

# Sicherheitskonzept

---

## 1 Zusammenfassung

---

smino AG unternimmt hohe Anstrengungen, um die (Daten-)Sicherheit von smino zu gewährleisten.

Die ergriffenen Sicherheitsmassnahmen orientieren sich an geltenden Industriestandards.

smino bietet ein sehr hohes Mass an Sicherheit.

### 1.1 Executive Summary

#### Verfügbarkeit

Die Server von smino stehen Ihnen grundsätzlich 24 Stunden am Tag, 365 Tage im Jahr zur Verfügung. Unser Cloud-Partner, Microsoft Azure, garantiert eine [Ausfallsicherheit von 99.95%](#). Im letzten Jahr stand smino rund 99.977% der Zeit zu Ihrer Verfügung. Das rechnet sich zu etwa 2 Stunden kumulierter Downtime über 365 Tage hinweg.

Wir pflegen eine kontinuierliche Veröffentlichungsstrategie von neuen Funktionen und Verbesserungen, auch "Rolling Release" genannt. Dies führt dazu, dass Sie auch während einer Veröffentlichung ohne Ausfälle oder Unterbrüche auf unserer Plattform weiterarbeiten können. Neue Versionen werden zuerst eingehend auf einem Testsystem geprüft. Sollten sich im Produktionsbetrieb trotzdem Fehler einschleichen, kann die vorherige Version ohne Unterbruch wieder eingespielt werden.

#### Datenablage

Unsere Server, und somit Ihre Daten, befinden sich in Rechenzentren von Microsoft Azure in einer unserer Verfügbaren Region. Ihre Daten werden in diesen Rechenzentren, welche zu den modernsten in ganz Europa zählen, nach dem neuesten Stand der Technik geschützt. Dazu gehören unter anderem physischer Einbruchschutz, Feuerlöschanlagen, Redundanzen, alternative Stromzufuhr und diverse andere Sicherheitsvorkehrungen. Die Rechenzentren sind mit über [40 Normen](#) wie ISO 27001, ISO 27018 und PCI DSS zertifiziert.

Alle Ihre Daten befinden sich in der spezifischen Region, mit der Ausnahme von BIM-Modellen, welche Kunden in unserem 3D-Viewer betrachten wollen. Diese Daten werden zusätzlich auf den Servern unseres Partners Autodesk-Forge gespeichert, welche sich in der EU befinden und den strengen Auflagen der DSGVO unterliegen. Auch dieser Partner ist um den [Schutz Ihrer Daten](#) bemüht.

Wir verschlüsseln jegliche Kommunikation nach dem neustem Sicherheitsstandard. Nur ausgewählte Mitarbeiter haben Zugriff auf Ihre Daten um diese für Sie zu warten und Ihre Supportanfragen zu beantworten. Durch unser ausgeklügeltes Rollensystem sind Ihre Daten von den Daten anderer Kunden klar abgegrenzt und vor einem unerlaubtem Zugriff geschützt. Wir setzen auf "Security by Design", sprich, wir orientieren uns an modernsten Sicherheitsrichtlinien und achten bereits bei der Entwicklung darauf, dass unser System so sicher wie nur möglich

ist. Wir sind bestrebt, auf dem neusten Stand der Technik zu bleiben und feilen so stets weiter an unserem Sicherheitskonzept.

## Backups

Die Datensicherung wird bei smino gross geschrieben. Mithilfe unseres detaillierten Sicherungskonzept werden Ihre Daten fortwährend, täglich, monatlich und jährlich gespiegelt. Diese befinden sich verschlüsselt auf einem Server und sind stets bereit, auf den Produktionsbetrieb zurückgespielt werden zu können.

Sie haben die Möglichkeit, die gesamten Dateien eines Projektes zu exportieren und bei Ihnen lokal zu speichern.

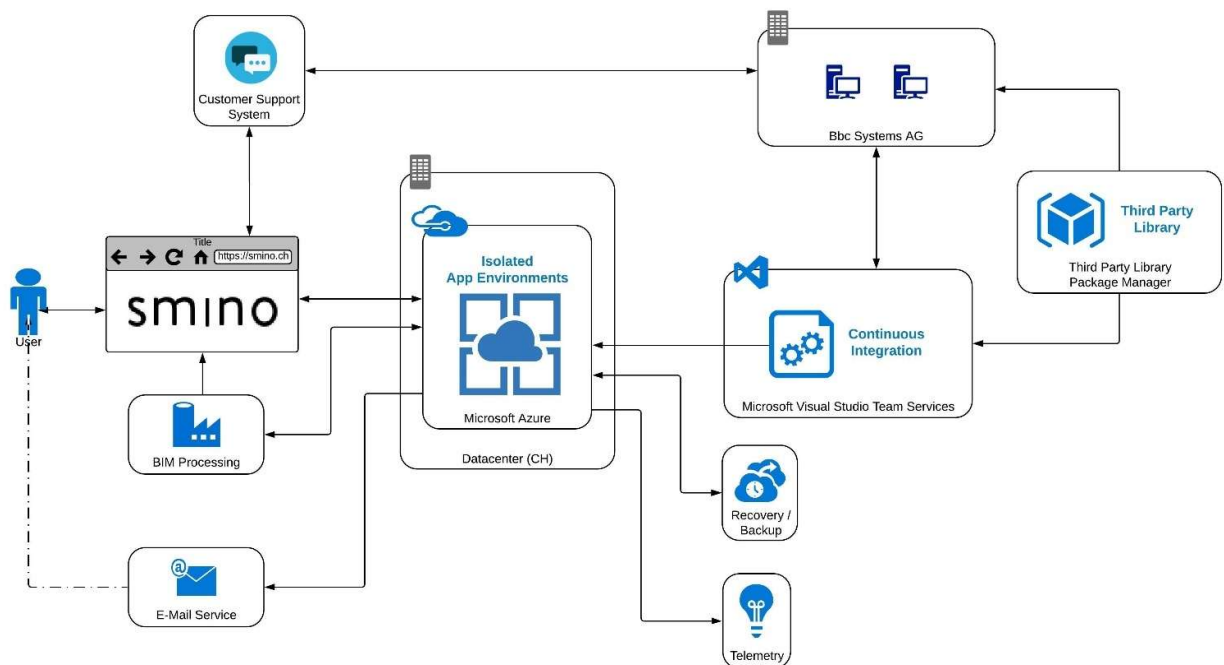
## 1.2 Kurzzusammenfassung

Ihre Daten liegen auf Servern von Microsoft Azure in einer der verfügbaren Regionen. Diese Server sind sehr sicher und haben sehr wenige Ausfälle. Ihre Daten werden in der spezifischen Region gespeichert. Einzig die BIM-Modelle sind zusätzlich noch auf Servern in der EU gespeichert, damit unser 3D-Viewer-Partner, Autodesk-Forge, diese verarbeiten und darstellen kann.

Alle Kommunikation mit unserem Service ist verschlüsselt und nur ausgewählte Mitarbeiter haben Zugriff auf Ihre Daten. Durch ein Rollensystem sind Ihre Daten auch vor einem unberechtigten Zugriff durch andere User geschützt.

Wir machen ein tägliches Backup Ihrer Daten, welches sicher und verschlüsselt aufbewahrt wird. Sie können jederzeit eine Kopie aller Ihrer Daten in einem Projekt herunterladen.

## 2 Übersicht



smino ist eine Web Applikation, welche sich perfekt in die Cloud Plattform von Microsoft Azure integriert. Die von smino genutzten Services, Hardware und Infrastruktur werden durch Microsoft Azure gekapselt, verwaltet, gesichert und in der entsprechenden Region gehostet.

Die Microsoft Azure Plattform bietet einen sehr guten Schutz:

<https://azure.microsoft.com/en-us/services/security-center/>

Diese Architektur setzt den Grundstein für unsere weiteren Sicherheitsmassnahmen.

### 3 Verfügbare Regionen

Smino steht in folgenden, komplett isolierten Regionen zur Verfügung:

Region	Betreiber	Primäres Rechenzentrum	Sekundäres Rechenzentrum	Standards
<b>Schweiz</b> <b>app.smino.ch</b>	Microsoft Azure Switzerland	Zurich (Switzerland North)	Geneva (Switzerland West)	FINMA, TruSight, Shared Assessments, PCI DSS, GxP, CDSA, SOC 3, SOC 2, SOC 1, ISO 9001, ISO 27701, ISO 27018, ISO 27017, ISO 27001, ISO 22301, ISO 20000, CSA STAR Self-Assessment, CSA STAR Certification, CSA STAR Attestation, CIS Benchmark
<b>Deutschland</b> <b>app.smino.de</b>	Microsoft Azure Germany	Frankfurt (Germany West Central)	Berlin (Germany North)	EU-US Privacy Shield, TISAX, IT-Grundschutz, C5, EU Model Clauses, GDPR, ENISA IAF, EN 301 549, EBA, TruSight, Shared Assessments, PCI DSS, GxP, CDSA, SOC 3, SOC 2, SOC 1, ISO 9001, ISO 27701, ISO 27018, ISO 27017, ISO 27001, ISO 22301, ISO 20000, CSA STAR Self-Assessment, CSA STAR Certification, CSA STAR Attestation, CIS Benchmark

### 4 Wo speichert smino Daten

Daten werden grundsätzlich in Rechenzentren von Microsoft Azure in der spezifischen Region gespeichert und verarbeitet.

Ausgenommen davon sind folgende Daten:

- Alle Pläne ausgenommen solche im PDF-Format, welche smino darstellen kann, werden von unserem Partner Autodesk auf Servern im europäischen Ausland gehostet und verarbeitet. Autodesk setzt die unter folgendem Link aufgeführten Massnahmen zur Datensicherheit und Compliance um:
  - <https://www.autodesk.com/company/legal-notices-trademarks/privacy-statement-de>

## 5 Sicherheitsanalyse

### 5.1 Risiken

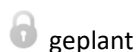
smino ist als Cloud Lösung den bekannten Risiken einer Web Applikation bzw. eines verteilten Systems ausgesetzt.

Um die Risiken zu handhaben und zu minimieren, stützen wir uns auf die Klassifizierungen und Massnahmen des OWASP Konsortiums. Der OWASP Top 10 Gefahrenkatalog ist der defacto Industriestandard für Web Applikationen (siehe <https://www.owasp.org>).

Im Weiteren wurden physische und interne Sicherheitsrisiken am Hauptsitz der smino AG identifiziert. Durch Zuhilfenahme verschiedener Ressourcen wie z.B. den ISO 9001 / 27001 Normen schätzen wir diese Risiken ein und ergreifen entsprechend Massnahmen.

### 5.2 Einschätzung

OWASP	Risiko	smino
A1	Fehler in der Zugriffskontrolle	  
A2	Kryptographische Fehler	  
A3	Injection	  
A4	Unsicheres Design	  
A5	Sicherheitsrelevante Fehlkonfiguration	  
A6	Nutzung von veralteten Komponenten oder mit bekannten Schwachstellen	  
A7	Fehler bei der Identifizierung und Authentifizierung	  
A8	Software- und Datenintegritätsmängel	  
A9	Sicherheitsprotokollierung und Überwachung von Fehlern	  
A10	Server-seitige Anforderungsfälschung (SSRF)	  
-	Interne IT Sicherheit	  
-	Interne physische Sicherheit	  
-	Recovery und Backup	  
-	Sicherheitsanalyse	  



## 6 Massnahmenkatalog

---

### 6.1 Technische Sicherheit

#### 6.1.1 A1: Fehler in der Zugriffskontrolle

Häufig werden die Zugriffsrechte für authentifizierte Nutzer nicht korrekt um- bzw. durchgesetzt. Angreifer können entsprechende Schwachstellen ausnutzen, um auf Funktionen oder Daten zuzugreifen, für die sie keine Zugriffsberechtigung haben. Dies kann Zugriffe auf Accounts anderer Nutzer sowie auf vertrauliche Daten oder aber die Manipulation von Nutzerdaten, Zugriffsrechten etc. zur Folge haben.

##### *Ergriffene Massnahmen*

- Feingranulares Autorisierungsprinzip für jegliche Ressourcen, welches bereits in der Designphase erarbeitet wird
- Kein externer Zugriff auf das File System möglich
- Nicht sequentielle, nicht erratbare GUIDs als Objektschlüssel
- Jeder Endpunkt durch automatisierte Tests auch hinsichtlich Autorisierung gesichert
- Tests um sicherzustellen, dass das Autorisierungsprinzip richtig funktioniert.

#### 6.1.2 A2: Kryptographische Fehler

Viele Anwendungen schützen sensible Daten, wie personenbezogene Informationen und Finanz oder Gesundheitsdaten, nicht ausreichend. Angreifer können diese Daten auslesen oder modifizieren und mit ihnen weitere Straftaten begehen (Kreditkartenbetrug, Identitätsdiebstahl etc.). Vertrauliche Daten können kompromittiert werden, wenn sie nicht durch Maßnahmen, wie Verschlüsselung gespeicherter Daten und verschlüsselte Datenübertragung, zusätzlich geschützt werden. Besondere Vorsicht ist beim Datenaustausch mit Browsern angeraten.

##### *Ergriffene Massnahmen*

- Kein unnötiges Speichern vertraulicher Daten
- Aktuelle, starke Algorithmen und Schlüsselmanagement verwendet
- Nutzung von Transportverschlüsselung mit sicheren Protokollen auf allen Ebenen
- Keine Caches für sensible Daten
- Passwörter mit einem adaptiven Salting-Hash Algorithmus mit hohem Rechenaufwand gespeichert

#### 6.1.3 A3: Injection

Injection-Schwachstellen, wie beispielsweise SQL-, OS- oder LDAP-Injection, treten auf, wenn nicht vertrauenswürdige Daten von einem Interpreter als Teil eines Kommandos oder einer Abfrage verarbeitet werden. Ein Angreifer kann Eingabedaten dann so manipulieren, dass er nicht vorgesehene Kommandos ausführen oder unautorisiert auf Daten zugreifen kann. Injection kann auch in Form von XSS (Cross-Site-Scripting) auftreten. XSS tritt auf, wenn Anwendungen nicht vertrauenswürdige Daten entgegennehmen und ohne Validierung oder Umkodierung an einen Webbrowser senden. XSS tritt auch auf, wenn eine Anwendung HTML- oder JavaScript-Code auf Basis von Nutzereingaben erzeugt. XSS erlaubt es einem Angreifer, Scriptcode im Browser eines Opfers auszuführen und so Benutzersitzungen zu übernehmen, Seiteninhalte verändert anzuzeigen oder den Benutzer auf bösartige Seiten umzuleiten.

### *Ergriffene Massnahmen*

- Strikte Trennung von Eingaben und Befehlen
- Einsatz von etablierten und sicheren Libraries bei jeder Art von Userinput, welche Benutzereingaben nicht direkt entgegennehmen, sondern zuerst auf unerlaubte Muster untersuchen und ggf. entfernen
- Konfiguration nach Umgebungsvariablen
- Keine sensitiven Konfigurationsdateien im System oder der Versionsverwaltung
- Kein dynamisches Generieren von Datenbankstrukturen anhand von User-Eingaben
- Strikte Validierung von User Input auf mehreren Levels durch bewährte und etablierte Libraries
- Einsatz von Best Practices im Umgang mit User generierten Daten (z.B. Sanitizing)
- Setzen der für XSS relevanten HTTP-Header
- Detaillierte CSP-Policy eingerichtet

#### 6.1.4 A4: Unsicheres Design

Anwendungen können bereits vor der Implementierung unsicher designed werden und weisen somit Schwachstellen auf. Unsicheres Design ist nicht die Ursache für jede andere der zehn verschiedenen Kategorien. Weist die Anwendung ein sicheres Design auf, kann die Implementierung immer noch fehlerhaft durchgeführt werden. Ein unsicheres Design kann aber hingegen nicht durch eine perfekte Implementierung behoben werden. Einer der Faktoren, die zu einem unsicheren Entwurf beitragen, ist die fehlende Erstellung eines Geschäftsrisikoprofils für die zu entwickelnde Software oder das zu entwickelnde System und damit das Versäumnis, das erforderliche Sicherheitsniveau zu bestimmen.

### *Ergriffene Massnahmen*

- Dokumentation von Design Entscheidungen
- Limitierung von Ressourcen
- Reviews von Designentscheidungen und Code
- Integration von Sicherheitsgedanken in User Stories

#### 6.1.5 A5: Sicherheitsrelevante Fehlkonfiguration

Fehlkonfigurationen von Sicherheitseinstellungen sind das am häufigsten auftretende Problem. Ursachen sind unsichere Standardkonfigurationen, unvollständige oder ad-hoc durchgeführte Konfigurationen, ungeschützte Cloud-Speicher, fehlkonfigurierte HTTP-Header und Fehlerausgaben, die vertrauliche Daten enthalten. Betriebssysteme, Frameworks, Bibliotheken und Anwendungen müssen sicher konfiguriert werden und zeitnah Patches und Updates erhalten.

### *Ergriffene Massnahmen*

- Regelmässige Schwachstellen Audits
- Definition von sicheren Default Setups für alle wichtigen Komponenten
- Konfiguration über sichere Umgebungsvariablen
- Keine sensitiven Konfigurationsdateien in der Applikation oder der Versionsverwaltung
- Sicherheitsdirektiven wie Headers werden an alle Clients gesendet
- Verwendung von TLS
- Nur die nötigsten Features werden aktiviert, um die Angriffsfläche zu vermeiden
- Kein Anzeigen von Standardfehlermeldungen mit Serverinformationen gegenüber dem Benutzer

### 6.1.6 A6: Nutzung von veralteten Komponenten oder solchen mit bekannten Schwachstellen

Komponenten wie Bibliotheken, Frameworks etc. werden mit den Berechtigungen der zugehörigen Anwendung ausgeführt. Wird eine verwundbare Komponente ausgenutzt, kann ein solcher Angriff von Datenverlusten bis hin zu einer Übernahme des Systems führen. Applikationen und APIs, die Komponenten mit bekannten Schwachstellen einsetzen, können Schutzmaßnahmen unterlaufen und so Angriffe mit schwerwiegenden Auswirkungen verursachen.

#### *Ergriffene Massnahmen*

- Definition von Prozessen für das Verwenden und Aktualisieren von Abhängigkeiten mit dem Ziel, dass keine unsicheren Abhängigkeiten in unserem System vorkommen. Diese Prozesse sind fest in unser Projektmanagement und den Entwicklungsprozess integriert und werden alle 2 Wochen durchgeführt.
- Geregelte Zuständigkeiten für das Überwachen von Abhängigkeiten
- Regelmässige Analyse aller Abhängigkeiten
- Regelmässige Security Audits aller Abhängigkeiten

### 6.1.7 A7: Fehler bei der Identifizierung und Authentifizierung

Anwendungsfunktionen, die im Zusammenhang mit Authentifizierung und Session-Management stehen, werden häufig fehlerhaft implementiert. Dies erlaubt es Angreifern, Passwörter oder Session-Token zu kompromittieren oder die entsprechenden Schwachstellen so auszunutzen, dass sie die Identität anderer Benutzer vorübergehend oder dauerhaft annehmen können.

#### *Ergriffene Massnahmen*

- Benutzung und Implementierung von Best Practices und Prozessen in diesem Bereich
- Zusammenarbeit mit etablierten Anbietern und Libraries sowie der Einsatz bewährter Technologien
- Libraries von Dritten werden alle 2 Wochen nach sicherheitsrelevanten Updates überprüft und ggf. aktualisiert
- Alle Kommunikationskanäle (ausser E-Mails und die Anbindung an manche Druckereien) verschlüsselt
- Definiertes TLS & Zertifikat Management nach Best Practices, sicheres Sessionmanagement durch Nutzung von JSON Web Tokens (JWT) mit etablierten Libraries

### 6.1.8 A8: Software- und Datenintegritätsmängel

Unsichere, weil unzureichend geprüfte Deserialisierungen können zu Remote-Code-Execution-Schwachstellen führen. Aber auch wenn das nicht der Fall ist, können Deserialisierungsfehler Angriffsmuster wie Replay-Angriffe, Injections und Erschleichung erweiterter Zugriffsrechte ermöglichen.

#### *Ergriffene Massnahmen*

- Keine Akzeptanz serialisierter Objekte von unsicheren Quellen
- Serialisierung durch etablierte Libraries wird nur zum Transport von einfachen Data Transfer Objekten (DTOs) eingesetzt
- Serialisierungen setzen sich aus primitiven Datentypen zusammen
- Type-Checking nach Serialisierung durch etablierte Libraries
  - Keine dynamische Typangabe für Deserialisierung durch User

- Einsatz von bewährten Libraries zu Serialisierung / Deserialisierung
- Reviewen von Konfigurations- und Source Code Änderungen

#### 6.1.9 A9: Sicherheitsprotokollierung und Überwachung von Fehlern

Unzureichendes Logging und Monitoring führt zusammen mit fehlender oder uneffektiver Reaktion auf Vorfälle zu andauernden oder wiederholten Angriffen. Auch können Angreifer dadurch in Netzwerken weiter vordringen und Daten entwenden, verändern oder zerstören. Viele Studien zeigen, dass die Zeit bis zur Aufdeckung eines Angriffs bei ca. 200 Tagen liegt sowie typischerweise durch Dritte entdeckt wird und nicht durch interne Überwachungs- und Kontrollmaßnahmen.

##### *Ergriffene Massnahmen*

- Alle erfolglosen Login- und Zugriffsversuche werden mit aussagekräftigem Benutzerkontext protokolliert und über einen längeren Zeitraum aufbewahrt
- Protokollierungen werden in einem Format gespeichert, welches leicht mit Proktokollanalysetools und Managmentwerkzeugen durchsucht und analysiert werden kann
- Für wichtige Geschäftsvorgänge werden dedizierte Auditeinträge angelegt

#### 6.1.10 A10: Server-seitige Anforderungsfälschung (SSRF)

SSRF-Schwachstellen treten immer dann auf, wenn eine Webanwendung eine Remote-Ressource abruft, ohne die vom Benutzer angegebene URL zu validieren. Dadurch kann ein Angreifer die Anwendung dazu zwingen, eine manipulierte Anfrage an ein unerwartetes Ziel zu senden, selbst wenn sie durch eine Firewall, ein VPN oder eine andere Art von Netzwerk-Zugriffskontrollliste (ACL) geschützt ist.

Da moderne Webanwendungen den Endbenutzern bequeme Funktionen bieten, wird das Abrufen einer URL zu einem häufigen Szenario. Infolgedessen nimmt das Auftreten von SSRF zu. Außerdem wird der Schweregrad von SSRF aufgrund von Cloud-Diensten und komplexen Architekturen immer größer.

##### *Ergriffene Massnahmen*

- Bereinigung und Validierung aller vom Kunden bereitgestellten Eingabedaten
- Kein Senden von «rohen» Antworten an den Client
- Massnahmen gegen eine DNS Rebinding Attacke (HTTPS, nur moderne Browser werden von smino unterstützt)

#### 6.1.11 Interne IT Sicherheit

Die Firmeninterne IT ist gegenüber gezielten oder zufälligen Attacken zu schützen.

##### *Ergriffene Massnahmen*

- Antivirus / Antimalware Schutz an allen Arbeitsplätzen
- Firewalls schützen das interne Netzwerk
- Mitarbeiter sind gegenüber Phishing sensibilisiert
- Mitarbeiter sind gegenüber Social Hacking sensibilisiert

## 6.2 Interne physische Sicherheit

Die Immobilien von smino AG enthalten sensitive Ressourcen und sind deshalb gegenüber gezielten oder zufälligen Attacken zu schützen.

### *Ergriffene Massnahmen*

- Password Policy
- Einsatz von Password Tools
- Physischer Zugang ins Gebäude nur für erlaubte Personen möglich
- Automatischer Screen Lock

## 6.3 Backups und Recovery

Sei es durch menschliche Fehler, versagen der Hardware oder gar durch Angreifer ausgelöst; ein ausgereiftes Konzept für das Datenbackup sowie dem Wiedereinspielen der Daten sollte für jedes IT-Unternehmen zu den obersten Prioritäten gehören. Zu den wichtigsten Überlegungen gehört, wie lange das eine vollständige Recovery nach einem Datenausfall dauert. Ausserdem muss bestimmt werden, wie fein granular die Daten gebackupt werden. Die Sicherheit der Backups selbst muss auch evaluiert werden, zudem müssen die Prozesse ausgearbeitet werden, die im Ernstfall zur Anwendung kommen.

### *Ergriffene Massnahmen*

- Genaue Definition, welche Daten wie oft gesichert werden. Die Zeiten richten sich nach der Wichtigkeit der Aktualität der Daten, welche wir feingranular festgelegt haben.
  - Dies ermöglicht eine automatisierte Lösung, die nicht durch menschliche Fehler beeinträchtigt werden kann.
- Die Transportwege zwischen den Live-Daten und dem Backup Ort werden mit einer starken Verschlüsselung versehen.
- Alle Backups befinden sich auf anderen physischen Geräten als die Live-Daten.
- Automatisierte Prüfung der Validität aller Backups sowie das automatische Wiederherstellen über Skripte werden regelmässig durchgeführt.

## 6.4 Sicherheitsanalyse

Bestehende Prozesse müssen regelmässig überprüft und gegebenenfalls optimiert werden. Ebenso muss auf Änderungen der Bedrohungslage schnell reagiert werden können.

### *Ergriffene Massnahmen*

- Jährliche Anpassungen der geltenden Sicherheitsrichtlinien entsprechend der gegenwertigen Bedrohungslage
- Priorisierte Reaktion auf wichtige neue Erkenntnisse bezüglich potentiellen Schwachstellen
- Regelmässige Risikoanalyse und Sicherheitsreviews sind feste Bestandteile des Entwicklungsprozesses

## 7 Anhang: Konkreter Massnahmenkatalog

### 7.1 A1: Fehler in der Zugriffskontrolle

Action	Status
Quarantine user contexts, fileuploads or similar should not be vulnerable for directory surfing or similar.	Done
Always use random, unguessable IDs such as Guids as references to objects	Done
Have a fine granular rights approach towards resources with roles and also user identities	Done
Keep Authorization Grant mechanism simple and in one place	Done
Always make authorization checks on the server side	Done
Make sure in code reviews, that when requirements have changed, that authorization mechanisms are updated	Done
Maybe keep a matrix with API endpoints and needed rights for documentation but also to consult in case of a security review	Done
Are we vulnerable when processing word and other files? Keep those libraries updated	Done, server-side virus scanning of documents not yet planned
Disable web server directory listing and ensure file metadata and backup files are not present within web roots	Done
Log access control failures	Done
JWT tokens should be invalidated on the server after logout	Done
Unique application business limit requirements should be enforced by domain models	Done
Rate limit API and controller access to minimize the harm from automated attack tooling	Not planned yet
Expect for public resources, deny by default	Done
Stateless JWT tokens should rather be short-lived so that the window of opportunity for an attacker is minimized. For longer lived JWTs it's highly recommended to follow the OAuth standards to revoke access.	Done
Create Unit and Integration Test Cases for Authorization Logic	Unit tests written, integration tests planned

## 7.2 A2: Kryptographische Fehler

### 7.2.1 Sensitive data protection

Action	Status
Password Storing: Generate unique salt per password, store hashed and salted	Done
Don't store sensitive data unnecessarily	Done
Use strong keys and encryption	Done
Classify data stored, processed or transmitted by an application. Identify which data is sensitive according to privacy laws, regulation requirements or business needs	Done
Apply controls as per the classification	Done
Make sure to encrypt all sensitive data at rest	Done
Ensure up-to-date and strong standard algorithms, protocols, and keys are in place; use proper management	Done
Encrypt all data in transit with secure protocols such as TLS with perfect forward secrecy ciphers, cipher prioritization by the server, and secure parameters. Enforce encryption using directives like HTTP Strict Transport Security (HSTS)	Done
Disable caching for responses that contain sensitive data	Done
Store passwords using strong adaptive and salted hashing functions with a work factor (delay factor), such as Argon2, scrypt, bcrypt, or PBKDF2.	Done
Verify independently the effectiveness of configuration and settings.	As soon as resources permit done via security audit
Encourage blueprint printshops to use a strong certificate	Continuously done but many printshops don't want to adjust their infrastructure
Document contingency plans if a key is compromised	Done
Always encrypt backups of all kinds	Done
Document contingency plans if a key is compromised	Done
Identify data that should also be encrypted in the database	Done
Keys should have a defined lifecycle. This includes specifying how they are generated, distributed, stored, used, replaced, updated, revoked and deleted	Done
Always store keys away from the data. It probably goes without saying, but if the very piece of information which is required to unlock the encrypted data – the key – is conveniently located with the data itself, a data breach will likely expose even encrypted data.	Done
Protect the keys. Once a key is disclosed, the data it protects can be considered as good as open.	Done

Keep keys unique. Some encryption attack mechanisms benefit from having greater volumes of data encrypted with the same key. Mixing up the keys is a good way to add some unpredictability to the process.	Done
Make sure data encryption protection is isolated from other Access Control Mechanisms (if a user gains access to the database, he should still not be able to read encrypted data)	Done
Avoid deprecated cryptographic functions and padding schemes, such as MD5, SHA1, PKCS number 1 v1.5.	Done
Do not use legacy protocols such as FTP and SMTP for transporting sensitive data.	Done everywhere except some printshops don't want to adjust their infrastructure to SFTP, when transmitting print jobs

## 7.2.2 TLS Configuration

Action	Status
HTTPS only -> Serve all content over HTTPS -> includes Css, scripts, images, ajax, third parties etc	Done
Set HTTP Strict Transport Security Header (HSTS)	Done
Use HTTPS everywhere, between nodes, within azure etc	Done
Do never mix TLS and non-TLS content	Done
No sensitive data in url	Done
Use well-guarded certificate	Done
Only support strong Protocols and strong CryptoSuite	<b>Done (grade A+ on SSL Labs)</b>
Only allow secure renegotiations as with RFC 5746 or none at all	Done
Disable compression	Done
Enforce TLS on all three levels (Infrastructure, Tokenmanagement, In Code)	Done
Absolutely no HTTP content from third parties	Done
Harden HTTP-Headers	Done
Use Extended Validation Certificates	Planned

## 7.3 A3: Injection

### SQL Injection

Action	Status
SQL Server: isolated?	Done
SQL Connection: Encrypted?	Done
SQL Server: Does it run on low system privileges?	Done
Connection String: is stored encrypted and as Env-variable	Done
Config Files: Encrypted and sensitive values stored as Env-variable	Done
Config Files: Read/Write Permissions Set right	Done
DB Access -> Do we run queries with a DB user that has adequate rights?	Done
All queries setup using ORM or parameterized queries	Done
No "Exec-Commands with arbitrary inputs" anywhere	Done
Use library calls rather than external processes to recreate desired functionality	Done
Ensure all external commands called from program are statically created	Done

### Cross-Site Scripting (XSS)

Action	Status
Angular sanitizes everything in the frontend, every attribute, text and the works; never build templates out of user generated parts	Done
TinyMCE does a really good job in preventing XSS -> They do regular security audits	Done
When we send blocks of json with the initial page load, always make sure that the content-type is "application/json" and never "text/html" but we probably will never need to do this anyway	Done
Set Content-Security-Policy Header	Done
Do not use CDNs for scripts except for really well trusted partners	Done
Never modify the DOM directly but always through angular templates and we should be good	Done
If there ever is reason to do DOM-Manipulation directly, make sure we implement proper sanitation techniques and use correct DOM-API (e.g. innerText instead of innerHTML)	Done
Set X-Frame Options Header	Done
Set X-XSS-Protection Header	Done

## 7.4 A4: Unsicheres Design

Action	Status
Limit resource consumption by user or service	Done
Write unit and integration tests to validate that all critical flows are resistant to the threat model	Unit tests written, integration tests planned
Establish and use a secure development lifecycle to help evaluate and design security and privacy-related controls	Done
Integrate security language and controls into user stories	Done
Document design decisions	Done
Segreate tier layers on the system and network layers depending on the exposure and protection needs	Done
Ensure business logic on all layers (frontend and backend)	Done
Use Secure Design Principles	Done
Client-Side Enforcement of Server-Side Security	Done

## 7.5 A5: Sicherheitsrelevante Fehlkonfiguration

Action	Status
Turn off all unnecessary features by default	Done
Inspect all tools and component for default passwords, change them immediately after deploying them	Done
Do not hardcode any backdoor account	Done
Encrypt Config Files that for instance contain connection strings	Done
Use TLS	Done
Only Store PWDs hashed	Done
Do not store credit cards	Done
Never store an encryption key on the same server as the encrypted data	Done
Use well defined error handling strategies to prevent information leakage	Done
Prevent unpatched security flaws in the server software	Done
Prevent server software flaws or misconfigurations that permit directory listing and directory traversal attacks	Done
Prevent unnecessary default, backup, or sample files, including scripts, applications, configuration files, and web pages	Done
Prevent improper file and directory permissions	Done
Prevent unnecessary services enabled, including content management and remote administration	Done
Prevent default accounts with their default passwords	Done

Prevent overly informative error messages (more details in the error handling section)	Done
Prevent misconfigured SSL certificates and encryption settings	Done
Don't use of default certificates	Done
Prevent improper authentication with external systems	Done
Do all DB-Access, Webapp accesses etc. run on least priviledges	Done
Keep all used components, frameworks, libraries etc. up to date and scan change logs for vulnerabilities. Set up processes on how this is done	Done
Encrypt all configuration	Done
<ul style="list-style-type: none"> <li>• Develop hardening guideline for the environment we will use, including configuration of all security mechanisms</li> <li>• Turning off all unused services</li> <li>• Setting up roles, permissions, and accounts, including disabling all default accounts or changing their passwords</li> <li>• Logging and alerts</li> </ul>	Partly done, definitive hardening guideline under development
<ul style="list-style-type: none"> <li>• Set up Processes that detail how the following points are done</li> <li>• Monitoring the latest security vulnerabilities published</li> <li>• Applying the latest security patches</li> <li>• Updating the security configuration guideline</li> <li>• Regular vulnerability scanning from both internal and external perspectives</li> <li>• Regular internal reviews of the server's security configuration as compared to your configuration guide</li> <li>• Regular status reports to upper management documenting overall security posture</li> </ul>	Partly done, resources do not permit external audits yet, otherwise done
Disable debugging in production and also maybe public facing test servers	Done
Use an automated process to verify the effectiveness of the configurations and settings in all environments	Done
Use custom error page	Done
Prevent including testing code in the production code	Done

## 7.6 A6: Nutzung von veralteten Komponenten oder mit bekannten Schwachstellen

Action	Status
Keep a list of all dependencies, used frameworks and the environment stack with detailed information such as version number	Done
Define a process that guides the programmer when adding a new dependency	Done
When adding a dependency, it should fulfill the following points: <ul style="list-style-type: none"> <li>• dependency must be actively maintained</li> <li>• dependency binaries must be generated directly from project source code using trusted tools</li> <li>• dependency with known vulnerabilities must be removed or updated as soon as possible after vulnerability announcement</li> </ul>	Done
Add as few dependencies as needed, but as many as beneficial	Done
Define a strict responsibility scheme on who supervises which dependency This includes <ul style="list-style-type: none"> <li>• Checking them in a regular interval</li> <li>• Subscribe to newsletters or similar of big dependencies</li> <li>• When an update is available, check the changelogs, if there were security vulnerabilities, update immediately and put all feature development on halt until the update is implemented</li> <li>• If the changelog does not contain security vulnerabilities, it still must be implemented but can follow a leaner policy (see point below)</li> </ul>	Done
Define a process that details in which timespan and how an update of a dependency should be implemented. This includes testing and checking for compatibility with other dependencies	Done
Automated vulnerability discovery must be part of any continuous delivery process <ul style="list-style-type: none"> <li>• Automated inventory of existing components, libraries and frameworks</li> <li>• Monitor usage of deployed components</li> </ul>	Done

## 7.7 A7: Fehler bei der Identifizierung und Authentifizierung

### 7.7.1 Username, Emailadress and Password

Action	Status
Username: Keep usernames unique	Done
Check emails for validity	Done
Make sure user has access to email-account and verifies it	Done
Password: Use minimum 10 characters and enforce complexity like upper/lower, special char and numbers	Done
Password: show user password strength	Done
Password: show user Tipps to strengthen his password	Done
Compare passwords to list of known weak passwords	Not Planned because of UX
Require Re-authentication for Sensitive Features E.G username and pwd changes	Done
Limit or increasingly delay failed login attempts. Log all failures.	Done
use standard HTML forms for username and password input with appropriate `type` attributes,	Done
do not artificially limit user passwords to a length "reasonable for humans" and allow passwords lengths up to 128 characters	Done
do not artificially prevent copy and paste on username and password fields,	Done
avoid plugin-based login pages (Flash, Silverlight etc)	Done
Turn off autocompletion on session related fields (pwd etc)	Done
2 Factor Authentication	Planned
Registration, Credentials recovery and API pathways are hardened against account enumeration attacks by using the same messages for all outcomes	Done
Session IDs should not be in the URL, be securely stored and invalidated after logout, idle, and absolute timeouts	Done
Ship the product with strong passwords	Does not apply to smino
Require Javascript and block headless browsers	Planned
Notify users about unusual security events	Planned

### 7.7.2 Session Management General Guidelines

Action	Status
Never reuse session IDs	Done
The name used by the session ID should not be extremely descriptive nor offer unnecessary details about the purpose and meaning of the ID.	Done
The session ID must be long enough to prevent brute force attacks, where an attacker can go through the whole range of ID values and verify the existence of valid sessions	Done
The session ID length must be at least 128 bits (16 bytes)	Done
The session ID must be unpredictable (random enough) to prevent guessing attacks, where an attacker is able to guess or predict the ID of a valid session through statistical analysis techniques. For this purpose, a good PRNG (Pseudo Random Number Generator) must be used	Done
The session ID value must provide at least 64 bits of entropy (if a good PRNG is used, this value is estimated to be half the length of the session ID).	Done
The session ID content (or value) must be meaningless to prevent information disclosure attacks, where an attacker is able to decode the contents of the ID and extract details of the user, the session, or the inner workings of the web application.	Done

## 7.8 A8: Software- und Datenintegritätsmängel

Action	Status
Do not accept serialized objects from unsafe sources	Done
Implemented integrity checks such as digital signatures on any serialized object	Not affected
Enforce strict type constraints during deserialization before object creation	Done
Isolate and run code that deserializes in low privilege environments if possible	Not affected
Log deserialization failures and exceptions such as when the incoming type is not the expected type	Done
Do not deserialize into user supplied types	Done
Ensure that the CI/CD pipelines has proper segregation, configuration and access control	Done
Review code changes	Done
Review configuration changes	Done
Scan software for security flaws, to verify that components do not contain known vulnerabilities	Partly done, additional automation planned

## 7.9 A9: Sicherheitsprotokollierung und Überwachung von Fehlern

Action	Status
Ensure all login, access control failures, and server-side input validation failures can be logged with sufficient user context to identify suspicious or malicious accounts, and held for sufficient time to allow delayed forensic analysis.	Done
Ensure that logs are generated in a format that can be easily consumed by a centralized log management solution.	Done
Ensure high-value transactions have an audit trail with integrity controls to prevent tampering or deletion, such as append-only database tables or similar.	Done
Establish effective monitoring and alerting such that suspicious activities are detected and responded to in a timely fashion.	Planned
Establish or adopt an incident response and recovery plan, such as NIST 800-61 rev 2 or later.	Done
Ensure log data is encoded correctly to prevent injections or attacks on the logging or monitoring systems.	Done
DevSecOps teams should establish effective monitoring and alerting such that suspicious activities are detected and responded to quickly.	Planned
Do not log sensitive information. For example, do not log password, session ID, credit cards, or social security numbers	Done
Forward logs from distributed systems to a central, secure logging service. This will sure log data cannot be lost if one node is compromised	Done
Periodic Penetration testing and scans by dynamic application security testing (DAST) tools (such as OWASP ZAP) of the application	Not planned

## 7.10 A10: Server-seitige Anforderungsfälschung (SSRF)

Action	Status
Sanitize and validate all client-supplied input data	Done
Enforce the URL schema, port and destination with a positive allow list	Done
Do not send raw responses to clients	Done
Disable HTTP redirections	Done
Be aware of the URL consistency to avoid attacks such as DNS rebinding and "time of check, time of use" (TOCTOU) race conditions	Done
Segment remote resource access functionality in separate networks to reduce the impact of SSRF	Done
For frontends with dedicated and manageable user groups use network encryption (e.g. VPNs) on independent systems to consider very high protection needs	Not affected

## 7.11 Interne Sicherheit

Action	Status
Antivirus / -malware software installed	Done
Company Network secured with Firewalls	Done
Employees sensitized to phishing and social hacking, process defined for new employees	Done

## 7.12 Interne physische Sicherheit

Action	Status
Password policy defined	Done
Use of password tools to protect and generate passwords	Done
Only employees with a key can physically enter the company facilities	Done
Screen lock policy defined	Done
Automatic screen lock on workstations	Done
Workstations use encrypted file storage	Done
Physical access to network components (e.g. switches) additionally secured	Done

## 7.13 Backups und Recovery

Action	Status
Defined which data gets backed up how often	Done
Encrypted backups and transportation to backup destination	Done
Backup stored at different physical location than production data	Done
Monthly validation and recovery test for backups	Done
Manual emergency drills regarding backup recovery after total data loss are practiced with every technician on a half yearly basis	As soon as resources permit
Choose and test concrete recovery strategy that is compliant with our SLAs in production environment	Done

## 7.14 Sicherheitsanalyse

Action	Status
Initial security concept and analysis	Done
Continuous security review and risk analysis integrated as part of the product development process	Done
Process defined and started to update and review the security status and concept on a yearly base	Done
External security audit (e.g. by Compass Security Network Computing AG)	As soon as resources permit

