Implementierung

Praxis der Softwareentwicklung

Wintersemester 17/18

Autorisierungsmanagement für eine virtuelle Forschungsumgebung für Geodaten

Autoren:

Bachvarov, Aleksandar
Dimitrov, Atanas
Mortazavi Moshkenan, Houraalsadat
Sakly, Khalil
Slobodyanik, Anastasia
Voneva, Sonya

07.02.18

Inhaltsverzeichnis

1 Einleitung

Dieses Dokument beschreibt die Änderungen, welche an unserem Entwurfsdokument vorgenommen wurden, um die Funktionalität des Projekts „Autorisierungsmanagement für eine virtuelle Forschungsumgebung für Geodaten“ zu gewährleisten. Das Ziel dieses Dokuments ist, die Gründe für diese Änderungen zu erläutern, beziehungsweise Probleme aufzuzeigen, die sich während der Implementierung ergeben haben.

Es werden Änderungen an Datenhaltung und Applikationslogik beschrieben sowie begründet. Des Weiteren wird ein Überblick darüber vermittelt, welche Kriterien aus dem Pflichtenheft im Ergebnisprodukt erfüllt wurden.

Am Schluss des Dokuments befindet sich ein Vergleich zwischen unserem ursprünglichen und tatsächlichen Implementierungsplan sowie Erläuterungen eventueller Verzögerungen im Zeitplan.

2 Änderungen am Entwurf

2.1 Model

Während der Designphase wurden Klassen des Model-Pakets entsprechend der Prinzipien objektorientierter Programmierung entworfen. Während der Einarbeitung in das Django-Framework hat sich allerdings herausgestellt, dass Model-Klassen eher als Datenstruktur-Träger benutzt werden und keine Anwendungslogik in sich kapseln. Daher wurde sämtliche Funktionalität in dem View-Paket implementiert, sodass Model-Klassen ausschließlich die Datenbankstruktur definieren.

- **Klasse CustomUser**
  Klasse *CustomUser* erbt von der Django-Klasse *User* und ersetzt Klasse *User* aus dem Entwurfsdokument. Sämtliche Attribute werden von Django vordefiniert und mussten nicht zusätzlich implementiert werden.

- **Klasse Admin**
  Da diese Klasse keine Funktionalität in sich kapselt, hat sich herausgestellt, dass sie durch die Benutzung des Attributs *is staff* ersetzt werden kann. Dieses Attribut wird in der *User*-Klasse von Django vordefiniert.

- **Klasse Resource**
  Um Lese- und Besitzerrechte zu implementieren, werden der Klasse zusätzliche Attribute hinzugefügt: die Listen *Readers* und *Owners*.

- **Klasse ResourceType**
  Im Implementierungsprozess hat sich herausgestellt, dass diese Klasse keine Auswirkung auf die Funktionalität des Produkts hat, weswegen diese Klasse nicht implementiert wurde, um unnötigen Aufwand zu vermeiden.

- **Klasse Request**
  Dieser Klasse wurde das *Description*-Attribut hinzugefügt. Dieses Attribut hat Type *String* und dient dazu, die Begründung zu speichern, welche vom Absender bei der Erstellung des Requests eventuell eingegeben wurde.

- **Klassen Logging und EmailMessages**
  Diese von Django vordefinierten Klassen mussten nicht extra implementiert werden.

2.2 View

- **Klasse ChosenRequestsView**
  Funktionalität dieser Klasse wurde auf vier Views verteilt:

  - *ApproveAccessRequest*
  - *DenyAccessRequest*
  - *ApproveDeletionRequest*
  - *DenyDeletionRequest*

  Diese Unterteilung hat bessere Trennung der Anwendungslogik für Bearbeitung unterschiedlicher Requests zur Folge.

- **Klasse DeleteResourceView**
  Dieser View erweitert Funktionalität von Klassen *ManageResourcesView* und *ResourcesOverview* und wird zum Löschen der Ressourcen von Administratoren des Portals benutzt. Absenden eines Löschrequests wird stattdessen im *SendDeletionRequestView* implementiert.

- **Klassen PermissionForChosenResourceView, PermissionsForResourceView und PermissionsForUsersView**
  Diese Views wurden durch *PermissionEditingView* ersetzt und für bessere Übersichtlichkeit durch *PermissionEditingViewSearch* erweitert. Bearbeitung der Rechte aller Ressourcen in einem View hat bessere Benutzbarkeit des Portals zur Folge.

- **Klassen ManageUsersView und ManageResourcesView**
  Diese Views wurden durch vordefinierte Django-Funktionalität ersetzt.

- **Klasse ResourcesOverview**
  Dieser View wurde durch einen zusätzlichen View *ResourcesOverviewSearch* für bessere Benutzbarkeit des Portals erweitert.

- **Klasse RequestView**
  Funktionalität dieses Views wurde für bessere Struktur der Anwendungslogik auf vier Views verteilt:

  - *SendAccessRequest*
  - *CancelAccessRequest*
  - *SendDeletionRequest*
  - *CancelDeletionRequest*

- **Klasse ResourceInfoView**
  Funktionalität dieses Views wurde durch einen Modaldialog implementiert, um überflüssige Codezeilen zu sparen.

- **Zusätzlich implementierte Views**
  Während der Implementierung hat sich herausgestellt, dass einige Funktionalitäten des Portals zusätzliche View-Klassen benötigen. Die noch nicht erwähnten Views heißen:

  – *AddNewResourceView* dient zur Erstellung einer neuen Ressource;

  – *EditNameView* wird zum Ändern des Benutzernamens benutzt.

2.3 URL-Verzeichnis

Änderungen an Views haben dementsprechende Änderungen am URL-Verzeichnis verursacht. Eine URL lokalisiert einen View, der entweder als ein Main-Fenster oder dessen Teil präsentiert wird.

Home-Seite mit Authentifizierungsfunktionalität für Testzwecke:

- **/home**

  /register

  /login

  /logout

  Verwaltungsseiten:

- **/resource-manager**
- **/user-manager**

  Benutzerseite:

- **/profile**

  /my-resources

  /resourceid-edit-users-permissions

  /search

  /add-new-resource

  /edit-name

Bearbeitung der Requests:

- **/approve-access-request/requestid**
- **/deny-access-request/requestid**
- **/approve-deletion-request/requestid**
- **/deny-deletion-request/requestid**

Absenden und Löschen der Requests:

- **/send-access-request/resourceid**
- **/cancel-access-request/resourceid**
- **/send-deletion-request/resourceid**
- **/cancel-deletion-request/resourceid**

Ressourcenübersicht:

- **/resources-overview**

  /search

Metadaten einer Ressource

- **/resources/resourceid**

Löschen der Ressourcen für Administratoren:

- **/delete-resource/resourceid**

2.4 Datenbank

Jede Änderung am Model-Paket beeinflusst die Datenbankstruktur. Der von Django unabhängige Teil der Datenbank, die für die Datenhaltung in unserem Projekt benutzt wird, wird auf der Abbildung 1 dargestellt. Die im Entwurf geplante Tabelle *Permission* wurde durch *many-to-many* Beziehungen zwischen Benutzern und Ressourcen ersetzt und in zwei Tabellen *Resource Owners* und *Resource Readers* gespeichert. Diese Änderung sorgt für bessere Antwortzeiten der Datenbank, was der besseren Benutzbarkeit dient. Unterschiedliche Arten von Requests werden in eigenen Tabellen gespeichert. Das ermöglicht bessere Trennung zwischen Benutzer- und Admin-Funktionalitäten. Bearbeitete Requests werden nicht weiter gespeichert und sind ausschließlich in der Logdatei aufgeführt.

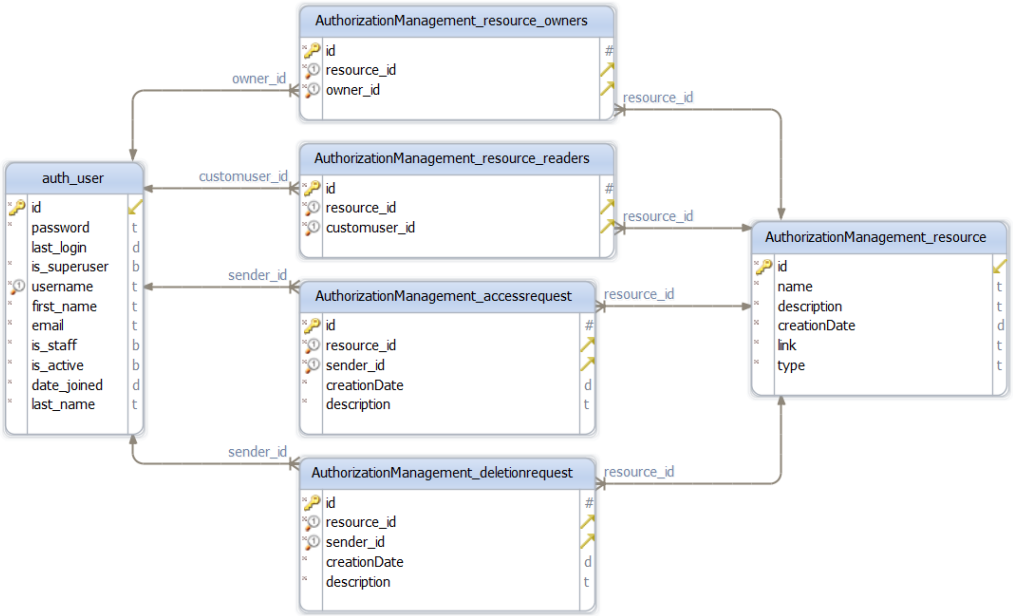


Abbildung 1: Schema der Datenbank. Jedes Rechteck repräsentiert eine Tabelle, jedes Attribut - eine Spalte in der entsprechenden Tabelle in der Datenbank. Die Pfeile geben Auskunft, welches Attribut in welcher Tabelle als Schlüssel vorkommt.

3 Implementierte Kriterien

3.1 Musskriterien

Während der Planungsphase wurde eine Anforderungsdefinition im Pflichtenheft festgelegt. Diese Anforderungen beschreiben, welche Muss-, Wunsch- und Abgrenzungskriterien das zu entwickelnde Produkt erfüllen muss. In der Implementierungsphase unseres Projektes wurden alle Musskriterien erfolgreich implementiert, was für präzise und eindeutige Aufgabenformulierung spricht und zeigt, dass das Wasserfallmodel an der Softwareentwicklung ergebnisreich angewendet werden kann.

3.2 Wunschkriterien

Bei der Implementierung funktionaler Anforderungen hat sich ergeben, dass die Realisierung einiger Wunschkriterien die Benutzbarkeit des Produkts deutlich verbessern würde. Als Folge wurden unten genannte Kriterien zusätzlich implementiert:

- Wird eine Ressource gelöscht, so werden alle Besitzer per E-Mail benachrichtigt.
- Ein Benutzer kann durch Administratoren blockiert werden.
- Der Administrator kann Benutzer anhand von Vorname und/oder Nachname suchen.

Unittests

Setup

Setup-Funktionen erschaffen die Testumgebung beziehungsweise Testressource für alle Testcases:

- setUpUsers
- setUpResourceAndRequests

Testcases

Um Codequalität zu gewährleisten, wurde zu jedem View entsprechende Testcases implementiert. Ein Testcase ist eine Testeinheit, die das angemessene Verhalten eines bestimmten Teils der Funktionalität unter unterschiedlichen Bedingungen überprüft. Die in unserem Projekt implementierten Testcases:

- TestHomeView
- TestResourceManager
- TestUserManager
- TestProfileView
- TestMyResourcesView
- TestSendDeletionRequest
- TestCancelDeletionRequest
- TestApproveAccesRequest
- TestDenyAccesRequest
- TestSendAccessRequest
- TestCancelAccessRequest
- TestDeleteResourceView
- TestSendDeletionRequest
- TestEditNameView
- TestResourcesOverview

- TestResourcesOverviewSearch
- TestPermissionEditingView
- TestPermissionEditingViewSearch

Cleanup

Cleanup-Funktionen bringen das System in den ursprünglichen Zustand, um Konsistenz der Datenbank vor und nach dem Testen zu garantieren:

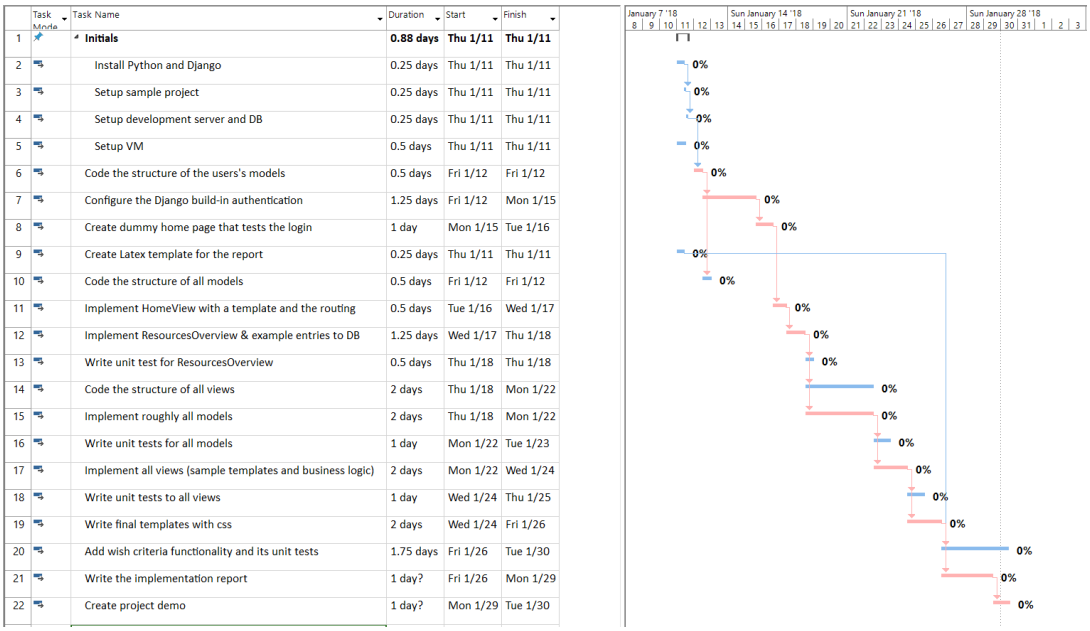
- deleteUsers
- deleteResourcesAndRequests

4 Implementierungsplan

Zu Beginn der Implementierung wurde ein Gannt-Zeitplan erstellt (Abbildung 2), in dem wir unsere ursprüngliche Aufwandsschätzung dargestellt haben. In dem realen Ablauf der Phase kam es zu einigen Abweichungen vom Zeitplan.

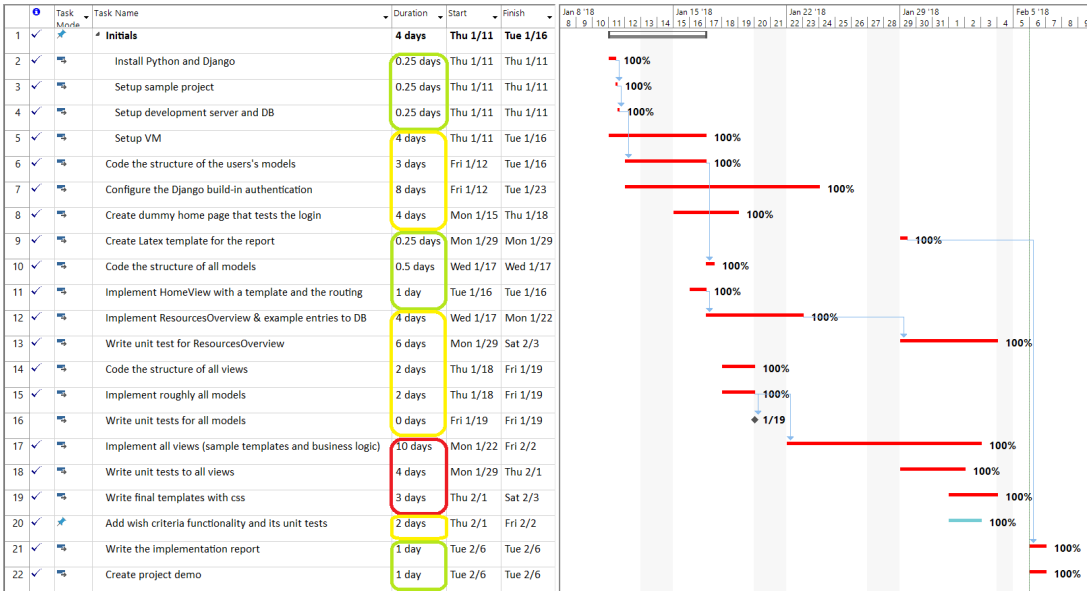
Ein Grund dafür ist, dass einige Aktivitäten stark zusammenhängen und Änderungen am Entwurf Anpassungen an schon implementierten Teilen erfordern. Als Folge werden abgeschlossene Aktivitäten vorgesetzt, weswegen die geplante Dauer sich mehrfach erhöhen kann. Strikte Zeitplanung jeder Aktivität gemäß des Wasserfallmodel erfordert Erfahrung mit benutzten Tools, in unserem Fall mit Django.

Realer Zeitablauf (Abbildung 3) zeigt Verzögerungen bei geplanten Aktivitäten und das Maß der Auswirkung jeder Verzögerung auf gesamten Zeitverlauf. Trotz aller Verzögerungen wurde die Implementierung des Projekts nahezu fristgerecht abgeschlossen.



	Task Mode	Task Name	Duration	Start	Finish
1		**Initials**	**0.88 days**	**Thu 1/11**	**Thu 1/11**
2		Install Python and Django	0.25 days	Thu 1/11	Thu 1/11
3		Setup sample project	0.25 days	Thu 1/11	Thu 1/11
4		Setup development server and DB	0.25 days	Thu 1/11	Thu 1/11
5		Setup VM	0.5 days	Thu 1/11	Thu 1/11
6		Code the structure of the users's models	0.5 days	Fri 1/12	Fri 1/12
7		Configure the Django build-in authentication	1.25 days	Fri 1/12	Mon 1/15
8		Create dummy home page that tests the login	1 day	Mon 1/15	Tue 1/16
9		Create Latex template for the report	0.25 days	Thu 1/11	Thu 1/11
10		Code the structure of all models	0.5 days	Fri 1/12	Fri 1/12
11		Implement HomeView with a template and the routing	0.5 days	Tue 1/16	Wed 1/17
12		Implement ResourcesOverview & example entries to DB	1.25 days	Wed 1/17	Thu 1/18
13		Write unit test for ResourcesOverview	0.5 days	Thu 1/18	Thu 1/18
14		Code the structure of all views	2 days	Thu 1/18	Mon 1/22
15		Implement roughly all models	2 days	Thu 1/18	Mon 1/22
16		Write unit tests for all models	1 day	Mon 1/22	Tue 1/23
17		Implement all views (sample templates and business logic)	2 days	Mon 1/22	Wed 1/24
18		Write unit tests to all views	1 day	Wed 1/24	Thu 1/25
19		Write final templates with css	2 days	Wed 1/24	Fri 1/26
20		Add wish criteria functionality and its unit tests	1.75 days	Fri 1/26	Tue 1/30
21		Write the implementation report	1 day?	Fri 1/26	Mon 1/29
22		Create project demo	1 day?	Mon 1/29	Tue 1/30

Abbildung 2: Ursprünglicher Implementierungsplan. Jede Aktivität wird in jeweiliger Zeile mit einem waagerechten Balken visualisiert. Je länger der Balken, desto länger dauert die Aktivität. Die Beziehungen (beziehungsweise Abhängigkeiten) zwischen Aktivitäten werden mit Pfeilen dargestellt.



	Task Mode	Task Name	Duration	Start	Finish
1		**Initials**	**4 days**	**Thu 1/11**	**Tue 1/16**
2		Install Python and Django	0.25 days	Thu 1/11	Thu 1/11
3		Setup sample project	0.25 days	Thu 1/11	Thu 1/11
4		Setup development server and DB	0.25 days	Thu 1/11	Thu 1/11
5		Setup VM	4 days	Thu 1/11	Tue 1/16
6		Code the structure of the users's models	3 days	Fri 1/12	Tue 1/16
7		Configure the Django build-in authentication	8 days	Fri 1/12	Tue 1/23
8		Create dummy home page that tests the login	4 days	Mon 1/15	Thu 1/18
9		Create Latex template for the report	0.25 days	Mon 1/29	Mon 1/29
10		Code the structure of all models	0.5 days	Wed 1/17	Wed 1/17
11		Implement HomeView with a template and the routing	1 day	Tue 1/16	Tue 1/16
12		Implement ResourcesOverview & example entries to DB	4 days	Wed 1/17	Mon 1/22
13		Write unit test for ResourcesOverview	6 days	Mon 1/29	Sat 2/3
14		Code the structure of all views	2 days	Thu 1/18	Fri 1/19
15		Implement roughly all models	2 days	Thu 1/18	Fri 1/19
16		Write unit tests for all models	0 days	Fri 1/19	Fri 1/19
17		Implement all views (sample templates and business logic)	10 days	Mon 1/22	Fri 2/2
18		Write unit tests to all views	4 days	Mon 1/29	Thu 2/1
19		Write final templates with css	3 days	Thu 2/1	Sat 2/3
20		Add wish criteria functionality and its unit tests	2 days	Thu 2/1	Fri 2/2
21		Write the implementation report	1 day	Tue 2/6	Tue 2/6
22		Create project demo	1 day	Tue 2/6	Tue 2/6

Abbildung 3: Realer Zeitablauf der Implementierung mit geänderter Dauer der Aktivitäten. Abhängig von Folgen der Verzögerungen werden Aktivitäten farblich markiert: grün - rechtzeitig/keine negative Folgen, gelb - von anderen Aktivitäten verursachte Verzögerungen, rot - Aktivitäten, während deren Entwurfsprobleme gelöst werden mussten.