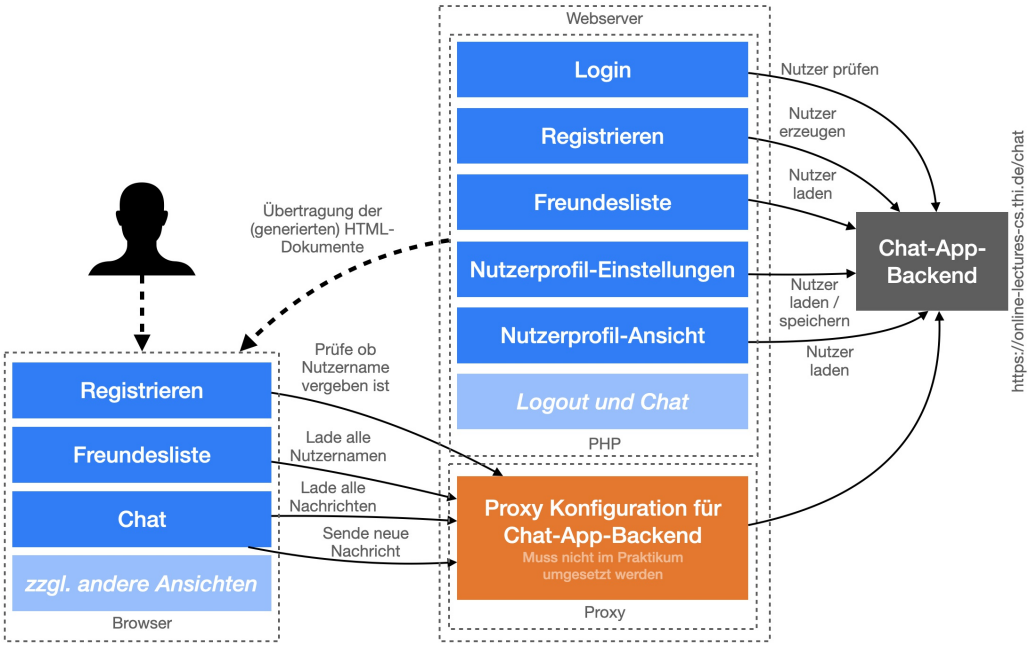
Aufgabe 5: Chat-App PHP

Anmerkung für INF-Bachelor:
Für Aufgabe 5 wird mit dem **Ergebnis der Aufgabe 3** fortgesetzt, d.h. die in Aufgabe 4 entwickelte Angular-App wird quasi ignoriert. Nach Fertigstellung der in diesem Aufgabenblatt geforderten PHP-Anwendung lassen sich dadurch sehr gut die Unterschiede zwischen *Single Page Applications* und Serverseitiger HTML-Generierung (z.B. PHP) erkennen!

In der Aufgabe 5 liegt der Schwerpunkt auf der Realisierung der serverseitigen PHP-Anwendung. Mit dieser Aufgabe werden die noch fehlenden funktionalen Bausteine ergänzt und mit der Finalisierung der Aufgabe steht eine voll funktionsfähige Chat-App zur Verfügung.



Anmerkung: Obige Software-Architektur könnte aus dem Blickwinkel der IT-Sicherheit heraus problematisch sein, da sowohl der Webserver als auch der Backend-Server für den Browser (und damit für sehr viele Rechner!) erreichbar sein müssen. Aus Gründen der Einfachheit wird diese Software-Architektur für das Praktikum beibehalten. Möglichkeiten zur Verbesserung wären

1. Weiterleitung der Backend-Aufrufe des Browsers über den Webserver: Der Browser ruft den Webserver (und nicht den Backend-Server) auf, z.B: mit einem Pfad-Präfix 'api'. Der Webserver erkennt das Pfadpräfix und leitet die Aufrufe an das Backend weiter.
2. Durchführung der Backend-Aufrufe durch den Webserver: Der Browser ruft ein PHP-Script (z.B. *backend.php*) auf anstatt direkt das Backend aufzurufen. Das PHP-Script führt die Aufrufe durch und liefert die Ergebnisse an den Browser zurück.

Beide Möglichkeiten führen dazu, dass nur der Webserver von "aussen" nutzbar sein muss, das Backend kann dagegen über entsprechende Firewall-Einstellungen besser geschützt werden, da es nur durch den Webserver und ggf. andere Server genutzt wird.

Vorbereitungen

Für diese Aufgabe wird ein Webserver benötigt mit konfigurierter PHP-Ausführungsumgebung. Sie können hier auf XAMPP zurückgreifen. XAMPP liefert (u.a.) Apache als Webserver und die Ausführungsumgebung PHP sowie das Datenbankmangementsystem MariaDB. Laden, installieren und starten Sie XAMPP.

Öffnen Sie anschließend den `htdocs`-Ordner (Sie finden hierzu einen Button in der GUI von XAMPP) und erstellen Sie einen neuen Ordner für Ihr Projekt. Platzieren Sie anschließend Ihr bisheriges Ergebnis aus dem Praktikum in diesem Ordner.

Im Folgenden sind alle Bestandteile der Anwendung unter Verwendung von PHP im Server vorzuverarbeiten. Hierfür ist die Umbenennung aller HTML-Dateien und in eine Datei mit einer `.php`-Endung erforderlich. Damit werden die Verlinkungen in Ihrem Dokument nicht mehr funktionieren, auch diese sind im Verlauf der Aufgabe zu korrigieren.

Anschließend sind die Arbeiten in dieser Aufgabe zu verteilen, vorgegeben ist hier:

Gemeinsam: Default-Routine, User-Klasse und BackendService-Klasse

1. Team-Mitglied: Login, Logout und Freundesliste
2. Team-Mitglied: Registrieren und Chat-Ansicht
3. Team-Mitglied: Nutzerprofil-Einstellungen und -Ansicht

Für Teams mit zwei Mitgliedern entfällt die Nutzerprofil-Einstellungen und -Ansicht.

Experimente

Im Folgenden sind empfehlenswerte Experimente, welche Sie ausprobieren können, um ein Verständnis über einzelne Mechanismen zu erhalten, die in dieser Aufgabe Verwendung finden.

Query-Parameter

Erstellen Sie eine neue PHP-Datei. Erzeugen Sie eine Grundstruktur und fügen Sie einen PHP-Block in das Body-Element:

```
<!-- ... -->
<body>
    <?php
        // ...
    ?>
</body>
<!-- ... -->
```

Mit diesem Experiment soll der Query-Teil der URL verarbeitet werden. Angenommen Ihr Beispiel ist in dem htdocs Ordner in einem Unterordner `praktikum` und die PHP-Datei nennt sich `test.php`, dann können Sie als Test im Browser `http://localhost/praktikum/test.php?test=1234` aufrufen. Die URL lässt sich dabei wie folgt zerlegen:

- Schema: HTTP
- Domain: localhost
- Port (default): 80
- Pfad: /praktikum/test.php
- Query: test=1234

In PHP können Sie die Informationen des Query über die magische Variable $_GET (oder $_REQUEST) abfragen:

```
    echo $_GET["test"];
```

Das Ergebnis sollte 1234 sein, wenn das Fenster geöffnet wird. Ändern Sie die URL im Browser auf `http://localhost/praktikum/test.php` und beobachten Sie was passiert. Je nach Konfiguration der PHP-Ausführungsumgebung kommt es nun zu einem Fehler, einem Hinweis oder einer leeren Ausgabe.

Sie müssen aus der Perspektive des Servers immer davon ausgehen, dass Anfrageparameter nicht so ankommen, wie Sie sich das vorstellen. Sie müssen also ggf. prüfen, ob diese auch existieren und z.B. nicht leer sind. Hierfür können bekannte Methoden genutzt werden, insbesondere helfen hier häufig `isset` und `empty`. Verändern Sie das Beispiel wie folgt:

```
if(isset($_GET["test"])) {
    if(!empty($_GET["test"])) {
        echo "Wert: " . $_GET["test"];
    } else {
        echo "Kein Wert!";
    }
} else {
    echo "Kein Parameter übergeben";
}
```

Betrachten Sie anschließend die folgenden URL-Ergebnisse:

- http://localhost/praktikum/test.php?test=1234

- http://localhost/praktikum/test.php?test=
- http://localhost/praktikum/test.php

(Optional) Sie können mit mehreren Parametern experimentieren. Mehrere Parameter in einem Query-String werden durch das &-Zeichen voneinander getrennt. Versuchen Sie z.B: http://localhost/praktikum/test.php?test=1234&foo=abc.
Es sollten entsprechend der oben beschriebenen Vorgehensweise zwei Parameter geprüft und ggf. ausgegeben werden.

Generieren von HTML in Kontrollflüssen

PHP wird im Server verarbeitet und erzeugt ein Ergebnis, welches anschließend an den Browser gesendet wird, welcher dieses interpretiert und darstellt. Wenn PHP ein HTML-Dokument erzeugt, kann der Browser dies entsprechend visualisieren. Für die folgenden Aufgaben müssen Sie Inhalte über verschiedene Kontrollflüsse generieren, um Ausgaben entsprechend der verfügbaren Informationen zu erzeugen.

Erstellen Sie eine neue PHP-Datei. Erzeugen Sie eine Grundstruktur fügen Sie PHP-Blöcke am Anfang des Dokumentes und in das Body-Element ein:

```
<?php
// ...
?>
<!-- ... -->
<body>
    <?php
        // ...
    ?>
</body>
<!-- ... -->
```

Für das Beispiel verwenden wir ein konstantes Array, welches drei Werte enthält, ergänzen Sie hierfür im PHP-Block am Anfang folgenden Code:

```
$list=array(1, 2, 3, 4, 5);
```

Auf der Ebene des PHP-Ausführung existiert nun die Variable list und kann in dem selben, aber auch in nachfolgenden Blöcken genutzt werden. Geben Sie den Inhalt im PHP-Block im Body-Element aus:

```
var_dump($list);
```

Die Methode `var_dump` kann genutzt werden, um sich den Inhalt einer Variable anzusehen, egal welcher Typ verwendet wird. Dies kann bei Arrays und Objekten hilfreich sein. Die Ausgabe ist jedoch kein sinnvolles HTML. Wenn Sie HTML erzeugen wollen, müssen Sie die Informationen geeignet verarbeiten und ein HTML-Dokument erzeugen. Iterieren Sie über die Liste und geben Sie Paragraphen aus:

```
foreach($list as $value) {
    echo "<p>" . $value . "</p>";
}
```

Damit platzieren Sie für jedes Element in der Liste eine entsprechende Ausgabe in dem Ergebnisdokument. Wird die Seite im Browser geladen, können Sie sich den Quellcode anzeigen lassen und sollten 1 2 3 4 5 vorfinden.

Bei einfachen Ausgaben ist dieses vorgehen machbar, es kann jedoch passieren, dass Sie komplexe HTML-Konstrukte erzeugen wollen. Damit diese Situation vereinfacht wird, können Sie den PHP-Block unterbrechen und anschließend direkt mit HTML-Quellcode arbeiten. Passiert die Unterbrechung in einer Schleife, wird der Teil mehrfach als Ausgabe erzeugt. Wird dies mit einer Bedingung kombiniert, wird die Ausgabe nur bei Eintreten der Bedingung ausgeben. Verändern Sie den PHP-Block wie folgt:

```
<!-- ... -->
<body>
    <?php
        foreach($list as $value) {
    ?>
    <p><?php echo $value; ?></p>
    <?php
        }
    ?>
</body>
<!-- ... -->
```

Sie können dieses Beispiel weiter vereinfachen, indem Sie die alternative PHP-Block-Form <?= ... ?> nutzen, welches für Ausgaben gedacht ist:

```
<!-- ... -->
<body>
    <?php foreach($list as $value) { ?>
    <p><?= $value; ?></p>
    <?php } ?>
</body>
<!-- ... -->
```

Sie können dieses Vorgehen auf verschiedene Art und Weise in PHP nutzen. Insbesondere sollten Sie dies mit typischen Schleifen (while, for, foreach) und Bedingungen (if, else, else if) ausprobieren.

Weiterleitung

Wie kann man einen Aufruf auf eine andere Webseite umlenken? PHP wird auf dem Server ausgeführt und Sie können den Nutzer nicht ohne weiteres auf eine andere Seite "schicken", solange sich die Ausführung auf dem Server befindet. Damit eine Weiterleitung funktioniert, muss *etwas* an den Browser gesendet werden, was anschließend den Browser veranlasst die Weiterleitung vorzunehmen. Neben diversen Möglichkeiten über HTML und JS ist hier insbesondere die Möglichkeit über einen HTTP-Header-Parameter herauszuheben. Sie können in dem HTTP-Response eine Information hinterlassen, die der Browser nutzt und damit automatisch auf eine andere Seite weiterleitet, egal was für ein HTML-Ergebnis erzeugt wurde.

Erstellen Sie eine neue PHP-Datei. Erzeugen Sie eine Grundstruktur fügen Sie einen PHP-Block am Anfang des Dokumentes ein:

```
<?php
header("Location: login.php");
?>
<!-- ... -->
<body>
    <p>Hallo, Welt!</p>
</body>
<!-- ... -->
```

Der Aufruf der Methode `header` ergänzt den Parameter im HTTP-Response. Probieren Sie dies aus und beobachten Sie was passiert.

Spannend ist hier, dass je nach Konfiguration des Webservers und insbesondere PHP es zu Fehlern kommen kann, wenn bereits Ausgaben erzeugt wurden. Es ist daher zu empfehlen, Methoden-Aufrufe von `header` immer in einen PHP-Block am Anfang des Dokumentes zu platzieren:

```
<?php
header("Location: login.php"); // gut!
$foo = 12435;
header("Location: login.php"); // gut!
echo "Hallo"; // erzeugt ausgabe
header("Location: login.php"); // schlecht!
?>
<!-- ... -->
<body>
    <p>Hallo, Welt!</p>
    <?php
        header("Location: login.php"); // noch schlechter!
    ?>
</body>
<!-- ... -->
```

Es empfiehlt sich, dass Sie die Ausführung des restlichen PHP-Codes abbrechen, wenn Sie durch eine Bedingung die Entscheidung getroffen haben, dass eine Weiterleitung notwendig ist. Hierfür kann die Methode `exit` genutzt werden. Zum Beispiel:

```
<?php
$foo = 12435;
if($foo == 12345) {
    header("Location: login.php");
    exit();
}
?>
<!-- ... -->
<body>
    <p>Hallo, Welt!</p>
</body>
<!-- ... -->
```

Die Ausführung (und damit auch alle nachfolgenden Ausgaben) wird mit dem Aufruf von `exit()` beendet. Alles was vor exit an Ausgaben erzeugt wurde (inkl. den Header-Informationen) wird zum Browser gesendet.

Formulardatenverarbeitung

In den folgenden Aufgaben müssen immer wieder Formulare verarbeitet werden. Nutzen Sie hierfür den Skript und probieren Sie für den Einstieg folgendes Experiment.

Erstellen Sie eine neue PHP-Datei. Erzeugen Sie eine Grundstruktur fügen Sie ein einfaches HTML-Formular in den Body:

```
<!-- ... -->
<body>
    <form method="get" action="test.php">
        <input name="test">
        <button type="submit">Absenden</button>
    </form>
</body>
<!-- ... -->
```

Öffnen Sie das Dokument und drücken Sie auf Absenden und betrachten Sie, wie der Browser die URL der angezeigten Seite verändert hat. Durch die Verwendung von `get` wurden beim Absenden des Formulars Query-Parameter erzeugt.

Ergänzen Sie nun im Button ein Attribut `name` und ein Attribut `value`:

```
<button type="submit" name="action" value="foo1">Absenden</button>
```

Laden Sie die Seite neu, drücken Sie auf Absenden und betrachten Sie erneut die URL. Hat ein Button einen Namen, wird der Wert unter den Namen als Query-Parameter mitgesendet. Auf diese Art und Weise kann man herausfinden, welcher Button genutzt wurde, um das Formular abzusenden. Hier können sogar mehrere Formularfelder zum Einsatz kommen:

```
<button type="submit" name="action" value="foo1">Absenden1</button>
<button type="submit" name="action" value="foo2">Absenden2</button>
<button type="submit" name="bar" value="foo3">Absenden3</button>
```

Laden Sie die Seite neu und probieren Sie die Buttons aus. Beobachten Sie die URL und welche Query-Parameter verwendet werden.

Abschließend können Sie die Methode des Formulars auf POST verändern:

```
<form method="post" action="test.php">
```

Laden Sie die Seite neu und senden Sie das Formular ab. Die URL verändert sich nun nicht mehr. Die Daten des Formulars werden als Payload in der Anfrage übermittelt. Zur Inspektion können Sie den Webinspektor des Browser, einen PHP-Debugger oder in einen PHP-Block `var_dump($_POST);` platzieren, um zu sehen, was hier übermittelt wurde:

```
<!-- ... -->
<body>
    <form method="post" action="test.php">
        <input name="test">
        <button type="submit" name="action" value="foo1">Absenden1</button>
        <button type="submit" name="action" value="foo2">Absenden2</button>
        <button type="submit" name="bar" value="foo3">Absenden3</button>
    </form>
    <?php var_dump($_POST); ?>
</body>
<!-- ... -->
```

Laden Sie auch diese Seite neu und probieren Sie die Buttons aus. Sie sehen, wie sich die Informationen die bei der PHP-Ausführung zur verfügung stehen, verändern. Unter Berücksichtigung des Experiments zu Query-Parametern und HTML-Generierung könnten Sie nun die Informationen der Anfrage entsprechend verarbeiten.

Abschließend können Sie in das Formularfeld einen Wert eintippen und dieses Absenden. Mit dem neuladen der Seite ist die Eingabe verschwunden (die Seite wurde neu initialisiert). Aber während die PHP-Ausführung stattfand, standen die Informationen über `$_POST` zur Verfügung. Setzen Sie den Cursor in die Adresszeile des Browser und drücken Sie Enter (nicht die Seite neuladen), dann sehen Sie wie das Feld leer bleibt (Seite wird erneut initialisiert) und die Formularinformationen sind nicht vorhanden - da diese nicht mit der Anfrage versendet wurden.

Für Formulare bietet sich neben der Verarbeitung der Informationen das Ausfüllen der Formularfelder basierend auf bereits eingegebenen Informationen an, insbesondere wenn Fehler vorliegen. Verändern Sie das Input-Feld:

```
<input name="test" value="<?php if(isset($_POST['test'])) { echo
$_POST['test']; } ?>">
```

Laden Sie die Seite neu, füllen Sie das Eingabefeld aus und senden Sie das Formular ab. Die Eingabe sollte nun im Formularfeld erhalten bleiben. Diese Variante wirkt unübersichtlich und es ist zu empfehlen, die Verarbeitung der Formulardaten an einer geeigneten Stelle durchzuführen und mit Variablen zu arbeiten. Als Beispiel könnte man dies wie folgt verändern:

```
<?php
$testWert = "";
if(isset($_POST['test'])) {
    $testWert = $_POST['test'];
}
?>
<!-- ... -->
<body>
    <form method="post" action="test.php">
        <input name="test" value="<?= $testWert; ?>">
        <button type="submit" name="action" value="foo1">Absenden1</button>
        <button type="submit" name="action" value="foo2">Absenden2</button>
        <button type="submit" name="bar" value="foo3">Absenden3</button>
    </form>
    <?php var_dump($_POST); ?>
</body>
<!-- ... -->
```

Teilaufgabe a: Default-Routine

Erzeugen Sie eine PHP-Datei `start.php`. Diese soll von allen anderen PHP-Dateien importiert werden und dient der Auslagerung von Initialisierungsaufgaben als auch Konfigurationen.

Eine notwendige Konfiguration ist die Definition, wie PHP mit unbekannten Klassennamen umgehen soll. Details dazu finden Sie im Skript. Fügen Sie folgende Anweisung in die Datei `start.php`:

```
spl_autoload_register(function($class) {
    include str_replace('\\', '/', $class) . '.php';
});
```

Die weitere notwendige Initialisierungsaufgabe dient dem Starten der Sitzung. In PHP kann dies mithilfe des Aufrufs `start_session();` realisiert werden. Fügen Sie ebenso diesen Aufruf in die `start.php`-Datei ein.

Abschließend können Konstanten in der `start.php` definiert werden. Hierfür bietet PHP die Funktion `define(name, wert)` an. Ergänzen Sie zwei Konstanten, um die Informationen für den Zugang zum Chat-Backend anzugeben:

```
define('CHAT_SERVER_URL', 'https://online-lectures-cs.thi.de/chat');
define('CHAT_SERVER_ID', '...'); # Ihre Collection ID
```

Sie können die `start.php` in allen Ansichten in der ersten Zeile über den Aufruf der Funktion `require` laden:

```
require("start.php");
```

Teilaufgabe b: User-Klasse und Friend-Klasse

Erstellen Sie einen neuen Ordner mit dem Namen `Model` und erzeugen Sie in diesen Ordner eine Datei mit dem Namen `User.php`. In dieser Datei soll eine Klasse User beschrieben werden, die alle notwendigen Informationen unseres Nutzers beschreibt. Damit die Klasse später korrekt geladen wird, muss diese zusätzlich einen Namespace zugeordnet werden. Starten Sie daher mit folgenden Aufbau, bevor Sie Attribute und Methoden ergänzen:

```
<?php
namespace Model;
class User {
}
?>
```

Ergänzen Sie anschließend die folgenden privaten Attribute: `username`. Fügen Sie für das Attribute einen Getter hinzu. Ergänzen Sie desweiteren einen Konstruktor. Dieser Konstruktor soll einen optionalen Parameter `username` erwarten, welcher auf das entsprechende Attribut zugewiesen wird. Der Default-Wert ist `null`.

Damit die Klasse anschließend für die Repräsentation eines Nutzer verwendet werden kann, der vom Backend geladen wurde oder zum Backend gesendet wird, muss beschrieben werden, wie die JSON-basierten Daten auf diese Klasse angewendet werden sollen. In PHP gibt es hierzu eine Hilfestellung über die Schnittstelle `JsonSerializable`. Wird diese implementiert, muss eine Methode implementiert werden, die beschreibt, wie welche Attribute in das JSON-Format übernommen werden sollen. Modifieren Sie die Klasse so, dass die Schnittstelle implementiert wird. Sie können hierfür folgende Basis nutzen:

```php
namespace Model;
use JsonSerializable;
class User implements JsonSerializable {
    // ...
    public function jsonSerialize() {
        return get_object_vars($this);
    }
}
```

Probieren Sie den aktuellen Stand in einer neuen PHP-Datei:

```php
require("start.php");
$user = new Model\User("Test");
$json = json_encode($user);
echo $json;
```

Sie sollten validen JSON-Code im Browser sehen, wenn Sie die Seite öffnen.

Abschließend ist die Verarbeitung von JSON-Code zur Erzeugung einer neuen User-Instanz notwendig. Ergänzen Sie hierfür in der User-Klasse eine statische Methode `fromJson`. Diese erwartet einen Parameter `data` der ein bereits verarbeitetes JSON-Paket als PHP-Standard-Objekt repräsentiert. Dieser enthält also alle Informationen wie `username`und `friends`, jedoch ist dies keine Instanz der Klasse User. Ihre Aufgabe ist es in dieser Methode eine neue Instanz der Klasse User zu erzeugen und anschließend alle Attribute aus dem Parameter in die neu erzeugte Instanz zu übernehmen. Sie können als Ausgangspunkt folgende Iteration verwenden:

```php
foreach ($data as $key => $value) {
    $user->{$key} = $value;
}
```

Probieren Sie den aktuellen Stand erneut aus:

```php
require("start.php");
$user = new Model\User("Test");
$json = json_encode($user);
echo $json . "<br>";
$jsonObject = json_decode($json);
$newUser = Model\User::fromJson($jsonObject);
var_dump($newUser);
```

Ergänzen Sie abschließend die Friend-Klasse nach dem selben Prinzip. Die Klasse soll die zwei Attribute `username` und `status` enthalten. Für beide Attribute sind Getter notwendig. Zusätzlich ist ein Konstruktor notwendig, welcher als optionalen Parameter den `username`empfängt, der Default-Wert ist `null`, und diesen auf das entsprechende Attribut zuweist. Stellen Sie sicher, dass auch hier die JSON-Serialisierung und -Deserialisierung möglich ist. Zusätzlich bieten sich zwei Methoden an, die den Zustand auf `accepted` bzw. `dismissed` setzen.

Teilaufgabe c: Service Anbindung

Wie auch die JavaScript-Bestandteile der Anwendung soll auch in der PHP-Implementierung auf das Backend zugegriffen werden. Auch hier ist der Zugriff über HTTP notwendig und auch hier wird mittels JSON zur Datenrepräsentation gearbeitet. Mit der Teilaufgabe c soll eine Klasse umgesetzt werden, welche die Kommunikation mit dem Backend vereinfacht.

Erstellen Sie einen neuen Ordner `Utils`. Fügen Sie die bereitgestellte Datei `HttpClient.php` in diesen Ordner ein. Fügen Sie zusätzlich eine Datei `BackendService.php`in diesen Ordner und definieren Sie die Klasse `BackendService`:

```
<?php
namespace Utils;
class BackendService {
}
```

Ergänzen Sie in der Klasse die privaten Attribute `base` und `id` sowie einen Konstruktor zwei Parameter erwartet, um die beiden Attribute mit einem Wert zu belegen.

Die folgende Aufgabe in der Klasse `BackendService` ist es, alle Funktionen, die am Backend genutzt werden, als Methoden in der Klasse vorzusehen. Als Basis können Sie hierfür die `HttpClient`-Klasse nutzen, welche die HTTP-Aufrufe für Sie abbildet. Diese Klasse bietet eine statische Methode `get(url, token)` für HTTP-GET und eine `post(url, data, token)` für HTTP-POST. Der `token`-Parameter ist optional. Ist der HTTP-Status der Antwort 200 wird als Rückgabe das JSON-Objekt (bereits deserialisiert) zurückgegeben. Ist der HTTP-Status 204 wird true zurückgegeben. Andernfalls werfen die Methoden eine Exception.

Starten Sie mit einem einfachen Experiment und ergänzen Sie eine Methode `test`, um sich von der Funktionsweise zu überzeugen:

```
public function test() {
    try {
        return HttpClient::get($this->base . '/test.json');
    } catch(\Exception $e) {
        error_log($e);
    }
    return false;
}
```

Erzeugen Sie eine neue PHP-Datei für einen Test mit folgendem Inhalt:

```
require("start.php");
$service = new Utils\BackendService(CHAT_SERVER_URL, CHAT_SERVER_ID);
var_dump($service->test());
```

Sie erhalten ein entpacktes Objekt vom Server und könnten dies nun im folgenden weiter verarbeiten. In diesem Fall ist es ein Array mit Nachrichtenobjekten. Optional können Sie versuchen über dieses zu iterieren und die Attribute der Nachrichten auszugeben.

Implementieren Sie alle notwendigen Methoden für den Zugriff auf das Backend, orientieren Sie sich an der Test-Methode. Jede Methode sollte eine Ausnahmebehandlung erhalten und im Fall einer Ausnahme `false` zurückgeben:

- Login `login(username, password)`: Führen Sie den HTTP-POST-Aufruf ohne Token aus und weisen Sie das Ergebnis einer lokalen Variable zu. Speichern Sie das Attribut `token` des Ergebnisses in eine Session-Variable, z.B. `chat_token` und geben Sie `true` zurück. Sollte der Login nicht geklappt haben liefert der Aufruf einen HTTP-Status-Code 403, es wird demnach durch die Ausnahme automatisch `false` zurückgegeben (vgl. test-Methode). Übergeben Sie als Daten ein dynamisch erzeugtes Array der Form: `array('username' => $username, 'password' => $password)`.
- Register `register(username, password)`: Realisieren Sie eine Implementierung analog zu der `login`-Methode
- Nutzer Laden `loadUser($username)`: Führen Sie einen HTTP-GET-Aufruf mit Token aus, weisen Sie das Ergebnis einer lokalen Variable zu und nutzen Sie die `fromJson`-Methode Ihrer `User`-Klasse. Geben Sie das Ergebnis zurück.
- Nutzen Speichern `saveUser(user)`: Führen Sie einen HTTP-POST-Aufruf mit Token aus. Übergeben Sie die Instanz des Users im Parameter `user` im Aufruf. Geben Sie das Ergebnis direkt zurück.
- Freunde laden `loadFriends()`: Führen Sie einen HTTP-GET-Aufruf mit Token aus, weisen Sie das Ergebnis einer lokalen Variable zu und nutzen Sie die `fromJson`-Methode Ihrer `Friend`-Klasse um Instanzen zu erzeugen. Bedenken Sie, das hier eine Liste von Freunden vom Backend geliefert wird. Geben Sie das Ergebnis zurück.
- Freundesanfrage `friendRequest($friend)`: Führen Sie einen HTTP-POST-Aufruf mit Token aus. Geben Sie das Ergebnis direkt zurück. Der Parameter `friend` soll eine Instanz der Klasse `Friend` sein.
- Anfrage annehmen `friendAccept($friend)`: Führen Sie einen HTTP-PUT-Aufruf mit Token aus. Geben Sie das Ergebnis direkt zurück. Der Parameter `friend` soll eine Instanz der Klasse `Friend` sein.
- Anfrage ablehen `friendDismiss($friend)`: Führen Sie einen HTTP-PUT-Aufruf mit Token aus. Geben Sie das Ergebnis direkt zurück. Der Parameter `friend` soll eine Instanz der Klasse `Friend` sein.
- Freund entfernen `friendRemove($friend)`: Führen Sie einen HTTP-DELETE-Aufruf mit Token aus. Geben Sie das Ergebnis direkt zurück. Der Parameter `friend` soll eine Instanz der Klasse `Friend` sein.
- Nutzername existiert `userExists($username)`: Führen Sie einen HTTP-GET-Aufruf mit Token aus. Geben Sie das Ergebnis direkt zurück.
- Ungelesene Nachrichten `getUnread()`: Führen Sie einen HTTP-GET-Aufruf mit Token aus. Geben Sie das Ergebnis direkt zurück.

Hinweis: Sie finden weitere Informationen zu den möglichen Anfragen über die API Dokumentation (http://localhost:3000/full). Sie können die API-Dokumentation für Ihre Collection konkretisieren, indem Sie die ID mitgeben: http://localhost:3000/full/91f60642-f8d4-42ec-a26d-84c7eb95dbc3.

Prüfen Sie anschließend alle Methoden der Implementierung Ihrer `BackendService`-Klasse, z.B.:

```
require("start.php");
$service = new Utils\BackendService(CHAT_SERVER_URL, CHAT_SERVER_ID);
var_dump($service->login("Test123", "12345678"));
var_dump($service->loadUser("Test123"));
// ... usw.
```

Es empfiehlt sich bis hier hin im Team gemeinsam vorzugehen und sicherzustellen, dass alle Methoden sicher funktionieren. Fahren Sie mit den folgenden Aufgaben erst anschließend fort.

Sie können abschließend in der `start.php` die Initialisierung der Service-Klasse vorsehen, da diese im folgenden auf allen Seiten benötigt werden. Ergänzen Sie hierfür den Aufruf `$service = new Utils\BackendService(CHAT_SERVER_URL, CHAT_SERVER_ID);` am Ende der Datei.

Teilaufgabe d: Login

Die Login-Funktion ist umzusetzen. Hierfür ist die Verarbeitung der Formulareingaben notwendig. Modifizieren Sie das Formular so, dass die Daten per POST an `login.php` versendet werden.

Ergänzen Sie am Anfang der `login.php`-Datei einen PHP-Block, welcher die Datei `start.php` lädt.

Verarbeiten Sie anschließend die Formularfelder um den Nutzernamen und das Passwort zu erhalten. Führen Sie die Verarbeitung nur im dem Fall aus, in dem das Formular auch abgesendet wurde. Rufen Sie anschließend die Methode `login` im BackendService auf. Ist das Ergebnis `true` soll der Nutzername in der Session-Variable `user` gespeichert werden und der Nutzer mit dem Aufruf `header("Location: friends.php");` auf die Freundesliste weitergeleitet werden. Andernfalls ist eine Fehlermeldung sichtbar zu platzieren. Bootstrap bietet hier eine Komponente für Meldungen (https://getbootstrap.com/docs/5.1/components/alerts/).

Ergänzen Sie abschließend direkt nach dem Aufruf `require("start.php");` eine Prüfung, ob die Session-Variable `user` bereits existiert. Ist dem so, kann davon ausgegangen werden, dass der Nutzer bereits angemeldet ist und direkt mittels `header("Location: friends.php");` weitergeleitet werden.

Teilaufgabe e: Registrieren

Realisieren Sie die Funktion zum Registrieren. Hierfür ist die Verarbeitung der Formulareingaben notwendig. Modifizieren Sie das Formular so, dass die Daten per POST an `register.php` versendet werden.

Ergänzen Sie am Anfang der `register.php`-Datei einen PHP-Block, welcher die Datei `start.php` lädt.

Verarbeiten Sie anschließend die Formularfelder um den Nutzernamen, das Passwort und die Wiederholung zu erhalten. Sie müssen in diesem Formular, bevor Sie den Nutzernamen und das Passwort an die entsprechende Service-Methode weitergeben sicherstellen, dass der Nutzer alles korrekt eingegeben hat. Die Prüfung in JavaScript reicht nicht, da diese auf der Seite des Clients umgangen werden könnte. Prüfen Sie daher, dass:

- der Nutzername nicht leer ist und min. 3 Zeichen hat
- der Nutzername nicht vergeben ist (siehe `BackendService`)
- das Passwort nicht leer ist
- das Passwort min. 8 Zeichen hat
- das Passwort mit der Wiederholung übereinstimmt

Nur wenn alle Bedingungen stimmen, soll die Methode `register` aufgerufen werden. Andernfalls ist eine Fehlermeldung zu platzieren. Ist das Ergebnis der Methode `register` `true` soll der Nutzername in der Session-Variable `user` gespeichert werden und der Nutzer mit dem Aufruf `header("Location: friends.php");` auf die Freundesliste weitergeleitet werden. Ist das Ergebnis `false` ist ebenfalls ein Fehler auszugeben.

Abschließend müssen Sie über Ihre PHP-Implementierung den Server und die Collection ID an den Client kommunizieren. Mit Bezug zu Aufgabe 3 wurde hierfür an einer geeigneten Stelle Konstanten im Dokument definiert, die Sie später in Ihrer JavaScript-Implementierung genutzt haben. Dadurch, dass Sie PHP nutzen, um eine Vorverarbeitung zur Erzeugung eines Ergebnisdokumentes zu realisieren, können Sie an dieser Stelle ebenso JavaScript-Fragmente produzieren. In diesem Fall nutzen wir dies, um Informationen an die Client-Anwendung zu übergeben. Sie können etwas der folgenden Form nutzen, passen Sie dies an Ihre aktuelle Lösung entsprechend an:

```
<script>
    window.chatCollectionId = "<?= CHAT_SERVER_ID ?>";
    window.chatServer = "<?= CHAT_SERVER_URL ?>";
</script>
```

Teilaufgabe f: Logout

Realisieren Sie eine Bereinigung der Sitzungsinformationen. Ergänzen Sie einen PHP-Block am Anfang der Datei, welcher die Datei `start.php` lädt und anschließend die Methode `session_unset` aufruft.

Teilaufgabe g: Freundesliste

Die Freundesliste ist der zentrale Anlaufpunkt, mit dem alle aktuellen Freunde und Anfragen aufgelistet werden und über verschiedene Aktionen die Freundesliste modifiziert wird. Zusätzlich ist die Ansicht unter Verwendung von PHP zu erzeugen. Ergänzen Sie einen PHP-Block am Anfang der Datei, welcher die Datei `start.php` lädt und prüfen Sie ob die Session-Variable `user` gesetzt und nicht leer ist. Andernfalls leiten Sie die Anfrage direkt auf die Login-Seite mittels `header("Location: login.php");` weiter.

Im ersten Schritt ist die Darstellung zu generieren. Hierfür muss die eigene Freundesliste geladen werden und über das resultierende Ergebnis für Iterationen genutzt werden. Rufen Sie an der Instanz zum BackendService die Methode `loadFriends` auf und beziehen Sie Freunde.

Anschließend muss die Liste an Freunden in HTML überarbeitet werden. Bisher waren statische Informationen eingetragen, jetzt soll hier unter Verwendung des Abfrageergebnisses die Liste erzeugt werden. Achten Sie darauf, dass Sie PHP-Blöcke unterbrechen können. Dies funktioniert ebenso innerhalb von Schleifen, auf diese Art können Sie HTML-Blöcke über PHP-Schleifen mehrmals in die Ausgabe platzieren. Beispiel dazu:

```
<?php
// Daten beziehen...
$liste = array(1, 2, 3);
?>
<ul>
    <?php foreach ($liste as $value) { ?>
    <li><?= $value ?>
    <?php } ?>
</ul>
```

Generieren Sie die Liste entsprechend des Arrays an Freunden. Achten Sie darauf, nur Freunde mit dem Status `accepted` auszugeben. Verlinken Sie jeden Listeneintrag so, dass über einen Query-Parameter der Name des Freundes an die Chat-Ansicht übergeben wird.

Für den Fall das keine Freunde existieren, soll ein entsprechender Hinweis ausgegeben werden.

Für die Liste an Freundschaftsanfragen ist analog zu der Freundesliste vorzugehen. Der Status ist hier `requested`. Die Freundesliste wird mit PHP so verarbeitet, dass eine HTML-Liste im gewünschten Layout entsteht. Sind keine Anfragen vorhanden, soll keine Liste entstehen und auch kein Hinweis angezeigt werden. Sie haben hier ggf. ein `onclick`-Ereignis eingearbeitet, hier muss der korrekte Nutzer mittels PHP platziert werden.

Als nächstes ist das Formular zum Hinzufügen von Freundschaftsanfragen und dessen Funktionalität in PHP zu realisieren. Stellen Sie sicher, dass der Button mit dem Typ `submit` einen Namen (z.B. `action`) hat und einen passenden Wert (z.B. `add-friend`). Ändern Sie die HTTP-Methode zur Übertragung der Formulardaten auf P0ST und setzen Sie als Aktionsziel die aktuelle Datei `friends.php` (optional).

Ergänzen Sie nun die Verarbeitung des Formulars im PHP-Block am Beginn der Datei. Prüfen Sie ob eine Anfrage mit einem Formularfeld `action` vorliegt und ob der Wert des Feldes `add-friend`ist. Wenn dem so ist, prüfen Sie ob entsprechende Informationen vorhanden sind (der Name des Freundes) und rufen Sie die Methode `friendRequest` an Ihrer Instanz der Klasse `BackendService` auf. Andernfalls stellen Sie sicher, dass eine Fehlermeldung über dem Formular zum Hinzufügen von Freunden angezeigt wird. Für den Aufruf benötigen Sie eine neue Instanz der Klasse `Friend`.

Anschließend sind weitere Verarbeitungen von Anfragen notwendig. Es empfiehlt sich, jede Anfrage mithilfe des Formularfeldes `action` zu spezifizieren und entsprechend zu behandeln. Achten Sie jedoch darauf ob die Information mittels GET oder POST übertragen wird und nutzen Sie die passenden PHP-Variablen für die Prüfung. Zu realisieren ist:

- Das Annehmen von Freundschaftsanfragen: Erweiteren Sie Ihre Ansicht (vgl. Modal) so, dass beim Bestätigen der Anfrage der anzunehmende Freund und ein passender Aktionsidentifizierer übermittelt wird. Besprechen Sie im Praktikum ggf. mögliche Strategien, wie hier vorzugehen ist. Ergänzen Sie in PHP die passende Anfrageverarbeitung, so dass der passende Aktionsidentifizierer genutzt wird und anschließend die korrekte Methode am `BackendService` aufgerufen wird.
- Das Ablehnen von Freundschaftsanfragen: Analog zur Annahme mit veränderten Aktionsidentifizierer und Aufruf am `BackendService`.
- Das Entfernen von Freundschaften: Diese Funktion wird später in der Chat-Ansicht genutzt und muss in der Freundesliste verarbeitet werden. Reagieren Sie geeignet auf eine Aktionsidentifizierer (über GET oder POST) und rufen Sie die entsprechende Methode am `BackendService` auf.

Hinweis: Achten Sie auf die Reihenfolge der Logik. Verarbeiten Sie erst die Nutzeranfragen und laden Sie anschließend die Daten vom Backend.

Abschließend müssen Sie über Ihre PHP-Implementierung den Server, die Collection ID und insbesondere das aktuelle Token an den Client kommunizieren. Mit Bezug zu Aufgabe 3 wurde hierfür an einer geeigneten Stelle Konstanten im Dokument definiert, die Sie später in Ihrer JavaScript-Implementierung genutzt haben. Dadurch, dass Sie PHP nutzen, um eine Vorverarbeitung zur Erzeugung eines Ergebnisdokumentes zu realisieren, können Sie an dieser Stelle ebenso JavaScript-Fragmente produzieren. In diesem Fall nutzen wir dies, um Informationen an die Client-Anwendung zu übergeben. Sie können etwas der folgenden Form nutzen, passen Sie dies an Ihre aktuelle Lösung entsprechend an:

```
<script>
    window.chatToken = "<?= $_SESSION['chat_token'] ?>";
    window.chatCollectionId = "<?= CHAT_SERVER_ID ?>";
    window.chatServer = "<?= CHAT_SERVER_URL ?>";
</script>
```

Teilaufgabe h: Chat-Ansicht

Realisieren Sie die Chat-Ansicht. Hier ist es notwendig sicherzustellen, dass ein Query-Parameter mit dem Chat-Ziel übergeben wurde und das der Nutzer authentifiziert ist. Wurde kein Chat-Ziel übergeben, soll der Nutzer zurück zur Freundesliste geleitet werden, wurde der Nutzer nicht angemeldet soll der Nutzer zur Login-Ansicht geleitet werden.

Beginnen Sie mit einem PHP-Block am Anfang der Datei, welcher die Datei `start.php` lädt und prüfen Sie ob die Session-Variable `user` gesetzt und nicht leer ist. Anschließend prüfen Sie ob der Query-Parameter für das Chat-Ziel (z.B. `friend`) existiert und nicht leer ist. Stimmen Sie dies mit der Implementierung der Freundesliste ab.

Abschließend müssen Sie über Ihre PHP-Implementierung den Server, die Collection ID und insbesondere das aktuelle Token sowie das Chat-Ziel an die Client-Anwendung übergeben. Gehen Sie hier analog zum abschließenden Schritt in der Freundesliste vor.

Teilaufgabe i: Nutzerprofil-Einstellungen

Realisieren Sie die Formulardatenverarbeitung für die Einstellungen des Nutzerprofils. Ziel ist es, alle definierten Formularfelder in der Nutzerklasse zu aktualisieren und anschließend den Nutzer im Backend zu speichern.

Beginnen Sie mit einem PHP-Block am Anfang der Datei, welcher die Datei `start.php` lädt und prüfen Sie ob die Session-Variable `user` gesetzt und nicht leer ist. Leiten Sie ggf. auf die Login-Seite weiter.

Erweitern Sie anschließend die Implementierung der Klasse `User` so, dass für jede Profilinformation in Ihren Einstellungen ein passendes Attribut in der Klasse vorgesehen ist. Ergänzen Sie für diese Attribute Getter und Setter.

Laden Sie anschließend den Nutzer über den `BackendService` unter Verwendung der Session-Variable `user` und verarbeiten Sie das Formular nach einem üblichen Vorgehen. Speicheren Sie abschließend die Eingaben unter Verwendung der Methode `saveUser` an dem `BackendService`.

Stellen Sie desweiteren sicher, dass das Formular ausgefüllt ist mit bereits bekannten Informationen. Insbesondere das Ausfüllen von Input-Radio- und Select-Elementen ist hier entsprechend umzusetzen.

Teilaufgabe j: Nutzerprofil-Ansicht

Realisieren Sie eine Nutzerprofil-Ansicht. Ziel ist es, alle Informationen aus einer Instanz der Klasse `User` geeignet darzustellen. Geladen wird der Nutzer, welcher über einen Query-Parameter angegeben wurde.

Beginnen Sie mit einem PHP-Block am Anfang der Datei, welcher die Datei `start.php` lädt und prüfen Sie ob die Session-Variable `user` gesetzt und nicht leer ist. Leiten Sie ggf. auf die Login-Seite weiter.

Laden Sie anschließend den Nutzer basierend auf dem im Query-Parameter vorgegebenen Namen des Nutzers. Sollte kein Nutzer angegeben sein, können Sie auf die Freundesliste weiterleiten.

Bewertungskriterien

- Fehlerfreie und funktionierende Umsetzung der Chat-Anwendung, dies umfasst insbesondere:
    - Anmelden mit Nutzernamen und Passwort
    - Registrieren eines neuen Nutzers
    - Anzeige von Freunden
    - Hinzufügen eines Freundes
    - Annehmen oder Ablehenen von Anfragen
    - Entfernen von Freunden
    - Chatten mit einem Freund
    - Anzeige und Bearbeiten von Nutzerprofilen (nur bei dreier Teams)
- Jedes Teammitglied muss eigene Funktionsbestandteile verantworten, insbesondere Registrieren, Freundesliste und Nutzerprofil-Einstellungen sind im Team aufzuteilen (vgl. Vorbereitung)
- PHP-, Bild-, CSS- und JS-Dateien korrekt verlinkt werden, Bootstrap kann über CDN-Ansätze integriert werden