified Geometry Data Structure.....39
Illustration 4: Mapped Models.....44

Index of Tables

Table 1: Market Analysis Result.....16
Table 2: Asset Management Component Operations.....27
Table 3: Files Management Component Operations.....28
Table 4: User Management Component Operations.....29
Table 5: Asset Preview Component Operations.....31
Table 6: Asset Load Component Operations.....32
Table 7: Back-end Project Structure.....43
Table 8: Front-end Project Structure.....43
Table 9: System Performance Test.....50

Table of Contents

1 Introduction....1
2 Problem Description....3
3 Methodology....4
3.1 Architecture Design Stage....4
3.2 System implementation Stage....5
4 Architecture Design....7
4.1 Requirements elicitation....7
4.2 Market analysis....8
4.3 Use Case and Access Control Definition....11
4.4 Models and business domain....14
4.5 Back-end Architecture Design....16
4.6 Front-end Architecture Design....22
5 System Implementation....25
5.1 Physical and Runtime Environment Evaluation....25
5.2 Technologies Selection....26
5.3 Project structure....33
5.4 Models definition and mapping....34
5.5 Components implementation....35
6 Conclusion....40
6.1 Performance....40
6.2 Suggestions....41
7 Bibliography....42
Attachment A: Full list of system requirements....43

1 Introduction

As the popularity of 3D modeling tools grows, more and more people are getting involved into 3D design. Produced 3D models can be used in many industries and for many purposes: medical and physical simulations, movies with computer graphics, 3D cartoons, computer games and many others. It becomes important to analyze different 3D models, as well as manage them efficiently. Analysis is based on the validity of the given model combined with a subjective judgments of the final user. Validity analysis can be automated in many cases, but it also depends heavily on how the model is used.

The main objective of this project is to design and implement a system for managing geometry in a digital form. A lot of requirements were set for the system. User should be able to load his geometry into the system. Authorized users may then approve this geometry and, optionally, rate it. Other users then should be able to preview this geometry and download it, if they want to. System should also expose validation abilities, such as reporting any errors about the given geometry. The full set of requirements is listed later in this paper.

Digital 3D geometry may be represented in many formats, created by different authorities in the 3D modeling industry. Different formats may include additional data, such as animations, bones structures and scene properties. The key requirement for the desired system is an ability to convert different 3D geometry formats into a versatile minimalistic data structure, containing only essential data from the original geometry format. As a result, such data structure serves as a layer of abstraction for the other operations running in the system, such as validation and previewing.

Additional assets may be required to evaluate the geometry: textures and materials. Textures are used to visually cover the geometry and give it realistic feel and look. They can be represented as any digital picture format, such as PNG or TGA. The glue between texture and geometry is often called material. Materials are data structures, which hold information about textures and colors used for the current piece of geometry. With a large set of possible texture and material formats, system should be able to use those for geometry validation and visualization.

Meta information is another key requirement for the system. Such information may be useful for the end-user to effectively search and update geometry when needed. Geometry formats usually do not hold meta information. System should fill this gap and provide users with an ability to include additional information, such as description, original author, category, rating and others.

2 Problem Description

As an internal system for "CG Programming" subject a database was instantiated to allow students to upload their 3D models and let professors evaluate uploaded content. Implemented as a server with FTP / WebDAV access, database allows students to connect and upload or download ZIP archives. Such archives contain 3D geometry, as well as related textures and materials. Even though each asset is required to contain a preview image, it is hard to evaluate model at first glance. To check the validity of the model, moderator has to load it using Assimp Library[1]. This library is utilized inside of internal framework, which students use to create 3D scenes as a task within the mentioned subject. Hence, it is important to ensure, that any uploaded content is valid with respect to the mentioned library.

Since database is rapidly updated (students upload new models by the end of each semester), maintaining such database manually is quite inefficient and consumes a lot of human resources, because every uploaded asset requires manual validation. It is important to control completeness, correctness and uniqueness of the content, as well as its proper categorizing. As a result, it was suggested to design and implement a system to automate most of the described procedures and original conformed specification was constructed to describe key requirements.

1 More information about Assimp Library can be found at http://assimp.sourceforge.net/

3 Methodology

There are many approaches to the development process. Most of the time, development steps are defined within the selected methodology. *Software development methodology* is a separation of development process into several steps for better project planning and management. Each step describes a certain set of actions to be performed [1]. Popular methodologies include such titles as spiral development, agile development, rapid development and many others. Methodology is chosen or created based on many factors: application complexity, requirements, subjective experience and many others. This project uses very simple methodology, which is represented as 2 stages: architecture design and system implementation.

3.1 Architecture Design Stage

During this stage the system is described using abstract terms. The Unified Modeling Language (UML) is a modeling language used in software engineering. It is created to standardize the way of visualizing system design [2]. UML is used partially to describe certain pieces of the current system without any implementation details. Specification of this language defines several important model types, which are used to describe the desired system: Business Domain model, Components model, Requirements model and Use Case model. Each model represents current system from different perspective. For example, Use Case model defines system operations and Business Domain model defines entities which system operates on.

The following algorithm was chosen to design application architecture:

1. Requirements Elicitation
2. Market Analysis
3. Use-case definition
4. Access control definition
5. Entities design
6. Architecture layers design
7. Architecture components design

Requirements elicitation step is taken to define responsibilities of the system. Market analysis is then performed to find and evaluate existing systems with a similar functionality to decide, if a custom system development is reasonable. Requirements are then used to construct a set of use cases to be implemented within the system. Finally access control definition is created, which describes operation access rights for different user roles. At this point solid base for the system architecture development is formed. Requirements and use cases are analyzed to create entities and naturally derive operations. Operations are then grouped using components. Additional step is taken to define architecture layers to categorize the components.

Once architecture is designed, it is evaluated once again with respect to given requirements and use cases to find any problematic spots, which deviate from the original conformed specification. The result is then used to implement the system.

3.2 System implementation Stage

Software development methodologies usually break the entire development process into several stages, with their own iterations and steps. Stages sometime mix the design and implementation steps, but in this project those 2 processes are completely independent. The following algorithm is chosen for the system implementation:

1. Evaluate the target physical environment based on requirements
2. Evaluate the target runtime environment based on requirements
3. Filter available technologies based on requirements
4. Select the programming language, tools, formats and conventions
5. Define the project structure.
6. Define models and map those to classes
7. Map components to classes
8. Implement operations
9. Perform system test

The implementation begins with choosing the appropriate programming language, correct compiler or interpreter, useful libraries containing desired functionality, data persistent servers and so on. For simplicity, in this section, all of those are called “technologies” or “tools”. It is taken for

granted that at the beginning of implementation phase, all the technologies are available. The physical environment is then evaluated to filter out those technologies which are not available. The runtime environment is evaluated as well, with the same purpose. We then filter out those tools, that are restricted by the conformed specification or non-functional requirements.

Final set of technologies is then chosen, based on specific set of requirements defined by the programmer or the team leader. Additional technologies are chosen for the development purposes (IDE, VCS, Package Managers…), internal/external system formats and conventions are selected to be used within the system. The 5th step is taken to define the project structure. It is a very important step, as it has a direct influence on programmer's productivity. After all the preparation steps are complete, model data structures are implemented. Programmer then maps components to classes and implements operations. The final step is extensive system testing.

4 Architecture Design

This section describes the results of each step of the architecture design algorithm, defined in the previous section, as well as statements, explaining why certain step is important. A lot of design aspects are expressed using diagrams to simplify reader's perception.

4.1 Requirements elicitation

Requirements elicitation is an important step, especially at the early stage of the development. Correctly gathered requirements provide a solid base for the project management and help to keep the development process in the right direction. In the beginning, requirements are usually gathered from the design document or conformed specification. It is important to note that requirements are unstable and change heavily during the entire development process, as stage or iteration results are consulted with stakeholders. It is also helpful to assign priority for each requirement, but in this case priorities are omitted. Initial requirements for the current project were gathered from the original conformed specification. Functional requirements are grouped logically and represent the source, from which entities and operations are derived.

Non-functional requirements include such interesting items as:

- system is a PHP application with a web interface
- system should not use persistence technologies which require standalone server
- allowed server-side technologies are: PHP, C++, XML
- validation of the asset ensures that 3D geometry can be imported using Assimp Library

See attachment I for the full list of requirements and their implementation states.

4.2 Market analysis

With a set of requirements constructed, several system with the corresponding functionality were evaluated. No standalone systems were found with matching capabilities. That is why the closest segment was also analyzed: 3D model markets. Such systems provide functionality very similar to the given requirements. The following use cases and properties were used to compare systems:

- Standalone version
- Distribution type
- Streamlined asset uploading functionality
- Automatic validation functionality
- Rating functionality
- Preview functionality
- Approve / Decline procedure

Turbosquid

Being one of the most popular 3D model stores, Turbosquid is a web-based trading platform. Designers may upload their models, set up the optional price, add description, tags, etc. Uploaded content is validated manually. Users may then choose models from the catalog. Turbosquid exposes CheckMate™ Certification standard, which defines certain quality level for a given 3D asset. This is a commercial system and the source code is not public.

CGTrader

Another popular service for 3D models management. Like Turbosquid, this service is a 3D model marketplace for computer graphics and 3D printing with a web interface. The work flow is very similar to Turbosquid, however author was unable to spot any predefined standard for uploaded content. This is a commercial system and the source code is not public.

Unity Asset Store

Unity Asset Store serves mainly to deliver addons, plug-ins, scripts and content for the Unity3D game engine. Developers and designer may upload their content to the asset store and set up the optional price, category, tags and description. Loaded content is then manually validated with the respect to mentioned game engine. Other users may rate the content after they buy it. This is a commercial system and the source code is not public.

3DExport

3DExport is another web resource for 3d geometry trading. The workflow is quite usual for any marketplace. Additionally this market exposes textures as a separated product. All the assets are properly structured. Rating system is available, as well as different meta information for each asset. This is a commercial system and the source code is not public.

FTP Based File Database

A simple server exposes remote access for the group of users and allows them to upload and download content without any validation or control. The system moderator or other responsible person is supposed to control loaded assets. This is how current solution for the described problem is implemented. As it was mentioned earlier, such system is simple but inefficient and lacks most of the required features. This is a setup, which does not require any special software and can be deployed within any popular operating system.

Market Analysis Result

The final comparison table follows:

Title	Standalone Version	Distribution type	Allows asset uploading	Automatic validation	Allows Rating	Online Preview	Approve/Decline Procedure
Turbosquid	No	Commercial	Yes	No	Yes	No	Yes
CGTrader	No	Commercial	Yes	No	Yes	No	Yes
Unity Asset Store	No	Commercial	Yes	No	Yes	No	Yes
3DExport	No	Commercial	Yes	No	Yes	No	Yes
File Database	Yes	Free	Yes	No	No	No	No

Table 1: Market Analysis Result

In the context of the current project, the main drawbacks of all mentioned systems are:

- market concept
- no preview
- no standalone version for local deployment (with exception of File Database)
- a lot of functionality which is not required

4.3 Use Case and Access Control Definition

Given the set of requirements, an appropriate use case diagram was constructed. This diagram describes what are the system entry-points and operations from the user perspective. Each use case corresponds to a requirement or a group of requirements.

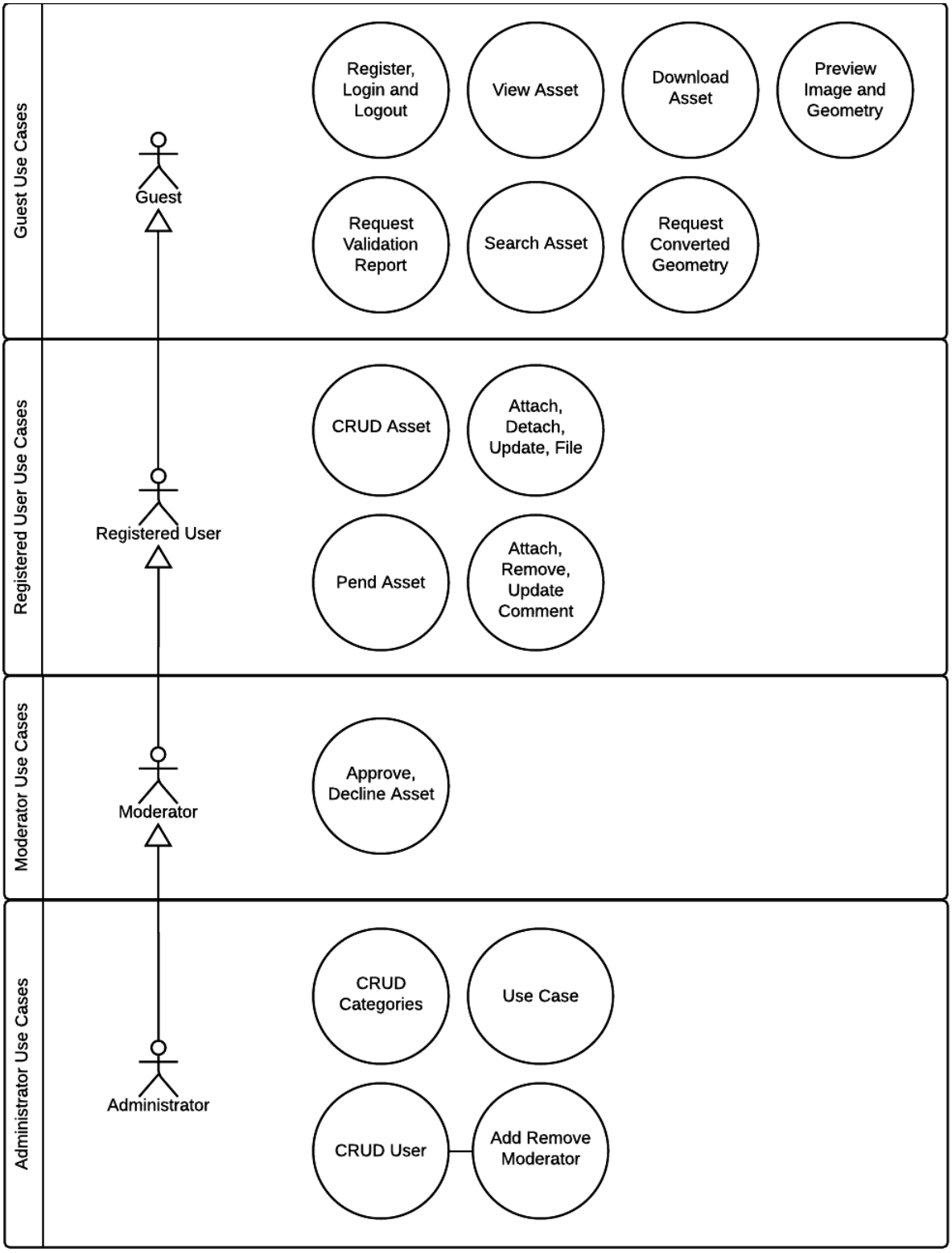


Illustration 1: Use Case Model Diagram

Specifically, this diagram reveals 3 important kinds of information: actors, system operations and unconditional access control. Actors represent different types of users. System operations, expressed as use cases, define the abstract system interface. As each use case is defined in the scope of a certain user role, basic access control rules can be easily derived. However, each access rule may have certain conditions. Those are described later in this paper. To make diagram succinct and easy to read, certain operations have been grouped and represented with a single use case. For example, "CRUD Asset" use case incorporates the following operations defined for the asset: Create, Read, Update, Delete.

Actors

System distinguishes between 4 user roles which are part of vertical hierarchy: Guest, User, Moderator and Administrator. Starting from the Guest, each new role is given access to additional functionality.

Guest

Guest is considered as unauthorized user of the system. Without any login information, such user may only browse certain information within the system and has no rights to change any content. The only available option for the Guest is Registration within the system.

User

User is a regular authorized user which has account information registered in the system. Such user is granted with rights to upload and change his own content.

Moderator

Moderator incorporates all of the guest and user rights but such user also has an ability to change the accept or decline any content which is pended for review.

Administrator

Administrator has all the rights available in the system. Such user has an ability to manage categories and users.

Use Cases

Use Cases define system operations from a user perspective. Those are usually derived naturally from the requirements definition. Requirements list contains several groups, such as:

- Asset Management
- Categories Management
- Files Management
- User Management
- Comments Management

As each use case represents certain requirement, it is implicitly related to the corresponding group.

Conditional Operation Access Control

The following rules describe access control conditions which cannot not be derived directly using use case model:

1. User may only log out if he is logged in.
2. Guest may only register or log in if he is not logged in.
3. User may only access, comment, manage files, preview, search, validate, convert and download assets which are created by this user or are approved.
4. User may only modify and remove assets, created by this user
5. Moderator may access, comment, manage files, preview, search, validate, convert and download any asset.

4.4 Models and business domain

Defining business domain is crucial for planning the development process. Not only domain serves as a blueprint for application models, it also helps to understand how application works. Business domain consists of entities. Each entity represents some data structure, which application operates on. Such data structures usually appear naturally based on requirements and hold logically connected data. A good example could be User Entity: data structure which hold user specific information: user name, email, password and so on. Entities serve as blueprints for models. The difference between models and entities is that entity describes the idea (some abstract concept), but model uses this idea and describes specific implementation details, like primary and foreign keys for the database and utility data-structures.

Model may differ from the corresponding entity as sometimes is needed to tweak or strip the model to avoid major problems during the implementation phase. Moreover, models may contain additional behavior. Indeed, models may hold business logic, but in this project author is trying to avoid this as it couples models with the current application [3]. Several important entities were defined for this system as follows.

Asset Entity

Asset entity represents a package which user loads into the system. Such package has some primitive data such as title, description, rating, author and so on. Asset item is connected to a specific category, asset files and asset comments. The following important data items are associated with this entity:

- Title
- Description
- Tags
- Rating
- State (Pending, Accepted, Declined)
- Original author

This entity also contains additional information such as the date of last modification, the date of upload, uploader user name and so on.

Asset File

Asset File entity serves to bind arbitrary data to an asset. Arbitrary data means some content, which is usually represented as file. Mentioned entity holds file content, as well as many other important data items: user name of the uploader, name of the file, path to the file and so on. This entity is connected with a parent asset. Main data fields defined for this entity are:

- Name
- Path
- Contents

Asset Category and Asset Comment

Asset Category entity is introduced due to the original requirement about categories management. As a result, categories are granted their own entity, which holds category title and optional description. Asset Comment is another minimalistic entity to hold messages bound to a specific author and asset.

User

User entity is quite light-weight. However, it is one of the most important entities, as it defines general information about the person registered in the system: email, user name, password. This entity is also used heavily for access control.

4.5 Back-end Architecture Design

Architecture Layers

Given a set of requirements, use cases and entities, initial high-level design was created. The application is separated into several layers:

- Data access layer
- Service layer
- Controller layer
- View layer

First 3 layers are grouped into back-end part of the application

Data Access Layer

Entity instance, which exists withing the system, should be accessed using Data Access Layer. To represent this layer, repository pattern was chosen as a battle-tested, recommended data-access solution. Repository is usually located between the domain and data access layers and exposes a collection-like interface to access model instances using criteria queries [4].

Service Layer

This layer serves to define an application's boundary using service components. Such components establish a set of available operations and defines the application's response in each operation [5]. Services are used in this project to hold operations logic. It is important to note, that some services are used for very specific purposes. For example “Files Management Service” holds the logic to save and load files, but under the hood it uses “Filesystem” service to abstract the file system.

Event Aggregator

Event Aggregator serves as a glue between Controllers and Service. This pattern combines events from different objects into a single or multiple channels to simplify registration for clients [6]. Such architecture decision was made with intent to simplify the implementation of minor side-effect operations. An example of such operation could be logging. This operation is may be performed when a certain system operation is performed. System just signals corresponding event and related components, like Logger Component, act accordingly.

Controllers Layer

Controllers are part of MVC pattern, which serves to split user interface integration into 3 distinct roles [7]: Model, View and Controller. Controller is responsible for handling signals from the view and dispatching those. In this application controller serves as a bridge between View and Services. it uses input data which comes from the view to invoke system operation. Some operations, like reading a model, are lightweight and do not require any model changes. In this case controller may directly refer to desired component and retrieve some information. But most of the time controller uses input data to compose, so called, command. Command is a data object which holds input for a particular system operation. After composing command, controller uses Event Aggregator to publish it. Different parts of the system may react to this command, since command is just an event. Command may also hold result, a property which is populated by one of the components. Controller may use result and send it back to the user.

View Layer

Views are also part of MVC. Representing user interface, they serve to accept signals generated by user and sending those to the appropriate controller. As a result, View is considered to be a front-end. Several front-ends may be implemented to communicate with the system. In fact, any application may use HTTP to communicate with the back-end and, so, represent another implementation of the View layer. Any third-party front-end program is beyond the scope of this project. However web-interface was designed due to the corresponding non-functional requirement. This web-based front-end is described later in this paper.

Components Model

Previous section reveals major application layers, but it does not define how the application is structured from components perspective. Component can be defined as a modular part of a system, that encapsulates its content and whose implementation is replaceable. A component defines its behavior by exposing provided and required interfaces [8]. Interfaces then define data-contracts for different operations. During the development stage each component is turned into a module or a set of modules. Each module is a cohesive unit designed to solve a specific task.
In object-oriented programming, module is just a class. Such layout helps to control cohesion and coupling, which is crucial for application maintainability, flexibility and extensibility.

The components diagram was created to define major components and their interfaces for the current project. In this project components often correspond to a specific use case group. For example, “Asset Management” Component exposes interface for different asset operations. This component incorporates other components such as Geometry Converter. Notice also that usually interfaces and components are separated. However, in the context each component defines interface implicitly.

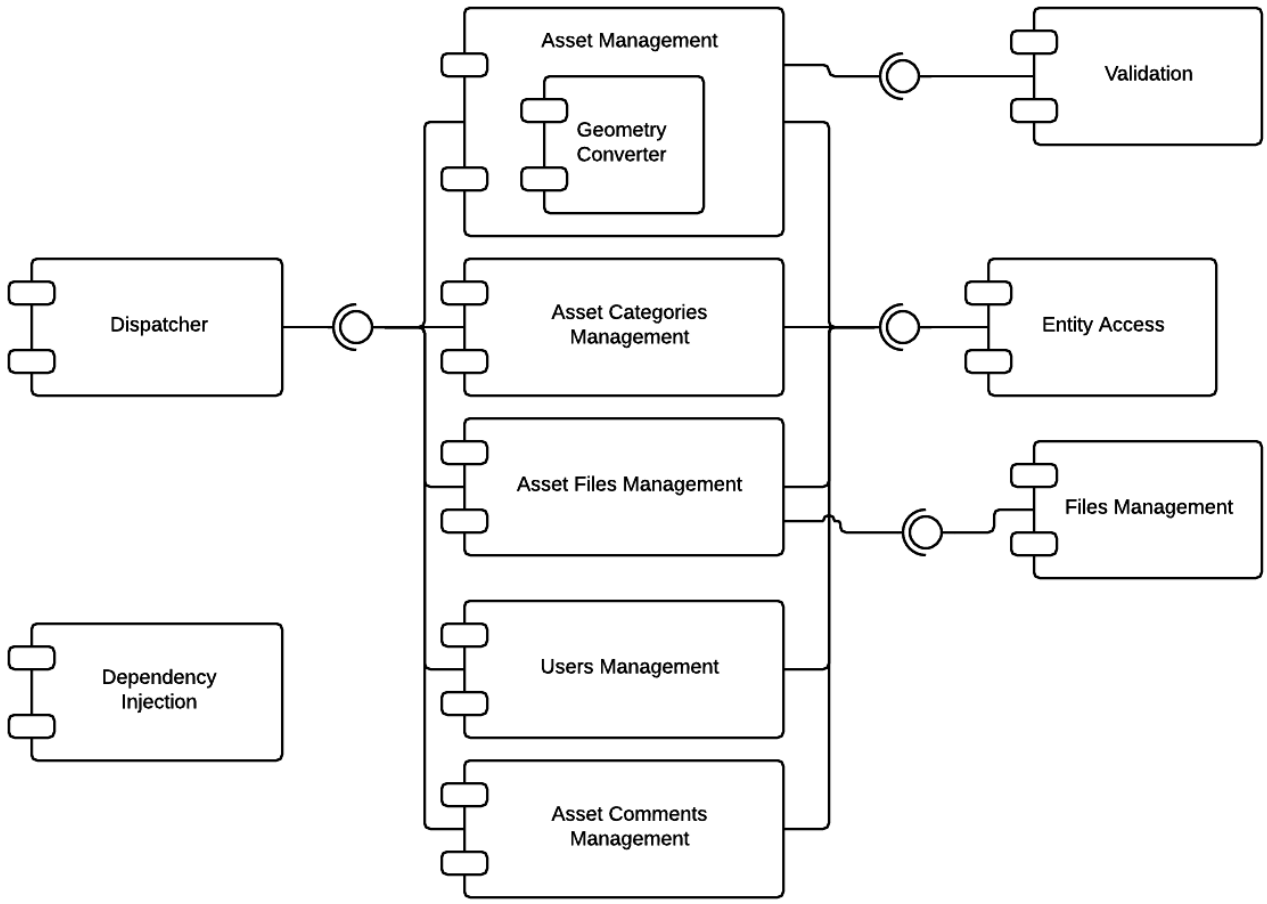


Illustration 2: Components Model

Component operations are derived from requirements and use cases. Such operations are combined under single interface to define responsibilities for the given component. Some components are strictly bound to a particular architecture layer, while others stand aside and expose handy functionality. As an example, Files Management component exposes interface to control physical files in abstracted manner. This functionality may be used by service layer to invoke file operations. Similarly, “Mesh Converter” component exposes specific operation to turn arbitrary 3D data into a versatile data format.

As it was mentioned in the previous section, the project is developed so, that many views could be used for the single back-end. That is why View component is not strictly defined. However, web-interface solution was designed to serve as a basic view implementation. It's architecture is described on the corresponding diagram later in this paper.

Asset Management Component

This component exposes operations defined for the asset entity.

Operation	Description	Input	Output
Create	Creates new asset	Asset data	Instance of asset
Read	Retrieves asset information using data access layer	Asset key	Instance of asset
Update	Updates the existing asset	Asset data to be modified	Instance of asset
Delete	Removes asset from the system	Asset key	Instance of asset
Validate	Validates asset	Asset + Context	List of validation messages

Table 2: Asset Management Component Operations

Asset validation logic is partially delegated to Validation Component. This operation also requires context to be available. The context of the asset is defined as files associated with this asset.

Asset Categories and Asset Comments Management Components

Those components provide solutions for categories and comments management and exposes primitive CRUD operations for each entity, very similar to those, defined for the Asset Management component.

Files Management Component

Operations defined for the asset file entity are combined under this component.

Operation	Description	Input	Output
Save	Physically save file content using path	File Object with defined path and content	Instance of file
Fetch	Retrieves file content using path	File Object with a defined path	Instance of file

Table 3: Files Management Component Operations

Geometry Converter Component

This component exposes a single operation – Convert. This operation transforms geometry asset file into a versatile data structure which can be later used to perform validation and analysis.

Entity Access Component

This component incorporates data access layer operations defined for different entities. Any CRUD operation is delegated to this component at some point.

Entity Transform Component

This component serves to transform different entities into different formats based on the given environment. Currently there are 2 operations available: Map and Transform. Map operation serves to convert external data into environment-compatible format, so that application can use the result as a native object. Transform operation does the opposite thing: given a native object, it maps it to a data format compatible with external services. Those operations are described later, in the implementation section.

Users Management Component

This component defines operations related to users and access rights:

Operation	Description	Input	Output
Log in	Signs user in	User credentials	-
Log out	Signs user out	-	-
Register	Signs user up	New user data	User entity

Table 4: User Management Component Operations

Additionally this component is used to expose currently logged in user object, on which any access control operation may rely. CRUD operations for the user entity are also part of this component.

Dispatcher Component

This component defines back-end access points. Being a web application, system is designed to receive request, perform certain operation and send the appropriate response. Hence, some component is responsible for mapping request to an operation. In web applications such component is usually represented with a router and a set of controllers.

Validation Component

This component is responsible for entity validation operations. By default, system performs minimal validation using "Validate" operation. However additional operations may be exposed to validate certain pieces of data.

Dependency injection

Dependency Injection Principle and DI Container were chosen as a component dependencies resolving solution. Acting as a glue for the entire application on components level, dependency injection container holds different solutions for different interfaces and exposes those for any interested instances. As an example, "Asset Files Management" component requires Files Management interface, and DI Container is the one, responsible for finding the appropriate component which will serve as a solution. As a result, the entire application is configurable, because every component is reusable and can be exchanged at any point.

4.6 Front-end Architecture Design

Web-interface is developed to fulfill the requirements and is designed as a separated web application. Having it's own layers and components this application follows the SPA idea. SPA stands for "single-page application". Such application usually fits on a single web page and delivers fluid user experience, very similar to desktop applications [9]. In this application, most front-end operations are delegated to the back-end service described in the previous sections. However, preview functionality is only available in the scope of front-end application, making it essential part of the current project.

Several approaches to front-end architecture development exist. Most of them are based on architectural patterns such as MVC or MVVM. MVVM stands for "Model-View-ViewModel", a pattern which defines separation of the GUI and business logic development using Model, ViewModel, View and Binder. MVVM uses Binder to automate the synchronization of data and it's view representation. As a difference from MVC, MVVM pattern uses ViewModel device instead of Controller. ViewModel may be considered as a view oriented controller. It is quite useful when the application does not require complex service layer; Since most of operations are delegated to the back-end, we only need light-weight components to expose view logic, mapping functionality and back-end connection. Hence, this pattern allows to keep the application lightweight.

Architecture layers

With a clear back-end separation and responsive bindings, MVVM becomes an obvious choice for the current application. Architecture layers are then derived naturally from the pattern specification.

View Layer

This layer should only serve to represent data and operation controls. No business logic should be defined within this layer. However light-weight validation logic is acceptable. In this application, View layer is usually represented with data representation templates. This is where the binding concept comes in. As front-end receives some model instance from the back-end, it binds it to a specific template, allowing this view to represent concrete data.

ViewModel Layer

To bind some data to a view, an appropriate ViewModel is used. ViewModel layer serves two purposes: data container and signal handler. When user activates certain controls on the view, signal is sent to the currently bound ViewModel. At this point ViewModel may update the data, execute request to the back-end or even transfer control to another ViewModel.

Back-end Access Layer

View Model layer manages the front-end application flow. However architecture still needs a component to abstract the back-end communication. Back-end Access layer holds such components and is responsible for making requests to the back-end application.

Components Model

Most of the front-end components are copies of their back-end counterparts. Exposing the same operations, however, they are part of another application, hence, implemented differently: all of them delegate operations to the back-end part of the system. Those will not be described here. However there are several components which belong specifically to the front-end.

Asset Preview Component

This components serves to fulfill the corresponding requirement. Main goal of this component is to expose operations for previewing different parts of the asset. Currently it consists of minor components to preview images and 3d geometry. The following operations are defined for this component:

Operation	Description	Input	Output
Open 3D Geometry	Shows 3D geometry preview window	Geometry file id and asset id	-
Open Image	Shows image preview window	Image file id	-

Table 5: Asset Preview Component Operations

Asset Load Component

To effectively preview any asset resource, web-interface should be able to load it. Asset Load component serves to provide this functionality. It is important to note that it is hardly possible to support all the formats and files ever created by the IT industry. That is why system only supports certain formats of 3D geometry, materials and textures. Exposed operations are:

Operation	Description	Input	Output
Load 3D Geometry	Loads 3D geometry from server converted to unified geometry data structure	Geometry file	-
Load Image	Loads image from server and converts it to be used by geometry preview component	Image file id	-

Table 6: Asset Load Component Operations

Router Component

Even though the application is constructed as a single page, it is designed to present many types of data and expose a lot of controls. Page composition is usually defined by the ViewModels which are currently bound to their views. To efficiently bind, unbind, show and hide different pieces of data, application needs a dedicated component for this purpose. Router component is designed to expose certain parts of the system based on a given route. Such an approach makes front-end routing very similar to the back-end dispatcher component. However front-end does not have controller, so only Router is defined.

Notification Component

As it was mentioned, front-end application is designed to provide native look and feel of desktop applications. Hence, it is important to define a separated component exposing operations for user notification.

5 System Implementation

At this point abstract architecture for the system is designed using UML models. This section is dedicated to the implementation process and problem solving. As it was mentioned earlier, several predefined steps will be taken to achieve working instance of the system.

5.1 Physical and Runtime Environment Evaluation

During this stage conformed specification along with the other non-functional requirements will be used to identify target environment. This information is used later to define the set of selected technologies and libraries.

Hardware

Conformed specification does not define any hardware restrictions. Neither does any of non-functional requirements. This means that minimal hardware requirements should be derived according to the target operating system and set of selected technologies and software components.

Operating system

According to the non-functional requirements, the system will run inside Unix-like environment. A lot of operating systems are considered to be Unix-like. The most popular systems are Mac OSX and Linux. As a result, platform-dependent components should be compiled for each platform independently, which is important, since this has a direct effect on how those components will be implemented. On a side note, development process is performed on Windows 8.1 operating system.

Front-end Runtime

World Wide Web technologies are designed to be platform independent. Indeed, a lot of systems have different browsers, but in the end they all represent the same content defined by the loaded web site. Non-functional requirements state, that the desired system will be accessed using Web-Interface. However, conformed specification does not define any browser requirements. This means that the minimal target browser should support the set of selected front-end technologies and frameworks.

Technology Restrictions

As conformed specification states, system is not allowed to use any database technology requiring standalone server. Also, allowed server-side languages are predefined: PHP and C++. This is a very wise choice, since PHP is a platform independent language which can be applied to solve most of the problems. C++ then serves as a complement and fulfills any platform-dependent gaps.

5.2 Technologies Selection

In this section technologies are selected to serve as components for the system. It is important to note that any use-case or component defined during the architecture design state could be implemented manually. However a lot of corresponding problems are beyond the scope of this project. That is why third-party open-source technologies will be used to replace corresponding components of the system and speed up the development.

Back-end Server Suggestions

In this context, server is an application which is capable of serving low-level HTTP requests. The most simple server is a program which accepts HTTP request and sends a valid HTTP response back to the remote machine. However, this project requires a server capable of processing PHP files. There are several popular servers which support this functionality:

- Apache HTTP Server
- nginx
- lighthttpd

Apache HTTP Server

Created back in 1996, this server remains the most popular web-server software. “In 2009, Apache was estimated to serve 54.2% of all active website and 53.3% of the top servers across all domains” [11]. It offers battle-tested technologies, flexible settings and hundreds of extensions. Apache is usually used with Unix-like systems.

nginx

Nginx is an open source reverse proxy server. It supports such technologies as HTTP(S), SMTP, POP3 and others. According to a web server survey taken in 2015 [12], nginx is the second most popular web-server solution.

lighthttpd

Lighthttpd is another popular web-server which is optimized for speed-critical tasks and low-resources environments. It is important to note that it runs natively on Unix-like operating systems. One of the key features of this web-server is support for “WebDNA” in-memory database technology. Such database does not require any additional server and could be used for the current project. However, in-memory persistence is unreliable. Moreover, such an implementation decision would lead to a strong coupling between web-server and the system, which is unacceptable.

Any of the described servers could be used to deploy this system. However, web-server is not part of the system and the final decision should be made by the system administrator or any other responsible person during the deployment stage.

Back-end Frameworks and Libraries

It is a common practice to integrate ready-to-use third party frameworks which solve boilerplate tasks for the current application type. In the scope of this project, web-framework needs to be selected. Such framework should expose implementation for components defined during the architecture design. Those include: Dependency Injection Container, Controller base, Model Base, Data Access layer, Router and more.

Almost any modern PHP framework implements common components. Hence, the decision was made with respect to performance and usability. According to the benchmark performed by Lukasz Kujawa in 2013 [10], Phalcon PHP Framework is the leader in terms of performance. Distributed as a PHP extension, this framework offers supreme performance, as well as modular structure, which can be adjusted based on current architecture design. Along with all the requested features, Phalcon also exposes several other components useful for the current application such as Event Manager.

At this point it is clear, that application will be mainly using 2 data formats: native PHP objects for the back-end and JSON data for the front-end. As it was defined during the architecture design stage, system should have Entity Transform component. Phalcon does not have any native tools to automate the conversion process. However other libraries may be used to fulfill this gap as almost any third-party component can be easily integrated into Phalcon environment.

Fractal[2] library was chosen to serve as a converter of PHP objects into JSON format. This library streamlines the process by introducing so called "transformers". Each transformer is tightly bound to a specific entity and defines how this entity should be transformed. Specifically it defines how each data field of the given object is transformed into a value, compatible with JSON format.

Another library, called JsonMapper[3], was selected to convert JSON data back to PHP objects. This library fully automates this process using naming convention: each JSON field has a name, which can be used to find the appropriate field of a blank PHP object and assign JSON value to it.

2 More information about Fractal library can be found at http://fractal.thephpleague.com
3 Source code for JsonMapper can be found: [13]

The missing piece is the File Management Component. This one is represented as file system object which is implemented inside of Gaufrette[4] Library. The responsibility of this library is to create a layer of abstraction for physical file operations. Gaufrette exposes single interface for the file system object and several implementation for different data storage types: local, Dropbox, Amazon Cloud and more. This application only operates in the scope of the local file system.

Database solution

As conform specification states, dedicated server database solutions are restricted. That is why application needs a server-less database technology. The only sensible solution is the implementation of data access adapter compatible with SQLite. It is an embedded ACID-compliant relational database management system. It implements most of the SQL specification and uses weakly typed SQL syntax. Hence, it does not guarantee domain integrity [14]. Lack of domain integrity control is a serious drawback for a complex application with a sophisticated model structure. However, this application only exposes 5 entities. It is important to note that Phalcon Framework exposes SQLite adapter for ORM component.

Geometry Unification

The desired system contains several requirements, which involve geometry manipulation. The most important of those are validation and visualization. As a result, system infrastructure needs a component to import different geometry formats and convert them into a unified data structure. As conformed specification states, uploaded geometry should be compatible with Assimp, a geometry import library written in C++.

4 More information about Gaufrette library be found at https://github.com/KnpLabs/Gaufrette

To solve geometry import problem, a corresponding software component was implemented using C++ and Assimp library. As a base, existing open-source code was taken from Lighthouse 3D demos and tools[5]. Originally this tool uses Assimp library to import 3D model from file and then streams out certain JSON to another file. The following modifications were made to the original code:

- Spaces and new lines are removed from the output.
- Texture is exported as file name without any path. If path exists, texture is complemented with texture reference. Such reference contains path to the file combined with the file name. The path is escaped to be valid against JSON specification.
- Application streams JSON or error messages directly to the system standard output.
- Bounding box calculation procedure was removed

Unified Geometry Data Structure

The following diagram describes unified geometry data structure. As it is shown, "model" object holds an array of meshes. Each mesh is represented with faces, positions, texture coordinates. Mesh may also expose material. Material is defined with colors and optional textures.

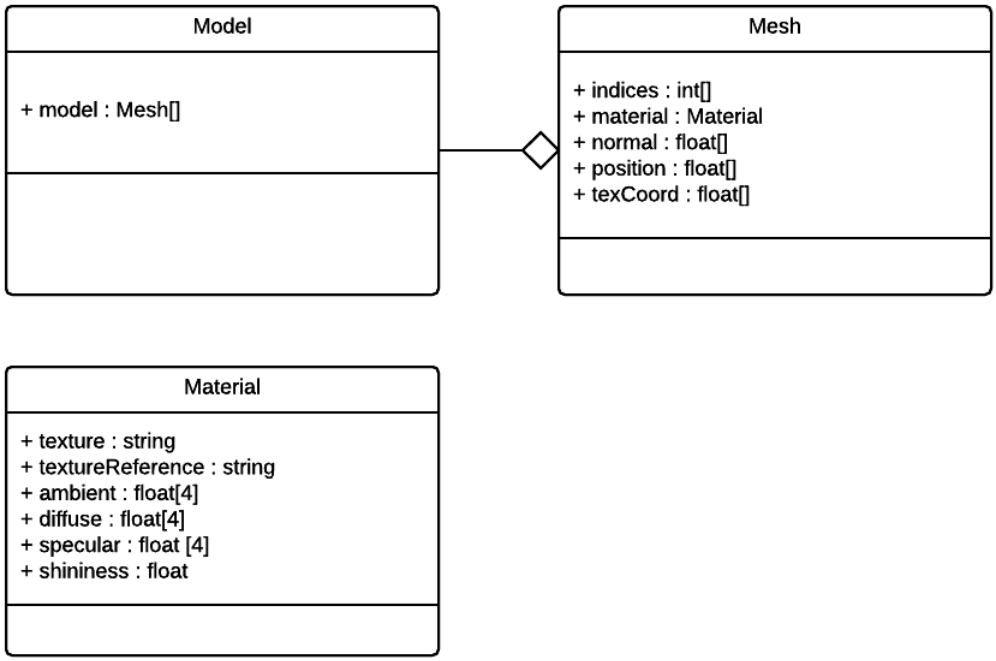


Illustration 3: Unified Geometry Data Structure

5 Original source code can be found: [15]

Front-end Frameworks and Libraries

As is was defined during front-end architecture design process, web application relies on MVVM pattern. A lot of client web frameworks use MVVM and expose all predefined layers and components. As a personal preference, author has chosen DurandalJS[6] framework. Being a very flexible library, it supports such patterns as MVVM, MVC or MVP. It also encourages SPA design and provides commonly used components, such as routers, communication modules, events, messages and so on. However, the MVVM core of DurandalJS is represented with another library called KnockoutJS[7]. This library serves as a Binder for MVVM setup and exposes observable variables, as well as HTML data binding mechanism.

The initial geometry preview functionality was implemented manually using WebGL and it's low-level API. However, such implementation was inefficient, slow and hard to maintain and change. The next version of preview functionality was using primitive object-oriented abstraction for WebGL API. Such implementation proved to be maintainable enough. However, key features were missing, such as texture import. Final version of geometry preview uses professional open-source object-oriented WebGL API abstraction called ThreeJS[8]. This library supports texture loaders, as an abstraction layer, to convert certain image formats to thee internal ThreeJS texture format.

To encourage modern, minimalistic and responsive design, application uses Twitter Bootstrap[9] framework. This framework exposes organized cascading style sheets to support advanced HTML layout and styling.

6 More information about DurandalJS can be found at http://durandaljs.com
7 More information about KnockoutJS can be found at http://knockoutjs.com
8 More information about ThreeJS can be found at http://threejs.org
9 More information about Twitter Bootstrap can be found at http://getbootstrap.com

Tools

While developing any application it is important to use reliable and efficient tools. Right tools streamline and accelerate the process dramatically. This project uses a set of tools for boilerplate steps automation, code navigation, components deployment and etc.

Integrated Development Environment

Integrated Development Environment is usually selected based on subjective experience of the developer. This tool usually provides the developer with an enhanced code editor and "Intelli-sence" support, which exposes whole set of features to help the developer navigate through the code. This project is created with Intellij Idea IDE created by JetBrains. Microsoft Visual Studio was used to implement C++ components.

Version Control System

Another important tool for any project is Version Control System. This one serves as a database of all the changes ever made to this project. Not only it ensures code safety, but also helps to spot changes, which caused a particular problem. This project was created using Git[10]. Being a distributed system, git supports several repositories with a single remote central repository.

Package Managers

A package manager is a software tool that automates installing and maintaining of software packages. With respect to software engineering, package manager usually serves to control different third party components used within the developed system. In this project 2 package managers. Bower[11] package manager is responsible for installing and configuring front-end JavaScript and CSS components. Composer[12] package manager, implemented as PHP script, takes care of third party PHP libraries and frameworks.

10 More information about Git can be found at http://git-scm.com/
11 More information about Bower can be found at http://bower.io/
12 More information about Composer can be found at https://getcomposer.org/

Formats

As it was already mentioned, application is using 2 data formats: native PHP object and JSON. JSON (JavaScript Object Notation) is an open standard format. It defines data structure which consists of human-readable data objects expressed using key-value pairs. It is used usually to transmit data between back-end and web application.

5.3 Project structure

Project structure is a crucial property of the development process. It defines where certain files, packages and components. Hence, it is recommended to keep project structure logically ordered. This dramatically helps any person involved into the development.
For the current system the following project structure was created:

Folder	**Content**
/app	holds back-end application scripts, libraries, etc.
/db	holds database files used by SQLite
/filesdb	holds uploaded content
/public	holds the front-end application and entry point for the back-end api
/tmp	holds any temporary files
/vendor	holds third-party back-end libraries managed with “Composer”

Table 7: Back-end Project Structure

Folder	**Content**
/app	holds front-end scripts (back-end connectors, viewmodels and views)
/cache	holds cached files
/css	holds styles
/fonts	holds fonts
/img	holds images
/js	holds js files not managed with “Bower”
/lib	holds any js libraries managed with “Bower”

Table 8: Front-end Project Structure

5.4 Models definition and mapping

According to the selected methodology, the next step is mapping entities to models and then implementing those as corresponding model classes. As it was mentioned earlier, models will be implemented using Model Base exposed by Phalcon Framework. Created classes are automatically compatible with the Phalcon Data Access Layer and can be persisted using SQLite PDO Adapter.

Entities mapping

Mapping is defined using the following diagram. Entities names and data fields are taken as a base. We then add additional system fields to support references between different models. We also define type for each data field of the entity. We finally define multiplicity for each reference. The result items look very similar to the original entities. However, important implementation details are defined.

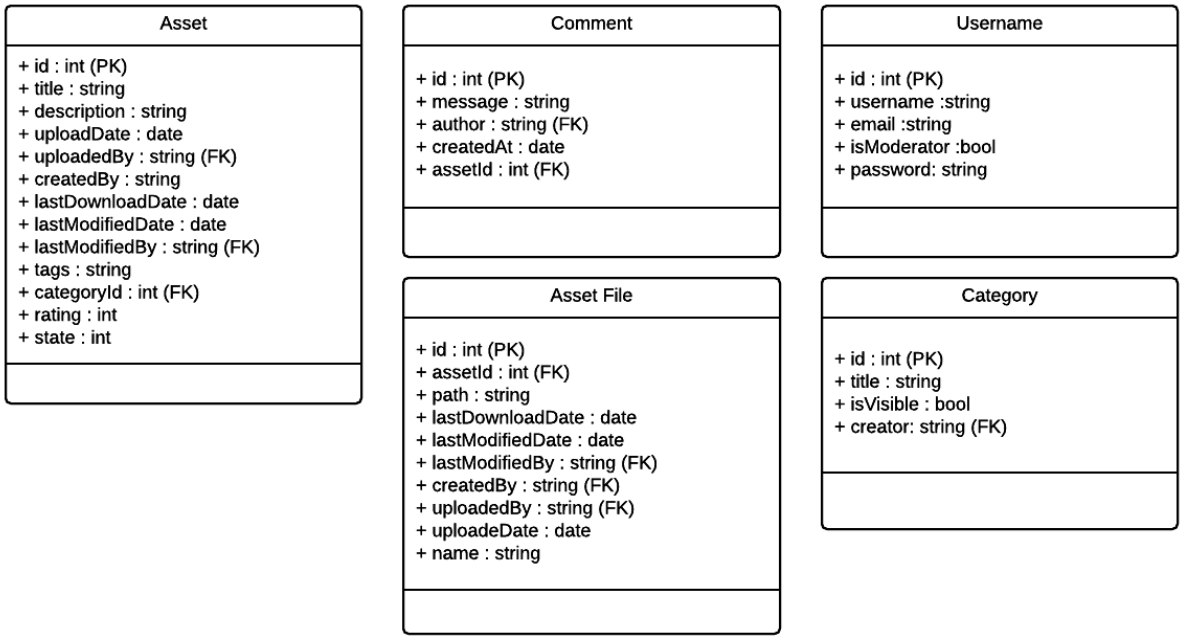


Illustration 4: Mapped Models

Models Implementation

Using models, we create classes and fill those with relevant content. We first define the class and derive it from the Phalcon Model Base. Then we add protected members for the sake of encapsulation. Each member represents data field defined within the current model. The next step is exposing public getters and setters. We then define and implement "column mapping" method. This method returns PHP array, which maps each member to an appropriate database column. Notice, that model may expose additional members and functions. Notice also that Phalcon Model Base supports life-cycle events. This means that developer may define a method called "beforeSave" which is invoked right before model is persisted.

5.5 Components implementation

Components implementation is the most important step, and the most sophisticated one. As a result, it is quite hard to describe every single implementation detail. More information about how each component is implemented can be found in the documentation. This section will highlight the most important details.

Transformers

Each transformer is represented as class, derived from AbstractTransformer class defined in the Fractal library. Transformer usually exposes one single method "transform". This method receives and an entity instance and returns a map. This map describes the result JSON: key of the map corresponds to the key in the result JSON. The value is then converted to a number or a string and output to JSON under corresponding name.

Repositories

Each repository is an abstraction over Phalcon Model which contains boilerplate methods to get entity by id or any other criteria. Repositories may be used inside services or directly in the controller, if the operation does not require any model modification. Search operation may serve as an example.

Commands

According to the architecture design, command class is defined for each operation. Such class contains input data for the corresponding operation, as well as any other data or methods. For example, “Validate Asset” operation is granted with it’s own command. This command references the asset to be validated, along with “addMessage” method, which allows any validator to add any error messages in case of validation failure.

Phalcon Events Manager may be used to publish such command every time. The handlers are registered using naming convention. Consider the following class “CreateAssetCommand”. This class, obviously, hold input data for the new asset. Once the command is published, the following methods are invoked on all the listeners:

- ValidateCreateAssetCommand
- ExecuteCreateAssetCommand

The first method is invoked on all the interested instances to analyze the input data and throw exception if data is invalid. If any exception was thrown, the next method is not invoked. If data is valid, however, any service may process the command and do something useful. Besides creating the asset itself, another service may notify relevant moderators or log the event for the system.

Controllers

Phalcon exposes both Router component and Controller base. The routing mechanism uses annotation to map each HTTP request to a specific discussion and will not be discussed in this paper. However, it is important to highlight how controllers are structured. Since application is trying to follow REST idea, each entity has a corresponding controller. When the request comes every controller is trying to follow the following plan:

1. If current operation does not require any model changes (READ operation)
    1. Control access rights
    2. Use data access layer to retrieve the model instance
    3. Map instance to JSON using the corresponding transformer
    4. Respond with some data
2. Else
    1. Map request body to the corresponding command using JsonMapper library.
    2. Assign any additional variable needed for the command.
    3. Publish the command.
    4. Optionally, map the result using appropriate transformer.
    5. Respond with some data

In the second case, access control is delegated to the command handlers. In case of an error, controller falls back and responds with an appropriate error message and a corresponding HTTP code. Since every controller is trying to follow this plan, it becomes a minimalistic cohesive unit with a very precise functionality.

Services

Services are the heart of the system. Each service is bound to solve a specific use-case or expose internal operations to be used within other services. Use case services are represented as classes and named conventionally, according to the entity:

- AssetsService
- AssetFilesService
- CategoriesService
- CommentsService
- UsersService

Each use case service also represents some management component defined during the architecture design. Hence, it exposes all the operations defined within the component. Application has several other services implemented. Some of them represented very specific application operations.

Geometry Converter Implementation

Original “Convert” operation belongs to the asset management component. However, it is important to keep high cohesion within a single service. That is why current functionality is delegated to another service called “MeshConverterService”. Conversion process is defined as follows.

Mesh converter service receives input, containing reference to geometry asset file with a defined path. Service then uses system shell and invokes platform-dependent software component with a path to the geometry file specified. Mentioned utility then streams out either JSON data or an error, which appeared during the conversion process. Finally, mesh conversion service collects the result and exposes it as a corresponding command result. Relevant platform-dependent software component was described in one of the previous sections.

Asset Download Implementation

This operation is defined within assets service. Once asset download is requested, all the files for the current asset are fetched from the database. For each file, then, content is fetched using file management service. Zip class is then used to compose zip file data using file names and content. The result is exposed as a corresponding command result.

Asset Validation Rules

Standard asset validation focuses mainly on geometry files. Converting every geometry to unified format, validation procedure controls the following facts:

- Geometry file cannot be converted (produces fatal error)
- Mesh contains vertex index out of bounds (produces error message)
- Texture reference contains path (produces information message)
- Mesh contains no material (produces warning message)
- Material references texture, but no texture with the same name is attached to the asset (produces error)
- Mesh has no texture coordinates but references a texture (produces error)
- Not every vertex has a texture coordinate (produces warning)

6 Conclusion

The desired system is implemented and corresponds to the conformed specification. As a final test, twenty five 3D models from the existing database were loaded into the implemented system. System shows acceptable performance and user experience. See Attachment I for full list of implemented requirements.

6.1 Performance

System performance was measured with respect to the most complex operation: asset validation. 2 assets were chosen as test cases. Test was performed using modern desktop PC with Intel® Core(TM) i5-3470 CPU running at 3.20GHz and 8,00 GB RAM. Browser used for the test: Google Chrome Version 42.0.2311.152 m. Test cases and corresponding results are the following:

Test Case	Geometry Formats	Vertices	Polygons	Validation Time
1	4	18466 * 4	16885 * 4	10 000.000ms
2	1	544 * 1	1599 * 1	301.709 ms

Table 9: System Performance Test

6.2 Suggestions

The following section describes author's suggestions for further development.

Unit and Integration Tests

To guarantee that system will run well in production environment, a set of unit and integration tests should be implemented along with UI tests. Additionally, stress testing is required to confirm, that system may handle reasonable amount of connections at once.

Asset File Types

System may be extended to handle other multimedia assets, such as sounds. Such extension will require several changes to Asset File entity, as well as validation implementation. Front-end may then use JavaScript API to play sound.

Validation Extensions

Geometry may be validated in a lot of different ways. Hence, custom validation components could be implemented to support any kind of validation, which can be performed using unified geometry data structure.

Unified Texture Format

Additional platform-dependent software component can be implemented to convert a variety of texture formats into a unified texture data structure. This structure then may be used for Front-end to avoid custom loader implementation for each texture format.

7 Bibliography

[1] "Selecting a development approach", Centers for Medicare & Medicaid Services (CMS) Office of Information Service (2008), United States Department of Health and Human Services (HHS), Accessed April 15, 2015, [online: http://www.cms.gov/Research-Statistics-Data-and-Systems/CMS-Information-Technology/XLC/Downloads/SelectingDevelopmentApproach.pdf]

[2] "Unified Modeling Language User Guide, The" (2 ed.). Addison-Wesley. 2005. p. 496. ISBN 0321267974.

[3] "Domain Model", M. Fowler, [online: http://martinfowler.com/eaaCatalog/domainModel.html]

[4] "Repository", M. Fowler, [online: http://martinfowler.com/eaaCatalog/repository.html]

[5] "Service Layer", M. Fowler, [online: http://martinfowler.com/eaaCatalog/serviceLayer.html]

[6] "Event Aggregator", M. Fowler, [online:http://martinfowler.com/eaaDev/EventAggregator.html]

[7] "MVC", M. Fowler, [online: http://martinfowler.com/eaaCatalog/modelViewController.html]

[8] OMG (2008). OMG Unified Modeling Language (OMG UML), Superstructure, V2.1.2 p.146.

[9] "Single-page Application", [online: https://en.wikipedia.org/wiki/Single-page_application]

[10] "Performance bechmark of popular PHP frameworks", Lukasz Kujawa, [online: http://systemsarchitect.net/performance-benchmark-of-popular-php-frameworks/]

[11] "June 2013 Web Server Survey", [online: http://news.netcraft.com/archives/2013/06/06/june-2013-web-server-survey-3.html]

[12] "April 2015 Web Server Survey", [online: http://news.netcraft.com/archives/2015/04/20/april-2015-web-server-survey.html]

[13] "JsonMapper", netreasearch, [online: https://github.com/netresearch/jsonmapper/blob/master/src/JsonMapper.php]

[14] Owens, Michael (2006). "Chapter 4: SQL". In Gilmore, Jason; Thomas, Keir. The Definitive Guide to SQLite. D. Richard Hipp (foreword), Preston Hagar (technical reviewer). Apress. p. 133. ISBN 978-1-59059-673-9. Retrieved 30 December 2014.

[15] "Assimp2JSON", LightHouse3D, [online: https://github.com/lighthouse3d/Demos-and-Tools/tree/master/Assimp2JSON]

Attachment A: Full list of system requirements

This list is used to control the requirements statuses. The requirement is considered to be implemented only if both back-end and front-end parts expose all the operation related to this requirement.

Functional Requirements:

Asset Management:

System should allow to create new asset [IMPLEMENTED]
System should allow to remove existing asset [IMPLEMENTED]
System should allow to access existing asset [IMPLEMENTED]
System should allow to modify existing asset [IMPLEMENTED]
System should allow to search assets by title, author, category and tags [IMPLEMENTED]
System should allow to preview web-compatible pictures within the asset [IMPLEMENTED]
System should allow to preview valid geometry within the asset [IMPLEMENTED]
System should allow to download the existing asset [IMPLEMENTED]
System should be able to control the completeness of existing asset [IMPLEMENTED]
System should be able to generate a file containing asset meta information [IMPLEMENTED]
System should store the asset in a separated folder [IMPLEMENTED]
System should allow to pend, approve or decline existing asset [IMPLEMENTED]

Asset Files Management:

System should allow to attach file to existing asset [IMPLEMENTED]
System should allow to change file attached to existing asset [IMPLEMENTED]
System should allow to detach file from existing asset [IMPLEMENTED]
System should allow to access file attached to existing asset [IMPLEMENTED]

Users Management:

System should allow to register user [IMPLEMENTED]
System should allow to remove user [NOT IMPLEMENTED]
System should allow to modify user data [IMPLEMENTED PARTIALLY]
System should allow to access user data [IMPLEMENTED]

Categories Management:

System should allow to add asset category [IMPLEMENTED]
System should allow to access asset category [IMPLEMENTED]
System should allow to modify asset category [IMPLEMENTED]
System should allow to remove asset category [IMPLEMENTED]

Comments Management:

System should allow to attach comment to existing asset [IMPLEMENTED]

System should allow to remove comment from existing asset [NOT IMPLEMENTED]

System should allow user to modify comment attached to existing asset [NOT IMPLEMENTED]

System should allow user to access comment attached to existing asset [IMPLEMENTED]

Geometry Conversion:

System should be able to convert any geometry supported by Assimp Library to a data structure.[IMPLEMENTED]

User Interface

System should expose online model viewer based on WebGL technology. Such viewer should be able to visualize the given 3D mesh, rotate it and change shading/lightning models. [IMPLEMENTED]

Non-functional requirements:

System should expose web interface for all the operations.

System should be implemented using PHP, C++, XML and web technologies.

System should not use any database technology, which requires standalone server.