Technische Dokumentation

WIP/outdated

Backups

Mithilfe der Backuplösung Tartarus werden die Daten aus den Verzeichnis AWP automatisiert in der Storage Box von Hetzner gesichert. Es wurde sich für diese Lösung entschieden, da auch schon die Server bei Hetzner gebucht sind und der Backup-Speicher gratis mitgeliefert wurde.

Der Befehl *tartarus* startet die Ausführung des Backups. Dabei muss noch eine Konfigurationsdatei angegeben werden. Diese liegen für verschiedene Backup-Profile unter */etc/tartarus/*.conf* . Sie enthalten die Informationen, welches Verzeichnis gesichert wird und wo Tartarus die Markierungen für die inkrementellen Backups ablegt. Die Markierungsdateien pflegt Tartarus selbst, allerdings muss für jedes Backupprofil zuvor ein Pfad zur Markierungsdatei angelegt werden. Dieser sieht für das Backupprofil des Minetestverzeichnis wie folgt aus:

`/var/spool/tartarus/timestamps/backupprofilname` Beim Anlegen eines neuen Backupprofils muss der Pfad angepasst werden.

Die *generic.inc* Datei enthält die wichtigsten Einstellungen bezüglich der Verbindung zum Hetzner-Server. Zusätzlich kann mit der Option *-i* auf inkrementelle statt komplette Backups gewechselt werden. Hierfür muss zuerst ein komplettes Backup durchgeführt werden, bevor ein inkrementelles Backup mit dem selben Profil gestartet werden kann.

`_sudo tartarus /etc/tartarus/Name der Backupprofilkofgurationsdatei_` erstellt ein Backup nach dem angegebenen Konfigurationsprofil.

`_sudo tartarus -i /etc/tartarus/Name der Backupprofilkofgurationsdatei_` erstellt ein inkrementelles Backup nach dem angegebenen Konfigurationsprofil.

Um nach jedem Backup eine Bereinigung der Storagebox durchzuführen, wird nach Tartarus das Skript *charon.ftp* unter */usr/sbin* / automatisiert aufgerufen. Dieses löscht nach dem Erstellen des neuen Backups alle zum gleichen Profil gehörenden Backups die älter als 13 Tage sind. So ist gewährleistet, dass der Speicherplatz der Storage Box wieder freigegeben wird. In der /etc/tartarus/`_generic.inc_` wurde dafür eine Hook eingefügt, dort lässt sich auch das maximale Alter festlegen. Komplette Backups können nur gelöscht werden, wenn alle darauf basierenden inkrementellen Backups gelöscht wurden. Im Verwaltungsrobot von Hetzner https://robot.your-server.de/storage kann außerdem der genutzte Speicherplatz überprüft werden. Wichtig zu beachten ist, dass der Robot von Hetzner nur alle 15 Minuten aktualisiert.

Um die Inhalte vom Server aus zu erreichen, wurden der Backupstorage unter `_/home/awp/backups_` gemounted. Mit `_sudo crontab -l_` lässt sich die aktuelle Automatisierung einsehen. Mit *sudo crontab -e* lässt sich die Automatisierung anpassen.

```
#
# m     h       dom     mon     dow     command
  30    22        *       *      mon-sat  /usr/sbin/tartarus -i /etc/tartarus/awp.conf
  30    22        *       *      sun      /usr/sbin/tartarus etc/tartarus/awp.conf
```

Diese zwei Einträge gewährleisten, dass um 22:30 Uhr eine Sicherung durch Tartarus ausgeführt. Montag bis Samstag werden diese Sicherungen mit dem Konfigurationsprofil inkrementell durchgeführt. Sonntags wird eine komplette Sicherung mit dem Konfigurationsprofil durchgeführt.

Dokumentation CheckMK

Mithilfe des Monitoring-Tools Checkmk werden die wichtigsten Services des Servers überwacht und überprüft. Check-MK ist über eine Weboberfläche zu bedienen und kann in dieser Oberfläche auch konfiguriert werden.

Die Weboberfläche ist nur nutzbar wenn das Monitoring gestartet wurde. Check-MK nutzt zum Monitoring die „open monitorung distribution“ (omd). OMD kann Instanzen, sogenannte sites, erstellen mit denen man unabhängig voneinander den Server überwachen kann. Aktuell ist eine Instanz erstellt worden, die „monitoring“ genannt worden ist. Um jetzt das Monitoring auf dieser Instanz zu starten, muss der Befehl *omd start monitoring* ausgeführt werden. Die Instanz kann mit dem Befehl *omd stop monitoring* auch wieder beendet werden.

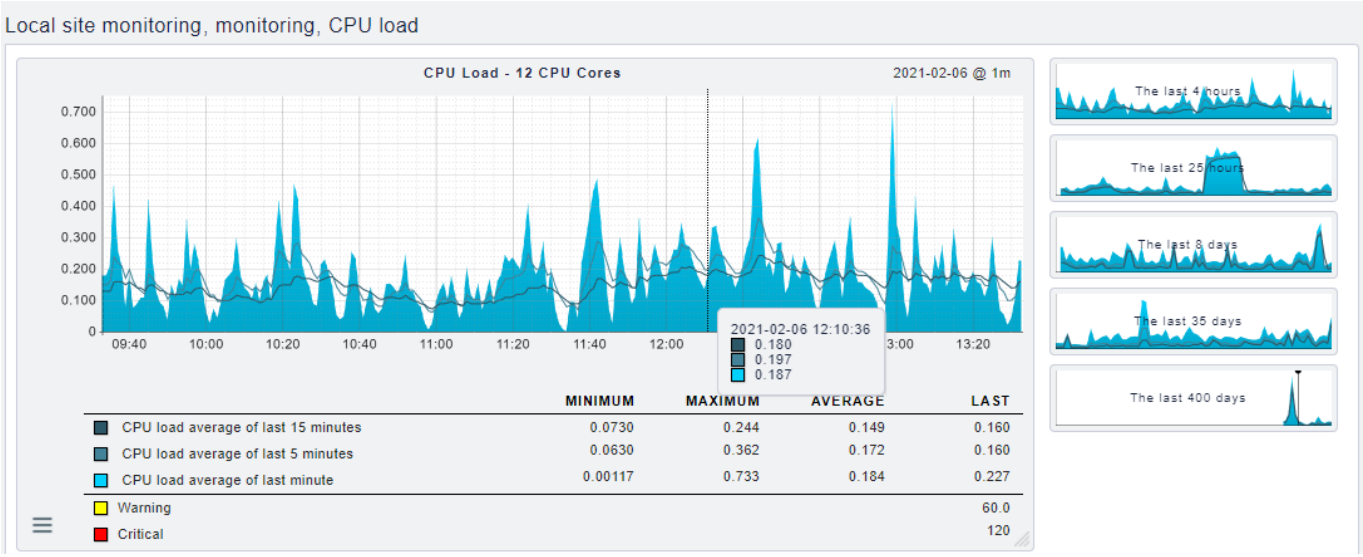
Ist die Instanz gestartet, kann die Weboberfläche über die URL http://minetest.lmz-bw.de/monitoring erreicht werden.

Hosts:

Checkmk braucht zur Überwachung von Services einen Host. Hier wurde der Linux-Server mit der IP-Adresse (188.40.113.24) als Host eingerichtet. Jeder Host hat eine Anzahl an Services die von Checkmk automatisch erkannt und überwacht werden.

Services:

Auf der Startseite werden alle aktuell überwachten Services angezeigt. Man bekommt die wichtigsten Informationen über jeden überwachten Service auf einem Blick. Wenn man auf den Namen des Service klickt (zum Beispiel CPU load), bekommt man alle Informationen zu diesem Service. Jeder Service bekommt vom System einen Zustand zugeteilt, der sich aus den festgelegten Regeln für diesen Service ableitet. Der Zustand ist dabei entweder ok, warn, crit, unknown oder pend.
Mit dem Menü-Button kann man den Check automatisch neu starten oder die Check-Parameter anzeigen lassen und mit dem Graphen-Symbol, bekommt man eine oder mehrere Grafiken, welche einem die Entwicklung der Zahlen zu diesem Service anzeigt.



In der Grafik sieht man eine Übersicht über den Verlauf des Services über die letzten 4 Stunden. Man kann die Grafik dabei sowohl an der x-Achse (Zeit), als auch an der y-Achse (CPU load) verändern und auch heranzoomen. Somit ist es möglich, sich die Grafik anzeigen zu lassen, die man braucht. Man bekommt außerdem die Schwellenwerte angezeigt, wann ein Wert den Status warning oder critical erhält und kann dadurch sehen wie sehr der Service aktuell ausgelastet ist.

Aktuell werden 20 Services überwacht. Um auf alle Services zugreifen zu können, beziehungsweise um andere Services auch zu überwachen, muss man die Konfiguration im Wato-Menü ändern. Dafür muss man in der Wato-Configuration auf Host klicken und beim Host „monitoring“ das Servicesymbol auswählen. In diesem Menü werden alle erkannten Services angezeigt und man kann Services nach Belieben aktivieren oder deaktivieren. Um die Regel für den Status eines Services zu ändern, muss man die Check-Parameter mit einer neuen Regel anpassen. Dafür auf das Parameter Symbol klicken und eine neue Regel mit neuen Werten im „main folder“ erstellen.
Wichtig ist, dass wenn etwas verändert oder etwas hinzugefügt wird, man diese Änderung erst noch bestätigen muss, bevor die Änderung vom System durchgeführt wird.

User:innen:

Man kann alle User:innen im Wato-Menü unter dem Punkt User einsehen. In diesem Menü können sowohl neue User:innen angelegt, als auch aktuelle User:innen bearbeitet werden. Aktuell sind die User cmkadmin und aufseher eingerichtet. Beide User haben volle Administrationsrechte und für beide ist auch eine Mail-Adresse eingetragen. Man findet die Login-Daten für beide User in der Passwort-Datenbank.

Benachrichtigung:

Für beide aktiven User ist eine Mail-Adresse eingetragen, die eine Mail erhält, sobald sich der Status eines Services verändert. In der Mail bekommt man dann Informationen, welcher Service sich verändert hat und eine Grafik die sich die Entwicklung des Service über den letzten Tag anzeigt. Beide Nutzer sind aktuell in der Kontaktgruppe „Everything“ eingetragen. Will man nicht über jeden Service benachrichtigt werden, muss man die Kontaktgruppe im Wato-Menü anpassen oder eine neue Kontaktgruppe anlegen.

Logs:

Die unterschiedlichen Logs werden von omd gespeichert. Sie sind in dem Ordner */omd/sites/monitoring/var/log* / zu finden.

Weitere Dokumentation:

Eine ausführliche Dokumentation zu allen Features von Checkmk ist hier zu finden: https://docs.checkmk.com/latest/de/index.html

Dokumentation Service-Architektur

2 Server:

- Webhosting (www.blockalot.de):

Services:

- Webseite
- Mail Server

Zugriff:

- KonsoleH unter der URL https:<nowiki>//</nowiki>konsoleh.your-server.de (Zugangsdaten: KonsoleH im Keepass)

Backend-Server (minetest.lmz-bw.de):

Services:

1. Minetest
2. Postgresdatenbank
3. Backend
4. Keycloak
5. CheckMK

Zugriff:

- ssh-Adresse: 188.40.113.24 oder minetest.lmz-bw.de (Zugangsdaten: *awp user server* oder *root user server* im Keepass)

Dokumentation Backend

Das Backend stellt über eine Schnittstelle die benötigten Funktionalitäten für das Frontend bereit.

Schnittstelle

Bei der Schnittstelle handelt es sich um eine Rest-Schnittstelle, über die mithilfe der HTTP-Methoden benötigte Daten und Funktionalität bereitgestellt werden.

Über die URL https:<nowiki>//</nowiki>minetest.lmz-bw.de:1234/api-docs wurde, mithilfe von *swagger,* eine Dokumentation der aktuell zur Verfügung stehenden Endpunkte auf Basis des Open API 3.0.3 Standards erstellt. Zur Nutzung der Schnittstelle wird eine Authentifizierung mit einem *Bearer*

Token benötigt.

Technologie

Als Programmiersprache wurde für das Backend //TypeScript// in der Version 4.1.3. verwendet. Zum Erstellen der Schnittstelle wurde Express in der Version 4.17.1 genutzt. Das genutzte DBMS ist *PostgreSQL*, da es auch als Datenbank für die Minetest-Server dient. Zur Benutzerverwaltung nutzen wir einen *Keycloak*-Server, welcher auf der Version 12 läuft.

Versionierung

Der Code für das Backend wurde mithilfe von *Git* versioniert. Die komplette Commit-History findet man auf GitHub. Dabei ist zu beachten, dass sich der aktuelle Stand der Entwicklung im *main*-Branch befindet. Im Branch *live* liegt der aktuell laufende Stand, verfügbar unter dieser URL.

Pipeline

Mithilfe von GitHub Actions wurden zwei Pipelines eingerichtet.

Bei der Ersten handelt es sich um eine Pipeline zur Qualitätssicherung. Es werden die folgenden Commands in der angegeben Reihenfolge ausgeführt:

1. npm ci → Überprüfung der eingesetzten Dependencies
2. npm run lint:ci → Überprüfung der Code-Qualität mithilfe von ESLint

Mithilfe der zweiten Pipeline kann der Code auf der aktuell genutzten Linux-Maschine bei Hetzner deployed werden. Bei jedem eingehenden Commit auf dem Branch *live* wird ein Skript ausgeführt, welches den Code via FTP auf den Linux-Server transferiert und die folgenden Commands ausführt:

1. *systemctl –user stop prod_backend* → Stoppen des Linux-Service
2. *npm install* → Herunterladen der benötigten Dependencies
3. *npx tsc* → Kompilierung mithilfe des TypeScript-Compilers
4. *systemctl –user start prod_backend* → Neustart des Linux-Service

Außerdem wurde mit *Husky* ein *pre-commit Hook* eingerichtet, der vor jedem versuchten Commit das Command *npm run lint:ci* ausführt. Damit soll verhindert werden, dass Code mit unzureichender Code-Qualität im Repository landet.

Authentifizierung

Um sich am Backend erfolgreich zu authentifizieren, muss man sich zunächst einen Token bei unserem Keycloak-Server unter dieser URL abholen. Dazu muss ein POST-Request mit dem Content-Type *application/x-www-form-urlencoded* mit dem folgenden Payload gesendet werden:

- grant_type: password
- username: <html><Testuser></html>
- password: <html><Passwort></html>

- client_id <html><Client-ID></html> → der Client, den wir derzeit für das Frontend verwenden heißt *vue-client*

Gibt man die o.g. Daten richtig an, erhält man in der Response unter der Property *access_token* einen *JWT*, der bei einem Request an das Backend als *Authorization-Header* mitgeschickt werden muss, um sich zu authentifizieren.

Ablauf eingehender Requests

Bei der Bearbeitung der eingehenden Requests wird der folgende Ablauf verwendet:

1. Parsing des Body mithilfe von express.json()
2. Sicherheitsüberprüfung des Requests mithilfe von helmet
3. Authentifizierung und Extrahieren der User ID aus dem Token → Custom Middleware unter /src/security/validatejwt.ts
4. Ausführung eines Handlers, der die Ablaufsteuerung für die Abarbeitung des Requests übernimmt → Handler-Funktionen unter /src/handler/<html><http-methode></html>-handler.ts
5. Ausführung der verschiedenen Services durch den Handler → Service-Funktionen unter /src/service
6. Error Handling → mithilfe eines Error-Handlers unter /src/handler/error-handler.ts
7. Senden einer entsprechenden Response

Code-Dokumentation

Der Code wurde nach der Empfehlung aus der TypeScript-Dokumentation mit Kommentaren im JSDoc-Format kommentiert.

Dokumentation Website

Das Frontend von Blockalot ist der Teil der Anwendung, den die Endnutzer bedienen, um ihre Lernräume zu verwalten.

Design

Das Design der Website orientiert sich am Corporate Design des LMZ: Die Farben, insbesondere Orange und Olivgrün, der Homepage des LMZ wurden übernommen. Auch das Logo sowie einige Grafiken wurden übernommen beziehungsweise abgewandelt. Für einige Elemente, zum Beispiel die „Karten“ der Klassenzimmer auf dem Dashboard, wurde *Neomorphismus* verwendet. Dieser sorgt zwar für ein modernes Aussehen, dessen Tauglichkeit, insbesondere im Bezug auf Kontrast, sollte weiter überprüft werden.

Ein Mockup für die Website wurde mit *Figma* erstellt. Über diesen Link kann es eingesehen werden. Falls Änderungen vorgenommen werden müssen, wenden Sie sich bitte an *rech1033@hs-karlsruhe.de*, um Zugriff zu erhalten.

Die auf der Website verwendeten Farben sind in der Datei *src/css/colors.css* als globale Variablen definiert. Dies ermöglicht, dass Änderungen an diesen Farbcodes die entsprechenden Farben auf der gesamten Website ändern. Somit könnte die Anwendung zum Beispiel mit geringem Aufwand an ein anderes Corporate Design angepasst werden. Außerdem kann hierdurch die Implementierung eines „Darkmodes“ umgesetzt werden, für welchen schon ein Beispiel im Mockup zu finden ist.

Technologie

Das Frontend ist mit dem *JavaScript-Framework Vue* in der Version 3.0.0 geschrieben. Zum *State-Management* wurde *Vuex* benutzt. Dabei wurde versucht, auf externen Code zu verzichten, um Abhängigkeiten zu reduzieren.

Versionierung

Alle Abhängigkeiten und deren Versionen können im Repository auf Github beziehungsweise in der Datei *package.json* eingesehen werden. Außerdem wurde dort mithilfe von *GitHub Actions* analog zum Backend eine Pipeline zur Qualitätssicherung und zum automatischen Deployment eingerichtet.

Aufbau

Die Root-Datei ist *src/App.vue.* Hier werden die *Navbar*, der *Footer* und die aktuelle *View* eingebunden. Die aktuelle View wird über *Vue-Router* bestimmt, die entsprechende Datei ist *src/router/index.js*. Die Views liegen unter *src/views* und entsprechen den unterschiedlichen Seiten der Anwendung (Home, Dashboard, FAQ, Klassenzimmer, Impressum und Datenschutz).

Die Views binden unterschiedliche Komponenten ein, welche unter *src/components* gespeichert werden. Einige dieser Komponenten, zum Beispiel der Button, können innerhalb des gesamten Frontends wiederverwendet werden. Mit *Vuese* können mehr Informationen über die einzelnen Komponenten generiert werden. *Vuese* erstellt automatisch eine Dokumentation der Komponenten. Die generierten Dateien sind in *Markdown* geschrieben und liegen unter *website/components*. Es kann eine Website lokal gehostet werden, die alle *Markdown*-Dateien der Komponenten-Dokumentation aufbereitet darstellt. Hierfür muss global *Vuese* installiert und anschließend *vuese serve –open* ausgeführt werden (siehe Vuese-Dokumentation).

Der *Store* (State-Management) ist in *src/store/index.js* definiert. Hier werden *State-Manipulationen* durchgeführt und *API-Calls* aus der Datei *src/api/api.js* aufgerufen. Diese sind für die Kommunikation mit dem Backend zuständig.

Die Funktionen für die Passwort- und PDF-Generierung sind in den Ordner *src/functions* ausgelagert, um den Code übersichtlicher zu machen. Alle Grafiken liegen unter *src/assets.*

Dokumentation Keycloak

Der Keycloak-Server wird zur *Benutzerverwaltung* für die User der Webseite verwendet. Der Name des Service auf der Linux-Maschine heißt *keycloak.*
Die *Logs* sind unter dem folgenden Pfad zu finden: */keycloak-12.0.2/standalone/log/server.log*

Die Keycloak-Installation befindet sich im Ordner keycloak-*akt. Versionsnummer*. Keycloak wird im Modus *Standalone* ausgeführt (in diesem Ordner befindet sich auch die entsprechende Konfiguration). Es gibt einen *Realm* (=Sammlung aller Einstellungen und Konfigurationen) zum Testen (*lmz_dev*) und einen, der live ist (*lmz_prod*). Die Konfiguration ist größtenteils auch über ein User Interface möglich (z.B. Anlegen neuer User):
https://minetest.lmz-bw.de:8443/auth/admin/lmz_prod/console/ (Konfiguration aller Realms möglich)
https://minetest.lmz-bw.de:8443/auth/admin/lmz_prod/console/Die Zugangsdaten dazu findet man im *Keepass*. Das Aussehen der Login Page, sowie der Userprofile muss über Keycloak und nicht über das Frontend angepasst werden (Ordner Themes).

Dokumentation Postgres

Postgres ist das *DBMS*, das wir sowohl für unser Backend als auch für die Minetest-Server nutzen. Es hat keinen eigenen Servicenamen wie z.B. Keycloak. Die Verwaltung geschieht mittels folgendem command:
sudo pg_ctlcluster 13 main (reload|start|restart) Die Logs liegen unter _/var/log/postgresql/postgresql-13-main.log_In Postgres hat jede Welt ihre eigene Datenbank, welche 7 Tabellen enthält. Außerdem gibt es noch eine Datenbank, auf der unser Backend seine Daten zu angelegten Klassenzimmern verwaltet.

Der Zugriff auf die Postgres-Datenbank geschieht über SSH. Mittels *sudo -u postgres psql* kann auf die Datenbank zugegriffen werden. Mit *\l* werden alle Datenbanken angezeigt. Mit *\c datenbankName* kann eine Verbindung zu einer bestimmten Datenbank hergestellt werden. Mit *\d* können die enthaltenen Tabellen angezeigt werden.

Zur bequemeren Verwaltung kann über *pgAdmin* auf die Datenbank zugegriffen werden. Hierzu ist eine lokale Installation sowie ein Hinterlegen der IP-Adresse in der Config-Datei unter */etc/postgresql/13/main/pg_hba.conf* notwendig. Hierzu muss die IP-Adresse unten angehängt werden (nach dem Muster der vorhandenen Einträge). Danach muss die Konfiguration mit dem Command *sudo pg_ctlcluster 13 main reload* neu geladen werden.

In unserer Datenbank vom Backend können u.U. inkonsistente Datenbestände durch Fehler beim Testen entstanden sein. Diese können mit dem folgenden Command ermittelt werden:

SELECT * FROM (SELECT d.datacl, d.datname, w.* FROM pg_database AS d FULL JOIN worlds AS w ON „w“.„worldId“ = substring(d.datname, 2)::INTEGER
WHERE datistemplate = false AND datname LIKE 'k%') AS all_databases WHERE port IS NULL OR datname IS NULL ORDER BY datname;

Das Command gleicht die Datenbanken der Klassenzimmer mit den Einträgen in unserer Backenddata-Datenbank ab und gibt die Deltas aus.

Minetest:

Der Name der Services der einzelnen Klassenzimmer ergibt sich nach dem folgenden Service: _k_<*klassenzimmerID*>
Die Logs liegen In der Datei *debug.txt* im jeweiligen Klassenzimmerordner unter */home/awp/minetest-live/worlds/<html><userID></html>/<html><klassenzimmerID></html>*.

Dabei ist die userID die ID des Accounts aus Keycloak. Im Ordner des Klassenzimmers befinden sich außerdem alle zugehörigen Konfigurationsdateien.

Im Home Verzeichnis AWP (*/home/awp*) befindet sich im Ordner *minetest-live* die aktuelle verwendete Minetest-Installation, dort befinden sich im Ordner */mods* die aktuell global installierten Mods. Eine Dokumentation über die installierten Mods ist mod_uebersicht verfügbar.

Jede Welt hat eine eigene *Postgres-Datenbank* in der die Ingame Daten (Spielerinventar, Zugangsdaten, Blöcke der Welt, etc.) gespeichert werden (die Zugangsdaten sind in der jeweiligen world.mt zu finden (oder in der Passwortdatenbank siehe. Postgres).

Jede Welt hat in Systemd einen eigenen Service.

Weitere Infos:

Die Installation ist folgendermaßen erfolgt:

Navigiere in den gewünschten Ordner

```
export branch=$(printf "Enter Minetest version: ">&2;read r;echo "$r")
```

Aktuelle Version: **stable-5**

```
(sudo apt-get install -y unzip g{it,cc,++} {c,}make
{zlib1g,lib{sqlite3,curl4-openssl,luajit-5.1,leveldb,pq}}-dev&& \
cd $(mktemp -dp /var/tmp)&&wget
downloads.sourceforge.net/irrlicht/irrlicht-1.8.4.zip -O irr.zip&& \
unzip -q irr.zip&&git clone -b $branch
https://github.com/minetest/minetest&&cd minetest&& \
git clone -b $branch {https://github.com/minetest,games}/minetest_game&& \
cmake . -
D{BUILD_CLIENT=0,{BUILD_SERVER,RUN_IN_PLACE,ENABLE_POSTGRESQL,ENABLE_CURSES}
=1,\
IRRLICHT_INCLUDE_DIR=$PWD/../irrlicht-1.8.4/include,POSTGRESQL_INCLUDE_DIR=/
usr/include/postgresql/,\
POSTGRESQL_LIBRARY=/usr/lib/x86_64-linux-
gnu/libpq.so.5.13,PostgreSQL_TYPE_INCLUDE_DIR=/usr/include/postgresql/}&& \
make -j$(nproc) package&&p=$(echo minetest-*.tar.gz)&&cp "$p" ~&& \
printf '\n\n\e[1;32mBuild successful, an archive called "%s" was placed in
your home folder.\nTo run the server, extract the archive\
and run the executable inside the 'bin/' folder\e[0m\n' "$p")||printf
'\n\n\e[1;31mBuild failed, review log output above to identify and fix the
issue.\e[0m\n
```

Zertifikate:

Die Zertifikate für das Backend wurden mit folgendem Kommando erstellt:

sudo certbot –apache -d minetest.lmz-bw.de

Output:

Saving debug log to /var/log/letsencrypt/letsencrypt.log

Renewing an existing certificate Created an SSL vhost at /etc/apache2/sites-available/000-default-le-ssl.conf Enabled Apache socache_shmcb module Enabled Apache ssl module Deploying Certificate to VirtualHost /etc/apache2/sites-available/000-default-le-ssl.conf Enabling available site: /etc/apache2/sites-available/000-default-le-ssl.conf

Please choose whether or not to redirect HTTP traffic to HTTPS, removing HTTP access. - 1: No redirect - Make no further changes to the webserver configuration. 2: Redirect - Make all requests redirect to secure HTTPS access. Choose this for new sites, or if you're confident your site works on HTTPS. You can undo this change by editing your web server's configuration. - Select the appropriate number [1-2] then [enter] (press 'c' to cancel): 2 Redirecting vhost in /etc/apache2/sites-enabled/000-default.conf to ssl vhost in /etc/apache2/sites-available/000-default-le-ssl.conf

- Your existing certificate has been successfully renewed, and the new certificate has been installed.

The new certificate covers the following domains: minetest.lmz-bw.de

You should test your configuration at: https://www.ssllabs.com/ssltest/analyze.html?d=minetest.lmz-bw.de -

IMPORTANT NOTES: - Congratulations! Your certificate and chain have been saved at: /etc/letsencrypt/live/minetest.lmz-bw.de/fullchain.pem Your key file has been saved at: /etc/letsencrypt/live/minetest.lmz-bw.de/privkey.pem Your cert will expire on 2021-04-07. To obtain a new or tweaked version of this certificate in the future, simply run certbot again with the „certonly“ option. To non-interactively renew *all* of your certificates, run „certbot renew“ - If you like Certbot, please consider supporting our work by:

Donating to ISRG / Let's Encrypt: https://letsencrypt.org/donate Donating to EFF: https://eff.org/donate-le

Diese Zertifikat wurden mit folgenden Anweisungen für Keycloak verwendet (eventuell direkt Pfad anpassen):

https://ordina-jworks.github.io/security/2019/08/14/Using-Lets-Encrypt-Certificates-In-Java.html

https://www.keycloak.org/docs/11.0/server_installation/index.html#_start_cli

openssl pkcs12 -export -in /etc/letsencrypt/live/minetest.lmz-bw.de/fullchain.pem -inkey /etc/letsencrypt/live/minetest.lmz-bw.de/privkey.pem -out /tmp/minetest.lmz-bw.de_2.p12 -name http://minetest.lmz-bw.de/ -CAfile /etc/letsencrypt/live/minetest.lmz-bw.de/fullchain.pem -caname „Let's Encrypt Authority X3“ -password pass:changeit

keytool -importkeystore -deststorepass changeit -destkeypass changeit -deststoretype pkcs12 -srckeystore /tmp/minetest.lmz-bw.de_2.p12 -srcstoretype PKCS12 -srcstorepass changeit -destkeystore /tmp/minetest.lmz-bw.de_2.keystore -alias minetest.lmz-bw.de/%%

Datenmodell

Die Anwendung greift auf Daten aus vier Quellen zu.

Quellenname	Beschreibung
Keycloakdaten	Hier werden alle Daten des Benutzers gespeichert (Name, Schule, Passwort), diese werden im Normalfall nicht durch die Anwendung verändert
Datenbank „Backendata“	Hier werden die zur (Klassenzimmer)verwaltung benötigten Informationen gespeichert
Datenbanken „k**“	Jede Welt besitzt eine eigene Datenbank auf der spielrelevanten Informationen gespeichert werden (z.B. Spielpasswörter, Inventar)
Verzeichnis minetest-live/worlds	Hier werden die nötigen Konfigurationsinformationen zu den Klassenzimmer gespeichert (Mods, Spawnpoint, etc.)

Die Template Datenbank wird als Template markiert und Connections werden verboten, da die Datenbank bei bestehenden Verbindung nicht als Template verwendet werden kann.
Dies haben wir mit den folgenden Commands eingerichtet:

UPDATE pg_database SET datistemplate = true where datname = 'minetest_template1';

UPDATE pg_database SET datallowconn = false where datname = 'minetest_template1';

Dokumentation Tests

Zum Testen des Backends existiert eine Postman Collection mit eingerichteten Tests. Diese können ganz einfach über den Collection-Runner ausgeführt werden (Zugriff muss angefragt werden – Nutzeranzahl leider begrenzt).
Das Frontend muss aktuell noch manuell getestet werden.