Hochschule RheinMain **Studiengang Medieninformatik**
Wolfgang Weitz

Webbasierte Anwendungen

13. April 2022

Wir arbeiten in dieser Veranstaltung **nur unter Linux** und nach Möglichkeit durchgängig mit dem Editor "VisualStudio Code" `https://code.visualstudio.com/`. Er wurde unter der Leitung des großmächtigen Erich Gamma entwickelt, unterstützt **alle** in der Veranstaltung benötigten Sprachen und Formate (Python, Java, Gradle, TypeScript, JSON, XML, HTML, CSS) sowie die Frameworks, die wir im Verlauf brauchen werden.

Technische Hinweise: CGI-Skripte müssen (auf unserem MI-Webserver) im Unterordner **`public_html/cgi-bin/`** Ihres Home-Directories liegen und der Dateiname muß mit `.cgi` enden (statt z.B. mit `.py`). Beachten Sie bitte, dass `public_html` *nicht* im URL-Pfad auftaucht.

Wenn Sie von Ihrem Faltrechner über das CampusWLAN oder zu Hause aus auf dem CGI-Verzeichnis arbeiten wollen, bietet sich die **VSCode Extension "Remote - SSH"** (`ms-vscode-remote.remote-ssh`) an. Vermöge ihrer können Sie aus VSCode heraus per `ssh` direkt auf Ihrem Hochschul-Verzeichnis arbeiten, und das nicht nur beim CGI-Entwickeln. Wenn Sie in VSCode ein Terminal öffnen, landen Sie auch direkt (per ssh) auf dem verbundenen Zielrechner und können dort Kommandos absetzen. Die Extension ist toll, man kann sogar auf dem Zielrechner transparent debuggen etc, das sehen wir später noch. Mehr Infos dazu auch unter `https://code.visualstudio.com/docs/remote/ssh`

Sollten `public_html` und/oder `cgi-bin` in Ihrem Homedirectory nicht angelegt sein oder die Rechte nicht stimmen, können Sie (auf einem der Hochschulrechner) das Kommando `fix-cgi-bin` ausführen, welches das korrigiert.

CGI-Dateien müssen **Ausführungsrechte** haben (diese müssen Sie nach Anlegen der Datei explizit setzen, z.B. `chmod 700 a1.cgi`, vgl. "Einführung in die MI"). Damit der Webserver weiß, dass diese Textdatei mit Python3 auszuführen ist, muss sie *als erste Zeile* einen entsprechenden **Hashbang-Pfad** enthalten (`#!/usr/bin/env python3`).

Aufgabe 1 (Entwickler-Tools, einfaches CGI-Skript)

Bitte öffnen Sie in VSCode Ihr `public_html/cgi-bin`-Verzeichnis (z.B. im File-Menü "Open Folder..." oder durch Angabe des Ordners beim Start von Code, z.B. "`cd ~/public_html/cgi-bin; code .`"). Von "Programmieren 3" sollten Sie noch die Python-Extension installiert haben.

Legen Sie nun bitte Ihr erstes CGI-File `a1.cgi` an. Dieses Python3-Skript soll eine **Funktion `dump()` bereitstellen**, die (alphabetisch sortiert) **alle Environment-Variablen** und danach **alle CGI-Parameter** (beides Mal Name und Wert) untereinander als HTML-Bulletliste ausgibt (`<ul>...<li>...</ul>`).

Am Ende des Skripts (außerhalb der Funktion `dump()`) erzeugen Sie die Antwort, indem Sie dem Client den `Content-Type` mitteilen und danach `dump()` aufrufen. Vergessen Sie die trennende Leerzeile zwischen Antwort-Header und Body (hier dem `dump()`-Output) bitte nicht. Binden Sie auch das `cgitb`-Modul ein und aktivieren Sie es gleich nach Ausgabe der HTTP-Headerszeilen (also gleich nach der trennenden Leerzeile), damit Sie lesbares Feedback bekommen, sollte Ihre CGI-Skript abstürzen.

Seien Sie bitte auch eingedenk, Ihr CGI-Skript ausführbar zu machen, und probieren Sie dann einen Abruf über `http://www.mi.hs-rm.de/~jbiff017/cgi-bin/a1.cgi` (bitte Usernamen anpassen, falls Sie nicht Jöndhard Biffel sind).

Hängen Sie nun weitere **Pfadkomponenten und einige Query-Parameter** an die URL an und stellen Sie anhand Ihrer Ausgabe fest, ob und wie Sie im Skript darauf zugreifen könnten (Beispiel: `http://www.mi.hs-rm.de/~jbiff017/cgi-bin/a1.cgi/docs/web/buch.pdf?ausgabe=2&autor=Friedfert+von+Senkel`).

Webbrowser enthalten oft hilfreiche Entwicklungswerkzeuge. Starten Sie bitte den Browser `chromium-browser` mit einer dieser URLs und öffnen Sie die **Entwicklertools** mit der Tastenkombination strg | shift | i. Wählen Sie den **Reiter "Network"** aus und laden Sie die URL erneut (normaler

Browser-Reload). Sie sehen die durchgeführten **Netzwerkanfragen**. Wenn Sie einen Eintrag anklicken, erhalten Sie die HTTP **Anfrage- und Antwort-Headerfelder** und weitere Werte. Rufen Sie nun bitte die HSRM-Homepage `http://www.hs-rm.de` ab und sehen Sie sich an, wieviele HTTP-Anfragen durch den Abruf der einen URL nachfolgend ausgelöst wurden.

Aufgabe 2 (CGI und Session-Management)

Das Thema des Auseinanderhaltes von Session-Kontexten verschiedener Dialogstränge, deren (Teil-)Schritte beliebig zeitlich vermischt beim Server ankommen, ist ein klassisches Web-Problem (... weil HTTP zustandslos ist), und zwei Ansätze wurden in der Vorlesung besprochen:

1. Sie legen beim **ersten** Abruf (typischerweise per HTTP-Methode GET) z.B. des HTML-Formulars über Ihr CGI-Skript eine **neuen Benutzersitzung** an, indem Sie ein **verstecktes Eingabefeld** (`<input type="hidden"...>`) in Ihr Formular integrieren, das eine zufällige, noch unbenutzte Session-ID enthält, welche bei Abschicken des ausgefüllten Formulars durch den Benutzer wieder an das CGI zurückkommt, so dass die Antwort anhand der ID einem Sitzungskontext zugeordnet werden kann.

   Diese ID wird ebenso bei allen Folgeantworten **derselben Session** eingebaut und so zwischen Server und Client hin- und hergespielt, die (eigentlich unabhängigen) Dialogschritte werden auf diese Weise miteinander verbunden (verschiedene Benutzersitzungen haben also auch unterschiedliche IDs – das ist ja genau der Sinn – nur innerhalb einer Benutzer-Dialogfolge wird die jeweilige ID hin- und hergespielt, um einen Sitzungs-Zusammenhang herzustellen).

2. Sie **verwenden Cookies** (das Prinzip mit dem Hin- und Herspielen einer je Dialogstrang eindeutigen ID zwischen Browser und Server bleibt dasselbe, nur dass der in den Browser eingebaute Cookie-Mechanismus das Senden der ID vereinfacht, indem er sie (als Cookie-Attribut) jedem Folgerequest derselben Dialogstrecke hinzufügt).

Da – wie wir in den vergangenen Semestern leidvoll erfahren haben – E-Learning ein Trend ist, wollen wir die Welle reiten und ein Python-CGI-Skript entwickeln, das Menschen beim Kopfrechnen (einfache Additionen) trainiert. Bitte implementieren Sie **zwei Kopfrechner-CGI-Skripte** `a2-hidden.cgi` und `a2-cookie.cgi`, die das Sessionmanagement-Problem a) mit Hidden Input Fields bzw. b) per Cookie lösen.

Wenn unser CGI-Skript (je Benutzersitzung) **erstmalig** (= per **`HTTP GET`**) aufgerufen wird, bekommt der Benutzer eine Rechenaufgabe und ein HTML-Formular für die Antwort vorgelegt (s. Screenshots weiter hinten).

Bei der Erzeugung dieses Formulars würfelt die Anwendung eine (zufällige) Session-ID aus, damit alle Folgeanfragen diesem Benutzer zugeordnet werden können. Bitte merken Sie sich serverseitig unter mit **Session-ID** als Schlüssel ein Dictionary mit den **Sitzungsdaten** wie der Startzeit und nötigen Zählern (s.u.) – in einer Python `shelve`-Datenbank[1]. Weitere Session-Daten können später dort ergänzt werden. Sie können dem Sitzungsdaten-Dictionary jederzeit eigene Einträge zur Umsetzung der Anwendung hinzufügen oder Einträge ändern.

Die Anwendung soll in jeder Runde **zwei ganzzahlige Zufallszahlen** zwischen 1 und 17 ziehen (Python-Modul `random`) und in der generierten HTML-Seite anzeigen, auf welcher der Benutzer nach der **Summe der beiden Zufallszahlen** gefragt wird (z.B. "Was ist 17 + 4?"). Er trägt dann sein Ergebnis in ein HTML-Formularfeld ein und schickt die Formulardaten zurück an das CGI-Skript. Die erwartete Lösung (hier: 21) merkt sich das CGI-Skript unter der Session-ID serverseitig in der Shelve-DB.

Der Server überprüft, ob die **Antwort stimmt**, und liefert ein passendes Ergebnis zurück ("Das ist richtig.", "Total falsch, richtig wäre ... gewesen") gefolgt von der **nächsten Aufgabe**.

[1] ... und versprechen Sie, dass das das einzige Mal in Ihrem Leben ist, wo Sie sowas machen, denn shelve ist nicht transaktionssicher und sollte in Anwendungen mit parallelen Zugriffen, wie es Web-Anwendungen oft sind, daher nicht verwendet werden. Aber es ist einfach, und "richtige" Datenbanken behandeln wir später mit JPA und Spring. Und da es uns hier um's Kopfrechnen und Session-Handling geht, sind wir an der Stelle mal pragmatisch.

Wenn der Benutzer **dreimal direkt nacheinander** richtig geantwortet hat, ist er fertig und bekommt ein Lob, die **Gesamtzahl seiner Versuche** und die **insgesamt benötigte Zeit** (in Sekunden) von der Anzeige der ersten bis zur richtigen Lösung der letzten Aufgabe ausgegeben. Löschen Sie zum Abschluss bitte auch den Eintrag dieser Session in der Sitzungsdaten-Datenbank (siehe letzter Screenshot).

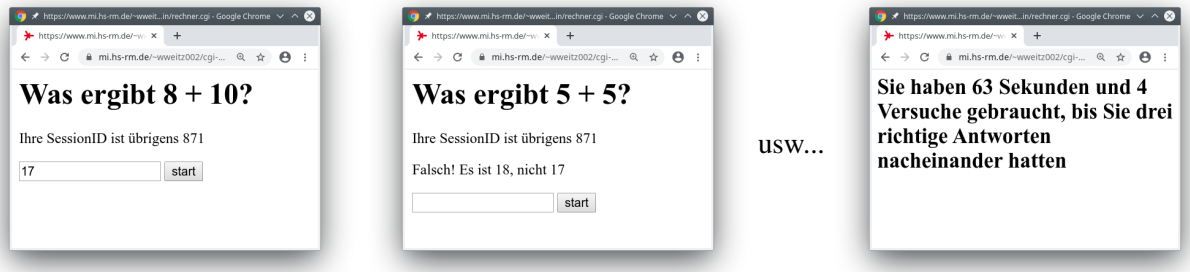
Lassen Sie sich auf der erzeugten Webseite jedes Mal die aktuelle **Session-ID mit ausgeben**. Auf diese Weise können Sie besser nachvollziehen, ob Sie wirklich für eine Dialogfolge über mehrere Schritte hinweg dieselbe SessionID nutzen. Sie können allerdings in Ihrem Browser auch den HTML-Quelltext der Seite betrachten – dort sollten Sie Ihr verstecktes SessionID-Feld mit dem entsprechenden Wert sehen.

Bei der Aufgabe soll **nur die Session-ID** als verstecktes HTML-Feld bzw. in Cookies eingebaut werden – Sie sollten **nicht** noch weitere Sitzungsdaten wie den Versuche-Zähler, die "Musterlösung" oder die zu addierenden Zahlen auf diese Weise in der Server-Antwort verstecken – wenn Sie sich darauf verlassen würden, könnte ein technisch versierter, aber rechenschwacher Betrüger diese versteckten Summanden-Felder z.B. mit den Werten 1 und 2 belegen, bevor er die Formulardaten an den nichtsahnenden Server schickt, und braucht dann nur zu wissen, dass 1+2 = 3 ist (das könnte er notfalls auch googeln).

Sie müssen den ganzen Ballast auch nicht immer hin- und herschicken, denn diese Angaben haben Sie ja serverseitig bei Erzeugung des HTML-Formulars in Ihrem `shelve` abgelegt, und alles, was Sie zum Zugriff darauf benötigen, ist die Session-ID, die Sie ja mit der Formular-Antwort zurückbekommen. Im erzeugten Formular stehen werden die Sitzungsdaten (Name, Aufgabe, ...) also nur *im Ausgabetext verwendet* und stehen als gewöhnliche, sichtbare Ausgaben im erzeugten HTML.

Das ganze Konzept der Sessiondaten-Verwaltung hat den Sinn, die verschiedenen Dialogstränge potentiell verschiedener Benutzer derselben Web-Anwendung auseinanderzuhalten. Bitte **testen** Sie ihre Anwendung daher auch dahingehend, indem Sie von **verschiedenen** Browsern (wirklich verschiedene, z.B. Firefox und Chrome – **nicht** einfach zwei Fenster desselben Browsers) aus mehrere Dialogstränge starten und in zufälliger Reihenfolge fortführen. Die Anwendung (und ihre "Buchführung" je Teilnehmer) darf dabei nicht durcheinander geraten.

Hinweise: Das Python-Modul `time` stellt eine Funktion `time()` bereit, das die Anzahl der Sekunden seit dem 1.1.1970 enthält (dem "Anfang der Epoche", `https://de.wikipedia.org/wiki/Unixzeit`). Hiermit können Sie leicht die geforderten Zeitrestriktionen prüfen.



Aufgabe 3 (Git wünschen)

Ab übernächster Woche benötigen Sie ein individuelles Git-Repository für die Veranstaltung. Darüber finden die wochenweisen Abgaben statt, die zum Erwerb der Praktikumsleistung erforderlich sind. Bitte…

- …loggen Sie sich **bis nächsten Mittwoch (20.04.)** unter `http://www.mi.hs-rm.de/~weitz/cgi-bin/webrepo.cgi` mit Ihrem Informatik-Account ein, um ein Git-Repository-Ordner auf `https://scm.mi.hs-rm.de/rhodecode` zu bestellen (dies ist ein spezieller Ordner für die Web-Veranstaltung und unabhängig von dem persönlichen Git-Ordner unter `stud/`*`username`* auf demselben Server.
- Eine Git-Einführungs gibt es nächste Woche im Softwaretechnik-Praktikum.