Password-Manager mit Verschlüsselung

Im Projekt wurde ein sicherer Password-Manager erstellt, basierend auf PHP, MySQL, HTML und diversen Libraries. Zur erweiterten Sicherheit werden die Anmelde-Passwörter nur als Salted-Hash abgespeichert. Die abgelegten Konto-Passwörter werden mit AES128 verschlüsselt, als Key dient das Passwort des angemeldeten Benutzers. Dadurch kann niemand anderes als der Benutzer die Passwörter entschlüsseln. Beim ändern des Anmeldepassworts (im Frontend über „Settings“) werden die Konto-Informationen mit dem neuen Passwort neu verschlüsselt.

Durch die Verwendung von etablierten Frameworks wird eine hohe Wiederverwendung erreicht, und der Code kann schlank und übersichtlich gehalten werden. Gerade die Frameworks „Slim“ und „Twig“ helfen dabei, den Code für HTML, PHP und SQL klar zu trennen.

# Architektur-Übersicht

Die Applikation wird ähnlich dem MVC-Framework in mehrere Komponenten aufgeteilt. Diese heissen hier **Model**, **Persistence**, **View** und **Controller**.

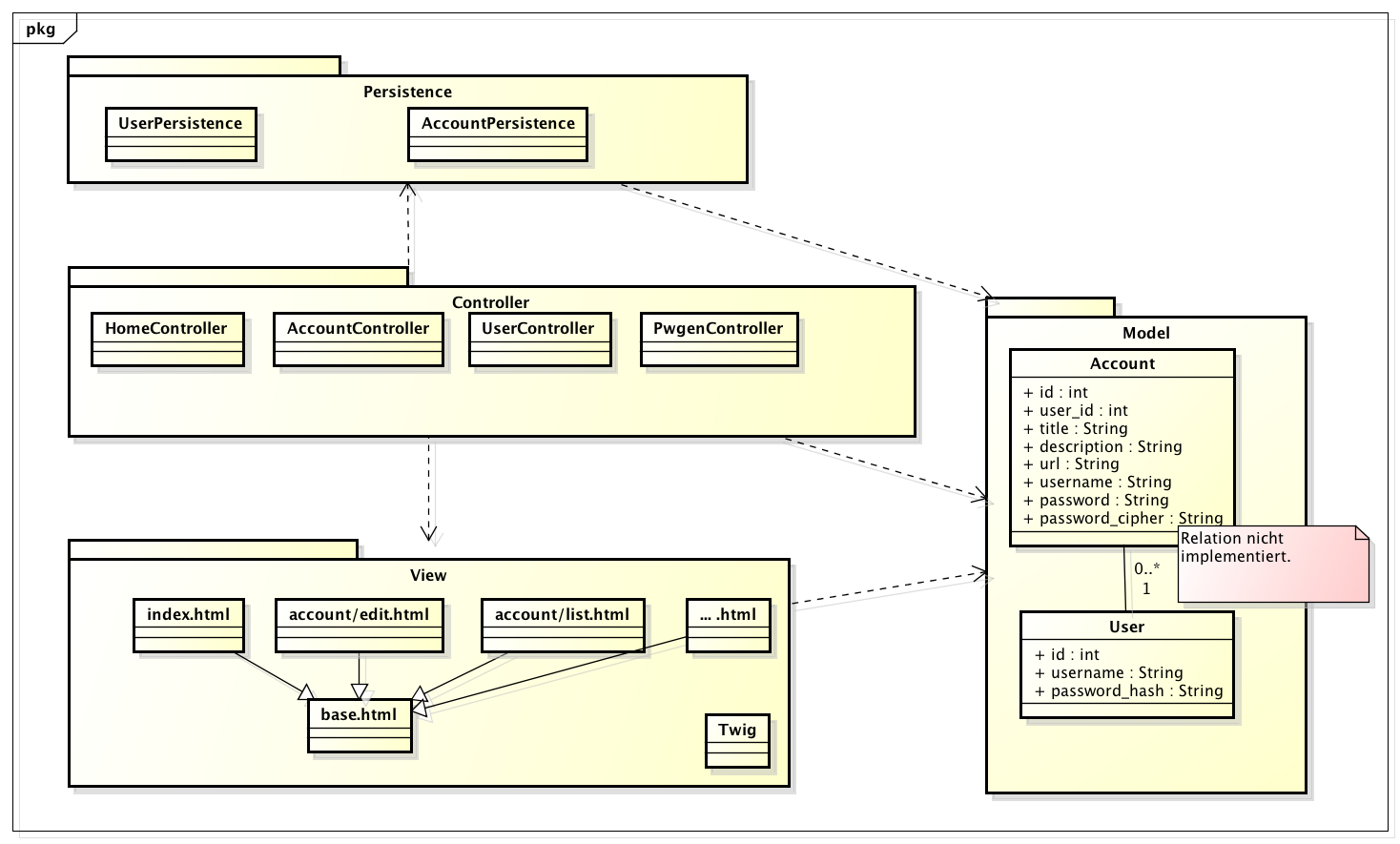


Abbildung : Die Komponenten sind strikt unterteilt, die Pfeile zeigen die Zugriffe

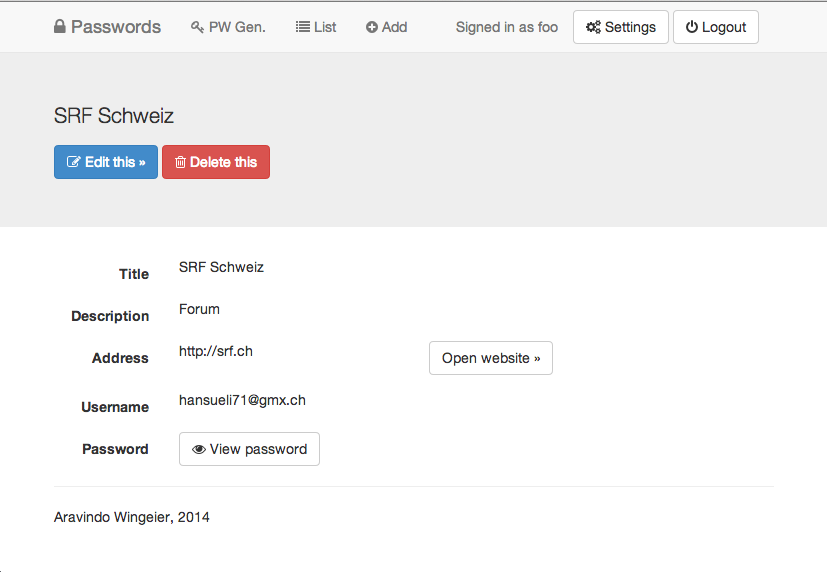
Das **Model** enthält die Daten, welche in allen Komponenten verwendet werden. Dies sind User (Anmeldeinformationen) und Account (gespeicherte Information).

Die **Persistence** stellt Funktionen zum persistieren, löschen, ändern und lesen von Model-Objekten aus der Datenbank zur Verfügung. Dadurch wird die Datenbank rein in diesem Packet abstrahiert.

Die **View** enthält HTML-Templates für die Präsentation im Browser. Die Templates werden mit dem Framework „Twig“ aufbereitet, welches Scripting unterstützt und die kompilierten HTML-Dateien für schnellen Zugriff cached. Die View könnte bei bedarf komplett ausgetauscht werden, da es keine Abhängigkeiten auf andere Komponenten hat.

Die **Controller** sind die Einstiegspunkte in die Applikation und kontrollieren den ganzen Ablauf. Sie laden über die Persistence ein Model, verarbeiten die Daten und rufen eine View auf.

# Frontend

Das Frontend wurde mit Bootstrap und JQuery umgesetzt, um eine dynamische und moderne Applikation zu erstellen. Es wird „Responsive Design“ implementiert, so dass sich der Inhalt automatisch dem Anzeigegerät anpasst. Auf dem Smartphone und Tablet wird dadurch eine optimierte Version dargestellt.

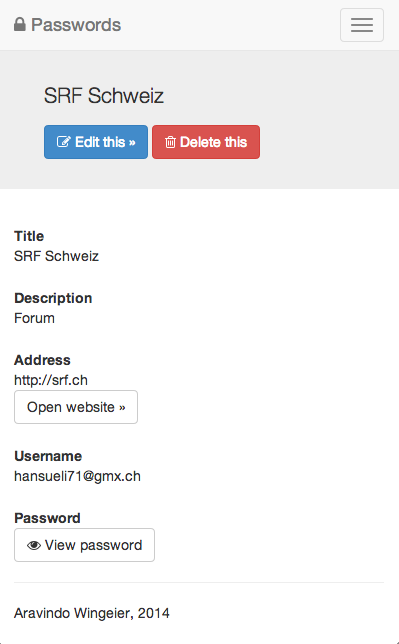


Abbildung : Tablet Ansicht

Abbildung : Mobile Ansicht

## Template-Engine „Twig“

Mit Twig kann der Webdesigner mit einfachem HTML arbeiten und bei Bedarf mit Scripting erweitern. In der Abbildung 4 wird gezeigt, wie einfach mit „if“ ein Wert geprüft, und mit {{var}} ein Wert ausgegeben werden kann.

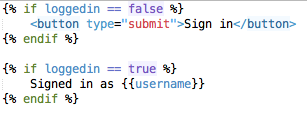


Abbildung : IF/Else in Twig sind einfach zu realisieren

Ausserdem kann Vererbung eine Basisklasse verwendet werden, um dann einzelne Bereiche an die Bedürfnisse der jeweiligen Seite anzupassen. In der X wird das Basistemplate „base.html“ verwendet, aber die Bereiche „jumbotron“ und „content“ überschrieben.

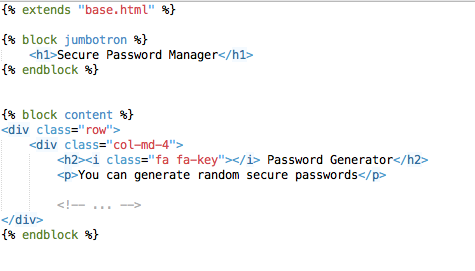


Abbildung : Vererbung und Erweiterung

# Backend

Im Backend kommt PHP mit dem Framework „Slim“ zum Einsatz, welches das Routing unterstützt. So kann ein übersichtliches Mapping zwischen URL und Controllern gemacht werden. Es braucht nur ein paar Zeilen Code:

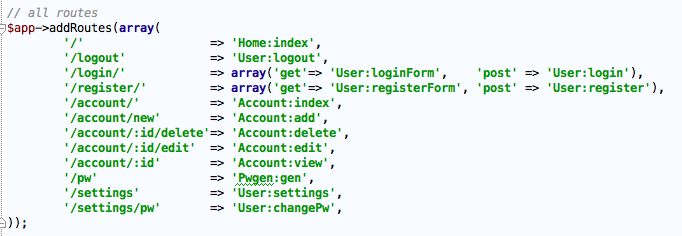


Abbildung : Routing der Applikation zu Controllern

## Controller

Ein Controller ist einfach eine PHP-Klasse, welche SlimController erweitert und über das Routing referenziert wird. In der Abbildung 7 wird der Controller für das generieren von Passwörtern gezeigt. Im Routing wird die Action „Pwgen:gen“ referenziert, somit wird in der Klasse PwGenController die Methode genAction() aufgerufen. Mit render() wird eine Twig-View aufbereitet.



Abbildung : Beispiel Controller: PwgenController

## Salted Hash

Die Anmeldepasswörter werden in der Datenbank nur als BLOWFISH-Hash gespeichert. Ab PHP 5.5.0 werden Methoden für die sichere Erstellung von Salt und Hash angeboten, die im Projekt verwendet werden. Die Methode password\_hash($password) generiert einen zufälligen Salt-Wert und erstellt einen Salted-Hash aus Passwort. Der verwendete Algorithmus das Salt werden als Teil des Hash zurückgegeben. Somit benötigt der Hash nur ein Feld in der Datenbank.

Mit password\_verify($password,$hash) kann ein solcher Hash verifiziert werden.

Beispiel-Hash: $2y$10$X2rpDFykc2RMEI6UX6klhOKKlK1q49pqy8jOA/6QHTlbzvl1puIrS

## Verschlüsselung

Für die Verschlüsslung und Entschlüsselung werden ebenfalls Methoden von PHP verwendet. Die Funktionalität wird im Projekt in der Klasse Crypto abstrahiert, es werden die Methoden encryptInformation($information, $encryptionKey) und decryptInformation($cipherWithIvBase64, $encryptionKey) angeboten.

Beim Verschlüsseln wird ein zufälliger IV erstellt und eine beliebige Information mit einem Key verschlüsselt (AES 128). Der IV wird dem Ciphertext vorangestellt und der ganze Ciphertext wird Base64 kodiert.

Bei der Entschlüsslung wird Base64-Dekodiert, der vorangestellt IV entfernt und der Rest entschlüsselt.

Siehe src/app/Crypto.php für die Implementation. Eine interessante Code-Stelle ist ebenfalls das ändern des Passworts, wo alle Konto-Informationen neu verschlüsselt werden müssen: src/app/Controller/UserController.php, Methode changePwAction().

## Dependency Management

Moderne PHP Applikationen verwenden Composer (<https://getcomposer.org/>) um benötigte Dependencies herunterzuladen und einzubinden. So sind auch in diesem Projekt in der Datei src/composer.json alle Abhängkeiten eingetragen. Mit dem Befehl composer install werden alle Libraries in das Verzeichnis „vendor“ geladen. Composer bietet ebenfalls einen Autoloader an, so dass die Klassen direkt geladen werden, ohne jedes mal require() aufzurufen. Siehe doc/INSTALL.md für Details zur Installation.

# Übersicht der Frameworks und Libraries

Wie erwähnt werden diverse Frameworks und Libraries verwendet. Dies hat auch den Vorteil, dass diese Funktionen bereits von der Community gut getestet wurden.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Quelle | Techn. | Beschreibung | Lizenz |
| Slim | <http://www.slimframework.com/> | PHP | MVC/Router | MIT |
| twig | <http://twig.sensiolabs.org/> | PHP | HTML-Templates | new BSD |
| Aura.Sql | <https://github.com/auraphp/Aura.Sql> | PHP | SQL-Zugriff | copyleft |
| valitron | <https://github.com/vlucas/valitron> | PHP | Validierung | new BSD |
| pwgen-php | <https://github.com/roderik/pwgen-php> | PHP | Passwort-Generator | GPL |
| monolog | <https://github.com/Seldaek/monolog> | PHP | Logging | copyleft |
| composer | <https://getcomposer.org/> | PHP | Class autoloading | MIT |
| bootstrap | <http://getbootstrap.com/> | HTML/CSS/JS | Frontend framework | MIT |
| jQuery | <https://jquery.com/> | JS | JavaScript DOM library | MIT |
| Font Awesome | <http://fontawesome.io/> | CSS | Icons | OFL/MIT |

Die GPL2-Lizenz dieses Projektes ist kompatibel mit den verwendeten Komponenten.

# Anhang: Source Code

Der Sourcecode des gesamten Projekts kann abgerufen werden unter: <https://github.com/synox/Password-Manager> (GPL2 Lizenz)