Spezifikation

StockService

Version 3.0

**Gruppe 1**

|  |  |  |
| --- | --- | --- |
| Angermeier Rainer | 1155727 | angermeier.rainer@gmail.com |
| Atzenhofer Thomas | 1155249 | thomas.atzenhofer@gmx.net |
| Dansachmüller Olivia | 1155584 | olivia.d@gmx.at |
| Eckmaier Alexander | 1157492 | alexander.eckmaier@gmail.com |

Kursleiter

Sametinger Johannes

Plösch Reinhold

Weinreich Rainer

Stritzinger Alois

Kurs

259.028 – Service Engineering KV

Datum

19.01.2015

Inhalt

[1 Einleitung 4](#_Toc409379056)

[1.1 Zweck und Ziel 4](#_Toc409379057)

[1.2 Name, Einsatzbereich, Abgrenzung 4](#_Toc409379058)

[2 Technologieauswahl 5](#_Toc409379059)

[2.1 Server 5](#_Toc409379060)

[2.2 Webclient 6](#_Toc409379061)

[2.3 Mobiler Client 6](#_Toc409379062)

[2.4 Cloud-Dienst 6](#_Toc409379063)

[2.5 Authentifizierung 7](#_Toc409379064)

[2.6 Datenformate 7](#_Toc409379065)

[2.7 Protokolle 8](#_Toc409379066)

[3 Umsetzung der Authentifizierung 9](#_Toc409379067)

[4 REST 10](#_Toc409379068)

[4.1 Adressierbarkeit 10](#_Toc409379069)

[4.2 Zustandslosigkeit 10](#_Toc409379070)

[4.3 Schnittstellen/Operationen 10](#_Toc409379071)

[4.4 Entkopplung von Ressourcen und Repräsentation 10](#_Toc409379072)

[4.5 Umsetzung von REST 11](#_Toc409379073)

[5 Funktionale Anforderungen 14](#_Toc409379080)

[5.1 Aktienübersicht 14](#_Toc409379081)

[5.2 Aktienhistorie 14](#_Toc409379082)

[5.3 Registrierung 15](#_Toc409379083)

[5.4 Login 15](#_Toc409379084)

[5.5 Aktienkauf 16](#_Toc409379085)

[5.6 Aktienverkauf 16](#_Toc409379086)

[5.7 Transaktionsübersicht (Portfolio) 17](#_Toc409379087)

[5.8 Passwortänderung 17](#_Toc409379088)

[5.9 Logout 17](#_Toc409379089)

[6 Fehlersuche 18](#_Toc409379090)

[6.1 Client-Server-Kommunikation 18](#_Toc409379091)

[6.2 Client-Server-Kommunikation (Error-Checking) 20](#_Toc409379092)

Abbildungsverzeichnis

[Abbildung 1: Aktienübersicht 15](#_Toc409378973)

[Abbildung 2: Aktienhistorie 16](#_Toc409378974)

[Abbildung 3: Registrierung 16](#_Toc409378975)

[Abbildung 4: Login 17](#_Toc409378976)

[Abbildung 5: Kauf von Aktien 17](#_Toc409378977)

[Abbildung 6: Verkauf von Aktien 18](#_Toc409378978)

[Abbildung 7: Ändern des Passwortes 18](#_Toc409378979)

[Abbildung 8: Logout 19](#_Toc409378980)

# Einleitung

Diese Einleitung soll einen detaillierten Überblick über die Softwareanforderungsspezifikation (SAS) verschaffen. Sie beinhaltet verschiedene Aspekte wie den Zweck und das Ziel der SAS und soll dazu beitragen, verschiedene Begriffe zu beschreiben und etwaige Unklarheiten auszuräumen. Des Weiteren werden in der Einleitung auch der Name des Softwareprodukts und dessen Einsatzbereich festgelegt sowie Abgrenzungen zu anderen Produkten erläutert.

## Zweck und Ziel

Zweck und Ziel dieser SAS ist es dem Auftragnehmer beziehungsweise den Softwareentwicklern in schriftlicher Form die Anforderungen an das Softwareprodukt gut strukturiert, prägnant, verständlich, eindeutig, nachvollziehbar und vollständig darzulegen.

## Name, Einsatzbereich, Abgrenzung

Der zu realisierende Service mit Web-Client wird im Folgenden als StockService bezeichnet und als Server-Client-Applikation realisiert. Aufgabe des Servers ist es, Aktienkurse von zehn an der Börse notierten Firmen zu sammeln, zu speichern und für den Client verfügbar zu machen. Dem User soll es folglich möglich sein, verschiedene Informationen über diese Aktien am Client abzurufen und auf einer Website zu betrachten. Dabei wird zwischen einem Standard-, das heißt nicht eingeloggten, und einem eingeloggten User unterschieden. Ersterem stehen folgende Möglichkeiten zur Verfügung: Er kann sich die Liste mit allen zehn verfügbaren Aktien und den dazugehörigen Basisdaten ansehen und je nach Belieben zusätzliche Details zu einer von diesen einblenden lassen. Diese zusätzlichen Details bestehen aus Informationen über die letzte Zeit bis zum aktuellen Datum sowie über Höchst- bzw. Tiefstände und sonstige Veränderungen. Des Weiteren hat der Standard-User auch noch die Möglichkeit sich zu registrieren und damit die Fähigkeit zu erwerben, zu einem eingeloggten User zu werden. Diesem stehen wiederum zusätzliche Möglichkeiten offen: Er kann sein User-Portfolio, dessen Daten vom Server in einer Datenbank gespeichert und fortlaufend aktualisiert werden, verwalten, das heißt neue bzw. zusätzliche Aktien kaufen, bereits gekaufte Aktien wieder verkaufen und sein Passwort ändern. Zusätzlich kann sich ein eingeloggter User auch ausloggen und damit wieder zu einem Standard-User werden.

Um Aktien zu kaufen benötigt der User Geld. Daraus folgt, dass er nicht beliebig viele Aktien, sofern verfügbar, kaufen kann. Auch kann er sein Konto nicht überziehen. Die Kosten für einen Aktienkauf setzen sich aus dem jeweiligen Tagespreis multipliziert mit der gekauften Stückanzahl zusammen. Das Verkaufen von Aktien führt zu einem Geldzufluss, der sich ebenfalls aus dem jeweiligen Tagespreis multipliziert mit der verkauften Stückanzahl ergibt.

# Technologieauswahl

Dieser Teil der Arbeit beschäftigt sich mit den von uns ausgewählten Technologien, Frameworks, Datenformaten und Protokollen. Es wird jeweils die verwendete Technologie genannt und kurz vorgestellt.

## Server

Der Server basiert auf dem Spring Framework. Das Springframework ist ein open-source Framework für Java-Plattformen, das die Entwicklung von Java Enterprise Applikationen erleichtern und unterstützen soll. Die großen Vorteile dieses Frameworks liegen nicht nur in der Möglichkeit der aspektorientierten Programmierung und der Dependency-Injection, sondern auch vor allem in der erleichterten Entwicklung von RESTful Applikationen, was ausschlaggebend für die Auswahl dieses Frameworks war. Das Springframework ermöglicht es durch Annotationen, ein effizientes und dynamisches Serververhalten abzubilden.

## Webclient

Am Client bedienten wir uns ebenfalls eines open-source Frameworks, genauer gesagt des von Google entwickelten AngularJS Frameworks. Auf Javascript basierend, ermöglicht AngularJS die Erstellung von HTML und JavaScript Webanwendungen nach einem MVC Muster. AngularJS ermöglicht nicht nur wie z.B. in JQuery die Manipulation von DOM Knoten in einem Dokument, sondern die komplette Erstellung eines DOM Baumes zur Laufzeit. Dies geschieht durch ein clientseitiges Model, das die gesamte Logik enthält. Dieses Modell wird in einem Controller definiert und dieser Controller zu einem Modul zusammengefasst, welches später in die HTML Datei eingebunden wird. Außerdem ermöglicht AngularJS die Erstellung von Direktiven. Direktive sind sozusagen selbsterstellte HTML Tags, die es ermöglichen, Funktionalität bzw. Code auszulagern und somit die Applikation übersichtlicher zu gestalten. Ein wesentlicher Grund für die Auswahl dieser Technologie war aber die Unterstützung des Frameworks bei REST Schnittstellen und der als Standard definierte Umgang mit dem JSON Format.

## Mobiler Client

Für die Bearbeitung der Aufgabe 2, Entwicklung eines mobilen Clients, haben wir uns für die Verwendung von Spring for Android entschieden.

Spring for Android ist eine Weiterentwicklung des Spring Frameworks, das wir schon für die Durchführung der Aufgabe 1 verwendet haben.

Mit Spring for Android ist es möglich, Spring-Client-Bibliotheken auch in Android-Umgebungen nutzen zu können. Das Android-Framework liefert einen [REST-Client](http://static.springsource.org/spring-android/docs/1.0.x/reference/html/rest-template.html), über den sich RESTful Web Services aus nativen Android-Anwendungen heraus ansprechen lassen.

## Cloud-Dienst

Für die Bearbeitung der Aufgabe 3, Umsetzung der Lösung als Prototyp in der Cloud, haben wir uns für die Verwendung von Amazon Web Services entschieden.

Amazon Elastic Compute Cloud (Amazon EC2) ist ein Web-Service, der anpassbare Rechenkapazität in der Cloud bietet und ist darauf ausgelegt, Cloud Computing für Entwickler zu erleichtern. Mit einer einfachen Web-Service-Oberfläche ist es möglich die Kapazität zu erhalten und zu konfigurieren (Anpassung an Datenverarbeitungsanforderungen). Man behält die vollständige Kontrolle über die Rechenressourcen sowie die Ausführung in der Rechenumgebung von Amazon. Es bietet den Entwicklern Tools, um ausfallsichere Anwendungen zu erstellen und diese von Fehlerszenarien zu isolieren.

AWS Elastik Beanstalk ist ein benutzerfreundlicher Service zum Bereitstellen und Skalieren von Webanwendungen und –services. Der Benutzer lädt den Code hoch und Elastic Beanstalk übernimmt automatisch die Bereitstellung, von der Kapazitätsbereitstellung, Lastverteilung und automatischen Skalierung bis hin zur Statusüberwachung der Anwendung. Der Benutzer behält jedoch die vollständige Kontrolle über die Ressourcen hinter der Anwendung und kann jederzeit darauf zugreifen.

## Authentifizierung

Für die Umsetzung der Authentifizierung haben wir eine Erweiterung des Spring Frameworks gewählt, die Spring Security. Mit Spring Security ist es möglich, eine Liste von Usern auf der Datenbank abzuspeichern.

## Datenformate

Als Standard für den Datenaustausch zwischen dem Server und dem Client haben wir das JSON Format gewählt. Weiters kommuniziert unser Server mit dem YAHOO-Finance Server, der aktuelle und vergangene Kurswerte ausgewählter Aktien ebenfalls im JSON Format an den Server übermittelt. Abgefragt werden die Daten mit einer YAHOO-spezifischen Query Language, der Yahoo-Query-Language (YQL).

## Protokolle

Grundlegend für die Auswahl der Frameworks und Datenformate war die Auswahl der Protokolle. Die Auswahl der Protokolle fiel auf das HTTP bzw. das REST Protokoll, wo wir unter Verwendung der GET, POST, PUT und DELETE Request-Methoden die Kommunikation zwischen Server und Client sicherstellen.

# Umsetzung der Authentifizierung

Wie schon erwähnt, erfolgt die Umsetzung der Authentifizierung mithilfe von Spring Security.

Folgende Services sind nur mit erfolgreicher Authentifizierung abrufbar:

* Passwortänderung
* Transaktionsübersicht im Portfolio
* Kaufen und Verkaufen von Aktien/Transaktionen

Alle genannten Calls müssen einen HTTP-Basic Authentication Header mitschicken.

Folgende Services sind auch ohne Authentifizierung abrufbar:

* Registrierung
* Aktienübersicht
* Aktien-History

Die Umsetzung erfolgt in den Controller Klassen:

* FinanceController:
  + Mapped auf /finance (für alle zugänglich)
  + Methoden:
    - getStocks: StockOverview
    - getStockHistory: StockHistory
* SecuredController:
  + Mapped auf /secured (nur für eingeloggte bzw. registrierte Nutzer zugänglich)
  + ROLE\_LOGGEDIN
  + Methoden:
    - addTransaction: Transaktionen zu Benutzerkontos hinzufügen
    - getUserTransactions: Transaktionen eines Benutzerkontos abrufen
    - updateUser: Passwort eines Benutzerkontos ändern
* UserController:
  + Mapped auf /users (für alle zugänglich)
  + Methoden:
    - registerUser: User registrieren

# REST

Wie schon erwähnt, ermöglichte die Verwendung der Protokolle HTTP bzw. REST bereits die Entwicklung der Aufgabe 1 REST-konform durchzuführen.

Nachstehend werden die Grundkonzepte von REST anhand unseres Web Services diskutiert.

## Adressierbarkeit

Jeder REST-konforme Dienst hat eine eindeutige Adresse. Dadurch können Ressourcen von anderen Komponenten des Systems über logische URIs angesprochen werden.

## Zustandslosigkeit

Jede REST-Nachricht von Client zu Server enthält alle Informationen, die für den Server bzw. Client notwendig sind, um die Nachricht zu verstehen.

Die Vorteile von Zustandslosigkeit sind die Skalierbarkeit, Verständlichkeit und Zuverlässigkeit.

## Schnittstellen/Operationen

Der Zugriff auf jede Ressource muss über einen einheitlichen Satz von Standardmethoden erfolgen. Beispiele hierfür sind die Standard-HTTP-Methoden, die wir auch für unsere Ressourcen verwendet haben.

## Entkopplung von Ressourcen und Repräsentation

Dienste können unterschiedliche Darstellungsformen (Repräsentationen) haben. Ein REST-konformer Server kann verschiedene Repräsentationen einer Ressource ausliefern, z.B. in verschiedenen Sprachen oder Formaten ([HTML](http://de.wikipedia.org/wiki/HTML), [JSON](http://de.wikipedia.org/wiki/JSON) oder [XML](http://de.wikipedia.org/wiki/XML)) oder auch die Beschreibung oder Dokumentation des Dienstes.

## Umsetzung von REST

### Aktienübersicht

Dieser Service bildet die funktionale Anforderung der Aktienübersicht ab. Er wird mithilfe der HTTP-Methode GET realisiert und liefert eine Liste der verfügbaren Aktien inklusive des Kürzels, des vollständigen Namens, der Aktienanzahl, des aktuellen Preises, des aktuellen Höchst- bzw. Tiefstands und der Kursänderung.

Der Call hierfür lautet:

* GET - /finance/stocks

Die Responses sehen wie folgt aus:

* success: Liste mit Stock Objekten: symbol, name, volume, lastTradedpriceOnly, daysHigh, daysLow, change

### Aktienhistorie

Dieser Service bildet die funktionale Anforderung der Aktienhistorie ab. Er wird mithilfe der HTTP-Methode GET realisiert und liefert die historische Aktienkursentwicklung des letzten Monats.

Der Call hierfür lautet:

* GET - /finance/stocks/{stocksymbol}/history

Die Responses sehen wie folgt aus:

* success: Liste mit StockHistory Objekten: symbol, date, close

### Registrierung

Dieser Service bildet die funktionale Anforderung der Registrierung ab. Er wird mithilfe der HTTP-Methode POST realisiert und legt den neuen User in der Datenbank an.

Der Call hierfür lautet:

* POST - /users?un=un&pw=pw

Die Responses sehen wie folgt aus:

* success: Keine spezifischer Rückgabewert

### Passwortänderung

Dieser Service bildet die funktionale Anforderung der Passwortänderung ab. Er wird mithilfe der HTTP-Methode PUT realisiert. Es wird ein Update auf den jeweiligen user realisiert. Des Weiteren befindet sich dieser Service unter /secured und muss daher beim Call einen HTTP-Basic Authentication Header mitschicken - um erfolgreich aufgerufen zu werden. (Benutzer muss existieren)

Der Call hierfür lautet:

* PUT - /secured/users/{username}?pw=pw

Die Responses sehen wie folgt aus:

* success: Kein spezifischer Rückgabewert

### Transaktionsübersicht (Portfolio)

Dieser Service bildet die funktionale Anforderung der Transaktionsübersicht ab. Er wird mithilfe der HTTP-Methode GET realisiert und liefert die Transaktionsinformationen des jeweiligen users. Des Weiteren befindet sich dieser Service unter /secured und muss daher beim Call einen HTTP-Basic Authentication Header mitschicken - um erfolgreich aufgerufen zu werden. (Benutzer muss existieren)

Der Call hierfür lautet:

* GET - /secured/finance/transactions

Die Responses sehen wie folgt aus:

* success: Liste der Transaktionen (stocksymbol, stockname, amount, date, price, transactionType), Liste Aktien-Portfolio (Gruppierung der Transaktionen nach Aktie)

### Transaktion hinzufügen (Aktienkauf, Aktienverkauf)

Dieser Service bildet die funktionale Anforderung des Aktienkaufs und des Aktienverkaufs ab. Er wird mithilfe der HTTP-Methode POST realisiert und fügt Transaktionen zur Transaktionsübersicht hinzu. Des Weiteren befindet sich dieser Service unter /secured und muss daher beim Call einen HTTP-Basic Authentication Header mitschicken - um erfolgreich aufgerufen zu werden. (Benutzer muss existieren)

Der Call hierfür lautet:

* POST - /secured/finance/transactions?symbol=YHOO&amount=20&isSell=true/false

Die Responses sehen wie folgt aus:

* success: Liste der Transaktionen (stocksymbol, stockname, amount, date, price, transactionType), Liste Aktien-Portfolio (Gruppierung der Transaktionen nach Aktie)

# Funktionale Anforderungen

Im folgenden Abschnitt werden sämtliche funktionale Anforderungen der Server-Client-Applikation übersichtlich dargestellt und im Detail erläutert. Diese decken das gesamte Spektrum an Möglichkeiten für User ab.

## Aktienübersicht

Die Aktienübersicht steht allen Usern zur Verfügung. Sie besteht aus einer Liste aller verfügbaren Aktien, in diesem Fall zehn. Zu jeder Aktie sind der vollständige Name und die Kursänderung ersichtlich. Diese Informationen werden ohne weiteres Zutun des Benutzers neben der entsprechenden Aktie dargestellt. Die Aktienübersicht wird durch die Aktienhistorie erweitert.

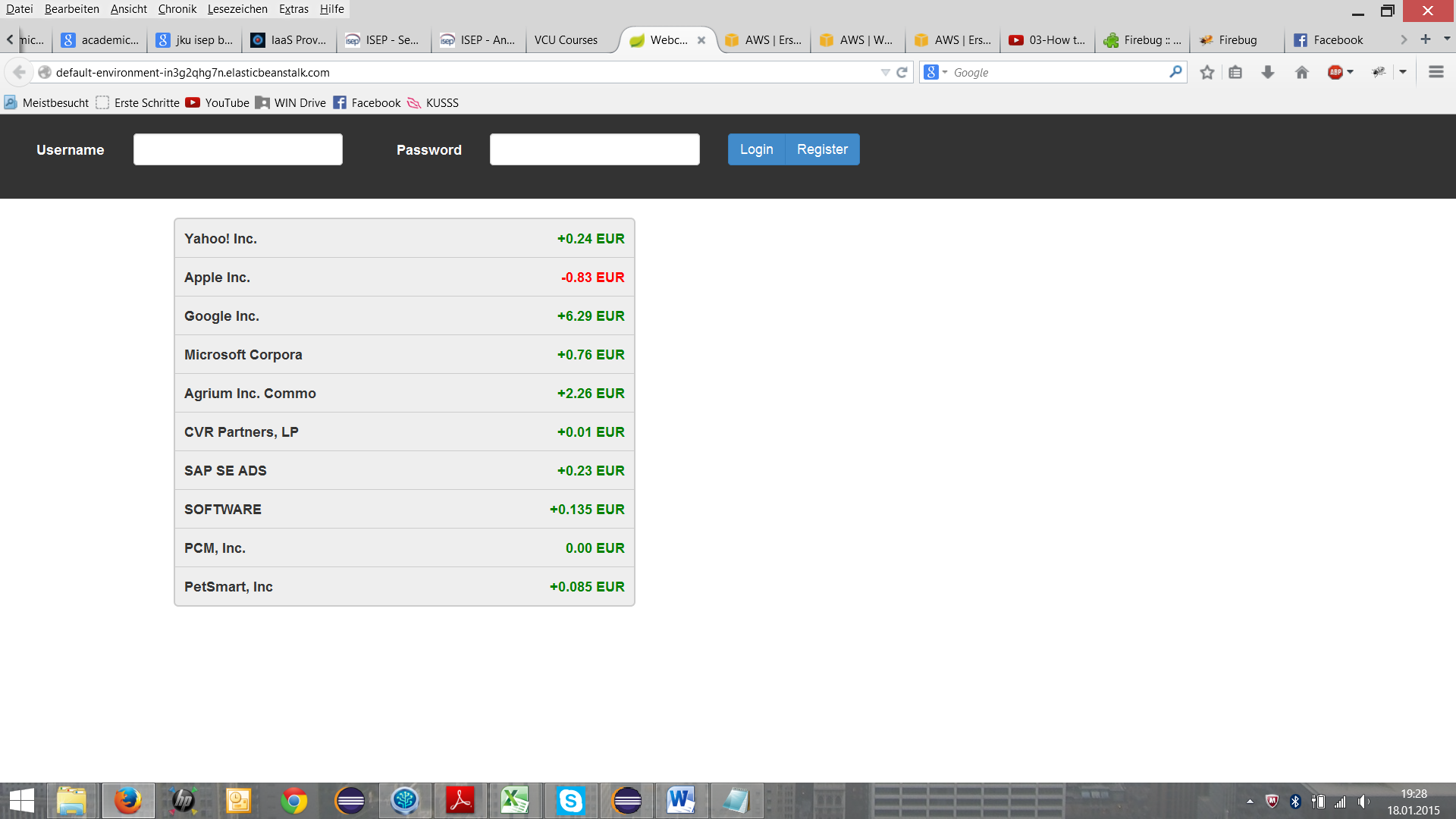


Abbildung 1: Aktienübersicht

## Aktienhistorie

Die Aktienhistorie steht allen Usern zur Verfügung. Sie wird aufgerufen, indem man eine bestimmte Aktie auswählt. Dies führt dazu, dass seitlich davon ein neues Fenster eingeblendet wird, welches Informationen über das Aktienkürzel, den vollständigen Namen, den aktuellen Preis, die Anzahl der gesamten bzw. derzeit verfügbaren Aktien, das Tageshoch, das Tagestief und die Kursänderung des letzten Monats (vom aktuellen Datum an gerechnet) in Form eines Graphen preisgibt.

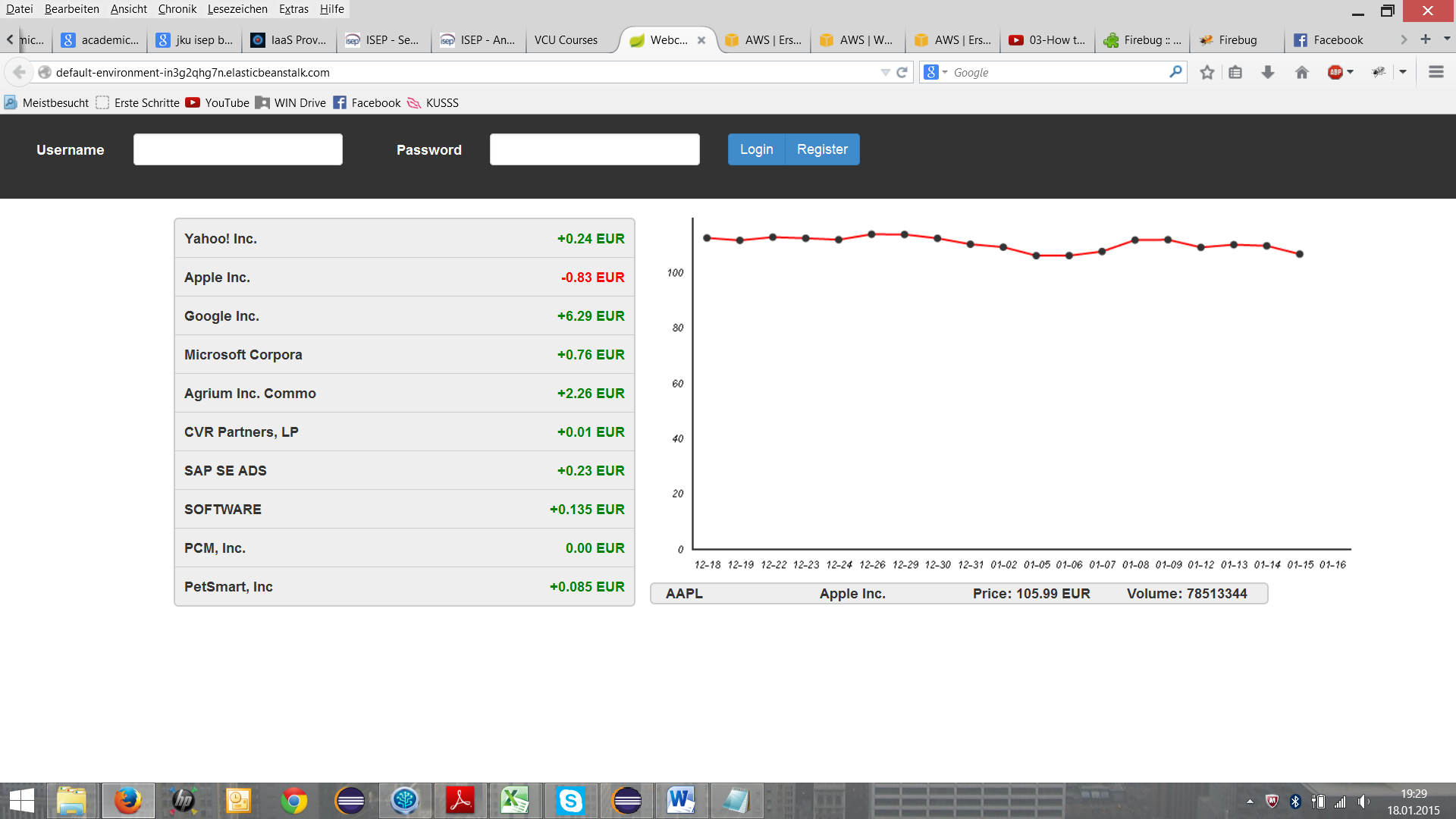


Abbildung 2: Aktienhistorie

## Registrierung

Die Registrierung steht allen Usern zur Verfügung. Sie kann von der Website aus aufgerufen werden, wobei die dafür benötigten Daten, d.h. Benutzername/Username und Passwort, bereits zuvor in entsprechende Felder eingetragen werden müssen. Hierbei ist es wichtig, dass der gewählte Benutzername noch nicht existiert. Sowohl dieser als auch das Passwort müssen gültig sein. Nach dem erfolgreichen Abschluss der Registrierung ist es nun möglich, sich einzuloggen.

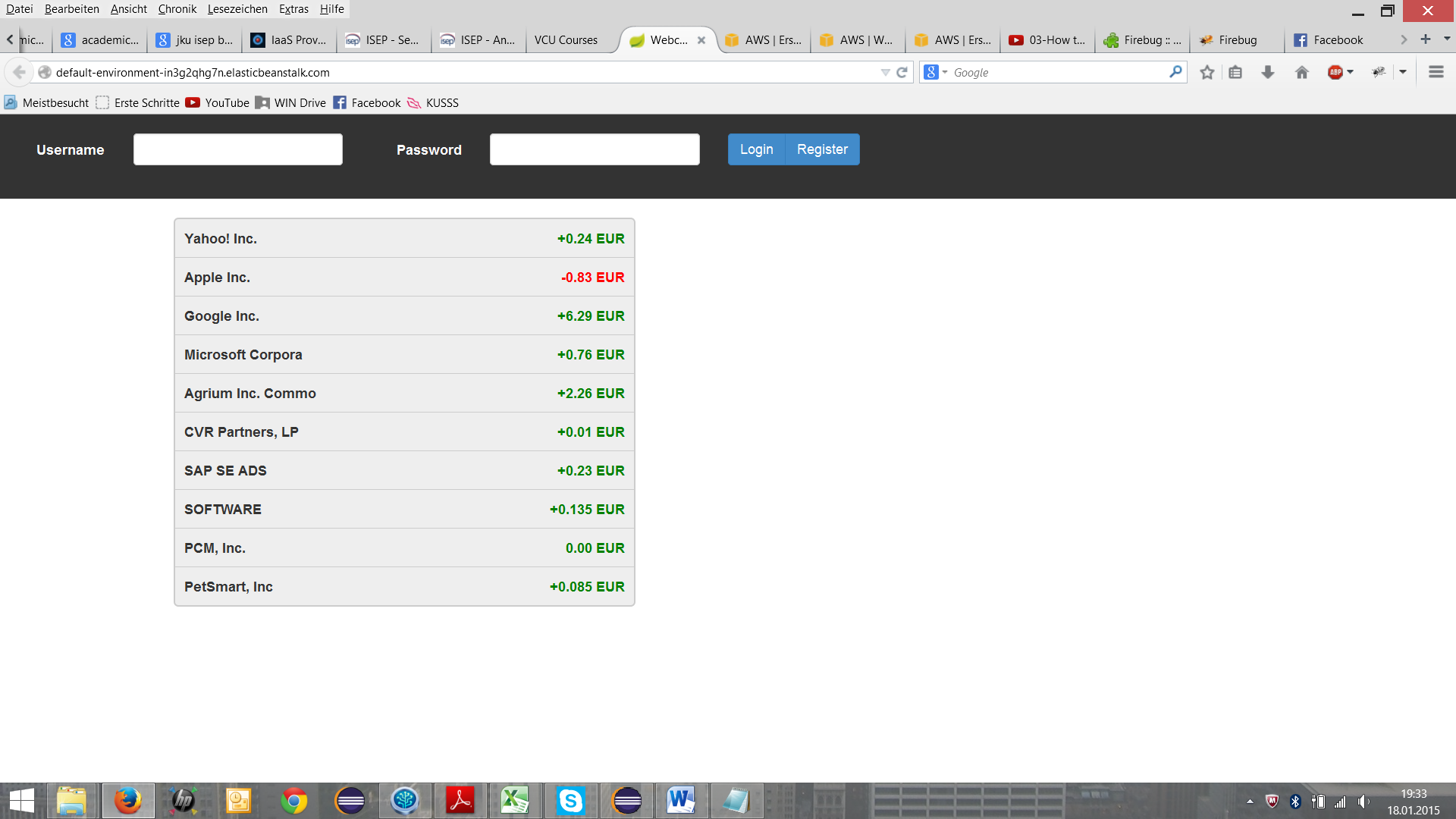


Abbildung 3: Registrierung

## Login

Der Login steht nur registrierten Benutzern zur Verfügung. Er funktioniert analog zur Registrierung. Ein existierender Username muss mit dem dazugehörigen Passwort korrekt eingegeben werden, um zum entsprechenden Account zu gelangen. Nach erfolgreicher Anmeldung können weiterhin die Liste mit den angebotenen Aktien sowie deren Details eingesehen werden. Zusätzlich bietet sich nun jedoch auch die Option, Aktien zu kaufen. Des Weiteren steht dem Benutzer sein persönliches Portfolio aller Transaktionen zur Verfügung, in dem sich seine derzeitigen Aktienbestände mit dem jeweiligen Gesamtwert und der im Besitz befindlichen Stückzahl wiederfinden. Von diesen können auch einige oder alle zum jeweiligen Tagespreis verkauft werden. Sämtliche Transaktionen, also alle Käufe und Verkäufe spiegeln sich in der Transaktionstabelle wider. Zusätzlich stehen dem User auch noch die Möglichkeit der Passwortänderung und des Ausloggens zur Verfügung.

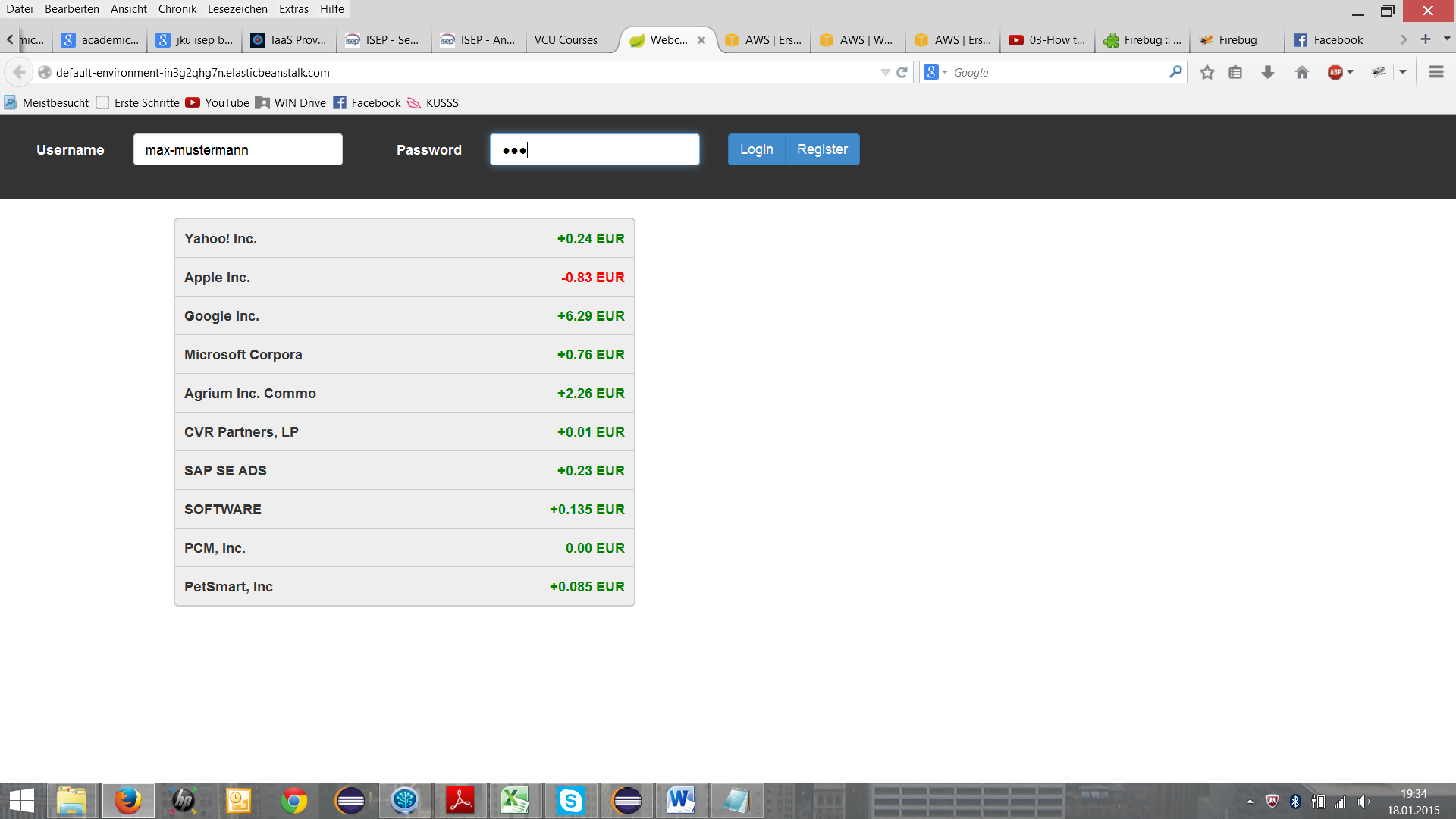


Abbildung 4: Login

## Aktienkauf

Der Aktienkauf steht nur angemeldeten Benutzern zur Verfügung. Es wird eine Liste aller angebotenen Aktien sowie deren Details angezeigt. Zusätzlich bietet sich nun jedoch auch die Option, Aktien zu kaufen. Dafür wird, wie bereits erwähnt, ein Guthaben benötigt. Der Benutzer kann auswählen, wie viele Aktien er kaufen möchte. Dies wird einerseits durch die verfügbare Stückzahl sowie andererseits durch das derzeitige Guthaben beschränkt. Im Portfolio des Benutzers finden sich seine derzeitigen Aktienbestände mit dem jeweiligen Gesamtwert und der im Besitz befindlichen Stückzahl wieder.

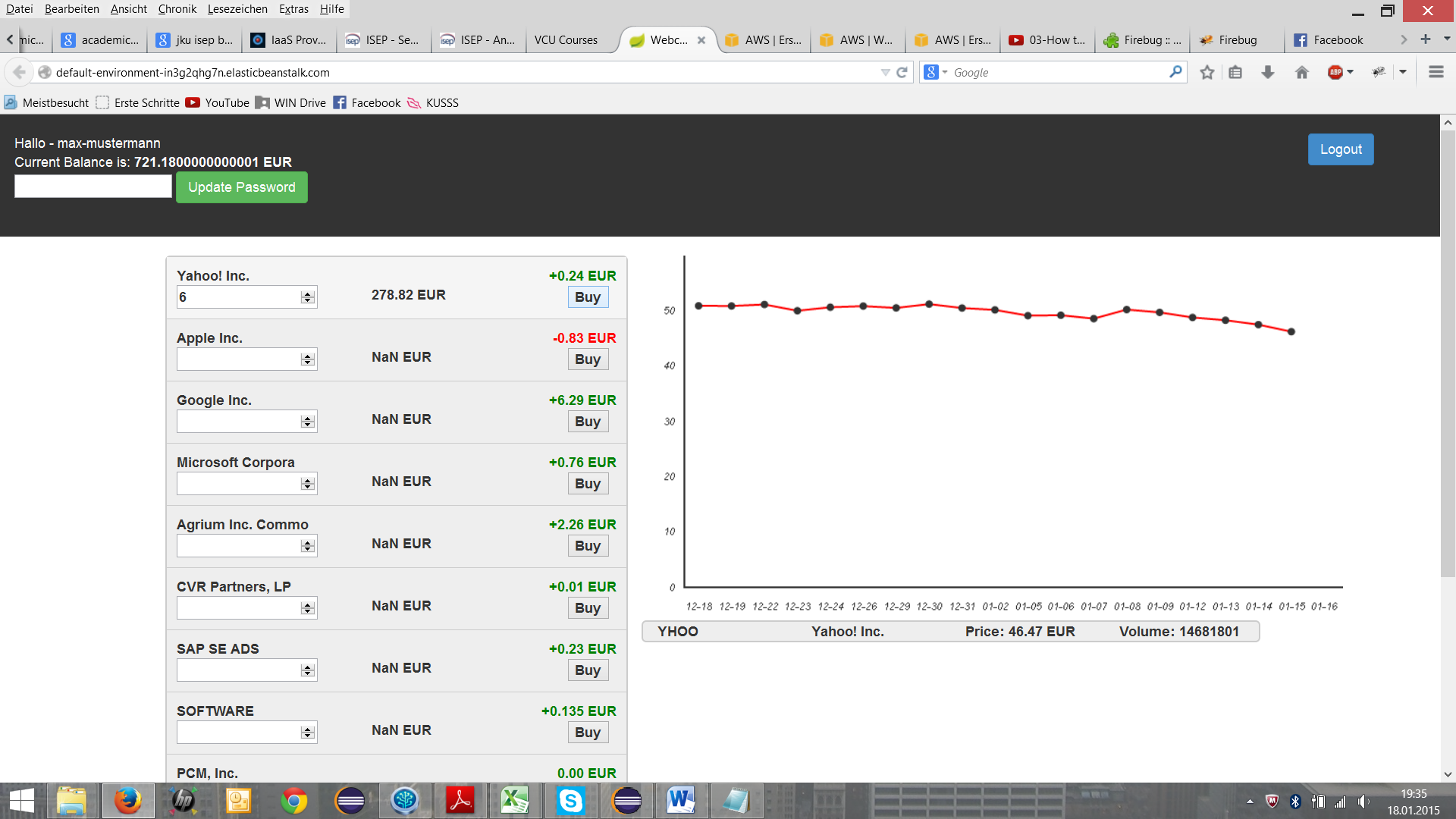


Abbildung 5: Kauf von Aktien

## Aktienverkauf

Der Aktienverkauf steht nur angemeldeten Benutzern zur Verfügung. Es wird eine Liste der vom Benutzer erworbenen Aktien angezeigt. Von dieser können hier wieder einige oder alle zum jeweiligen Tagespreis verkauft werden. Sämtliche Transaktionen, also alle Käufe und Verkäufe spiegeln sich in der Transaktionstabelle wider. Dort sind das Kurzzeichen, der vollständige Namen, die Anzahl der gekauften/verkauften Aktien, der Kauf-/Verkaufspreis und der Typ (Kauf oder Verkauf) ersichtlich.

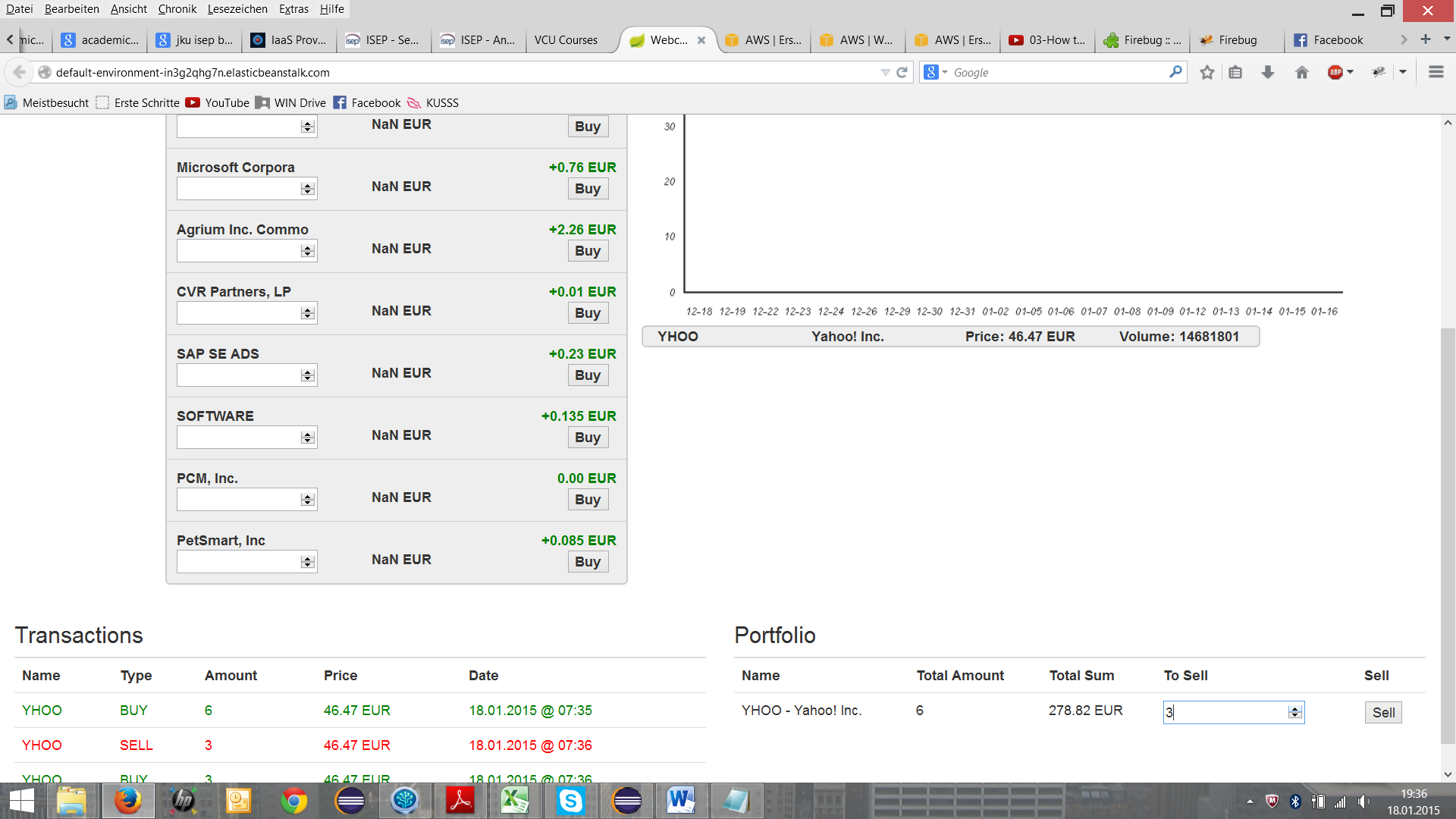
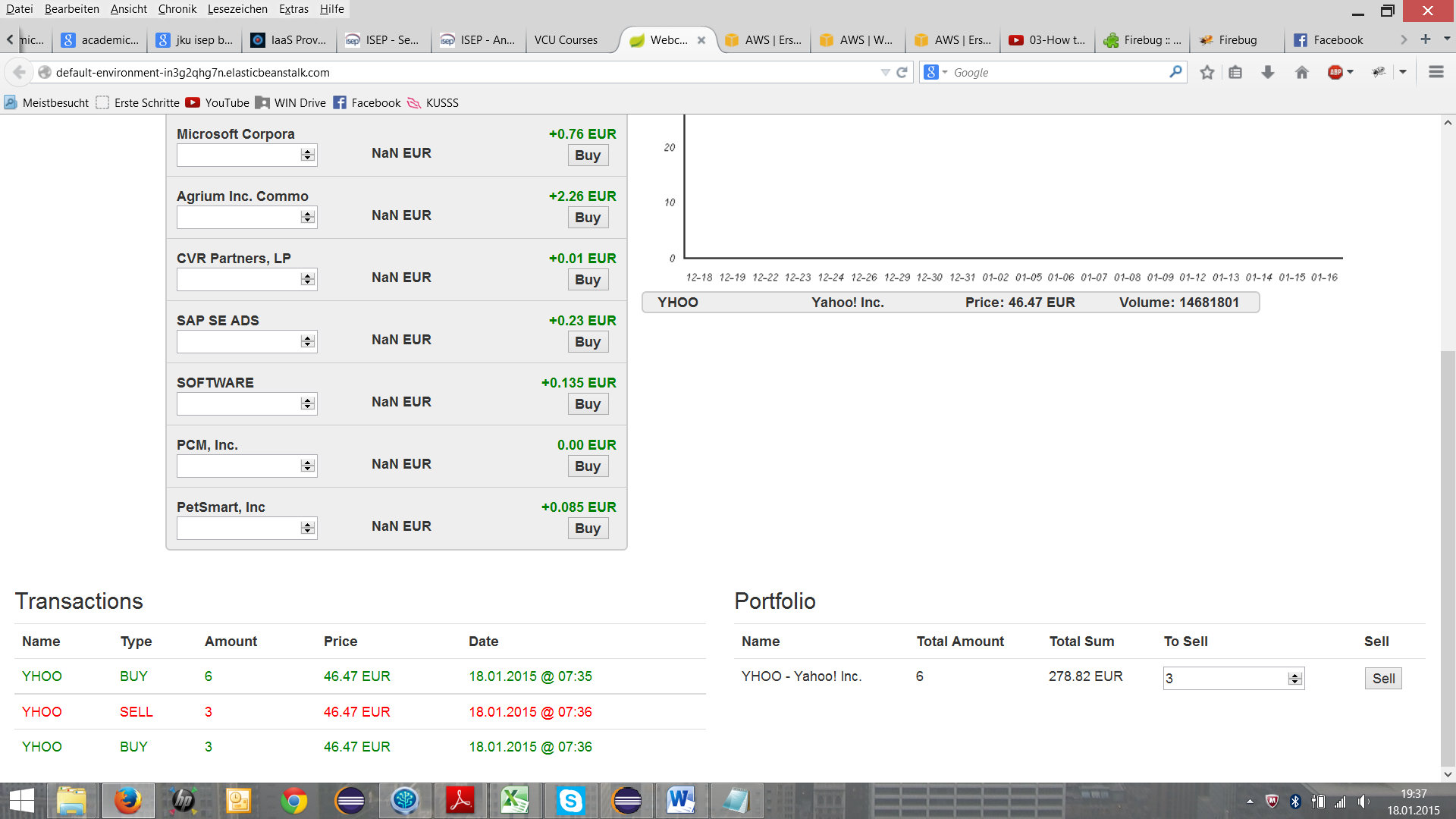


Abbildung 6: Verkauf von Aktien

## Transaktionsübersicht (Portfolio)

Die Transaktionsübersicht steht nur angemeldeten Benutzern zur Verfügung. Hier bekommt der User einen Überblick über alle abgeschlossenen Transaktionen, d.h. über seine Aktienkäufe und Aktienverkäufe. Jede Transaktion beinhaltet das Kürzel, den vollständigen Namen, die Anzahl der gekauften/verkauften Aktien, den Kauf-/Verkaufspreis und der Typ (Kauf oder Verkauf). Die Transaktionen werden nach Aktienname/Kürzel gruppiert und erhalten die gesamte aktuelle Aktienanzahl pro Aktie als Attribut.



## Passwortänderung

Die Passwortänderung steht nur angemeldeten Benutzern zur Verfügung. Hier hat der User die Möglichkeit eine Passwortänderung durchzuführen, die mit dem Speichern-Button abgeschlossen wird. Eine Änderung des Benutzernamens ist jedoch nicht möglich.

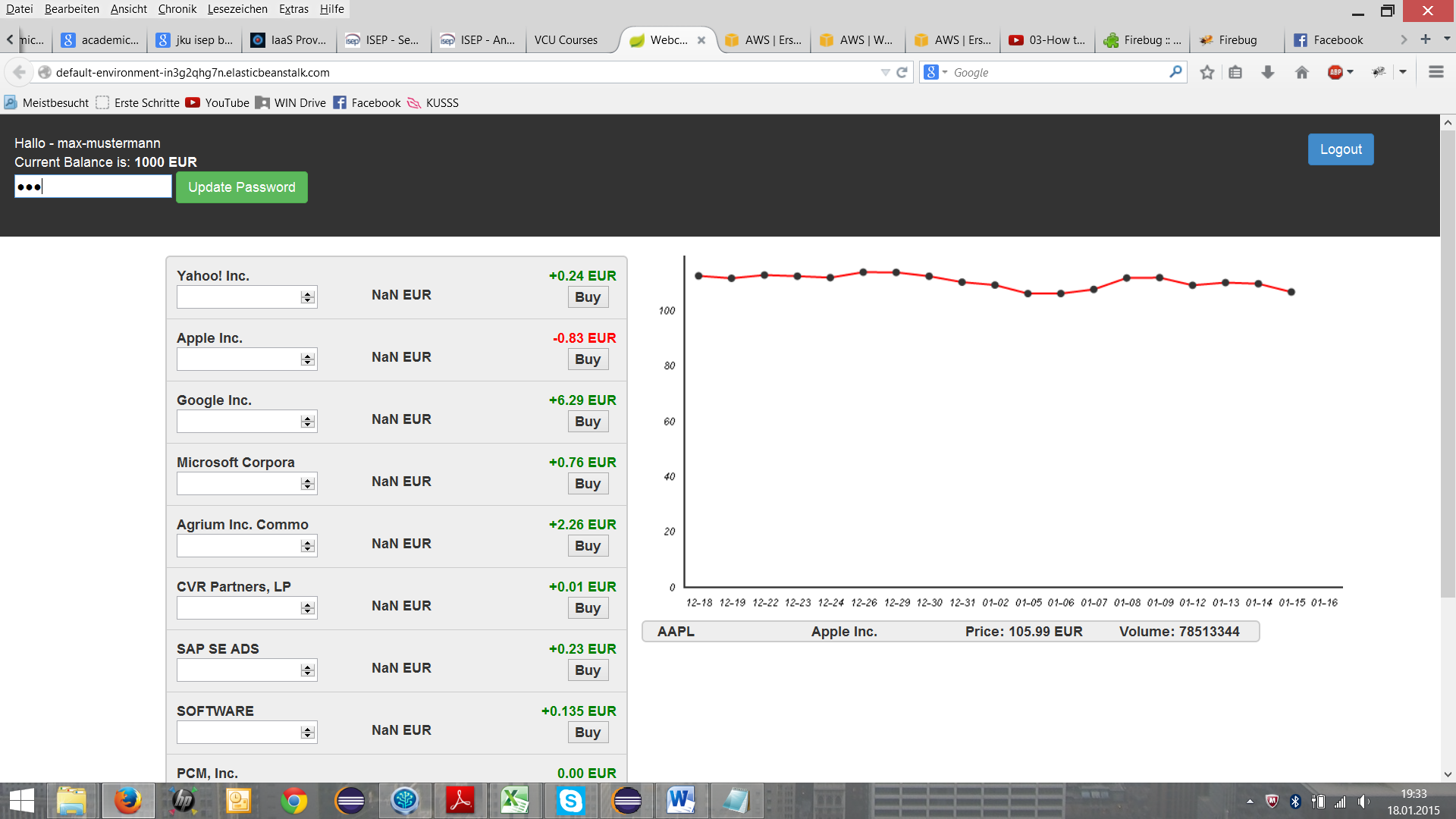


Abbildung 7: Ändern des Passwortes

## Logout

Der Logout steht nur angemeldeten Benutzern zur Verfügung. Um in seine persönlichen Daten wieder einsehen zu können, muss sich ein Benutzer nach erfolgreichem Ausloggen wieder neu einloggen.

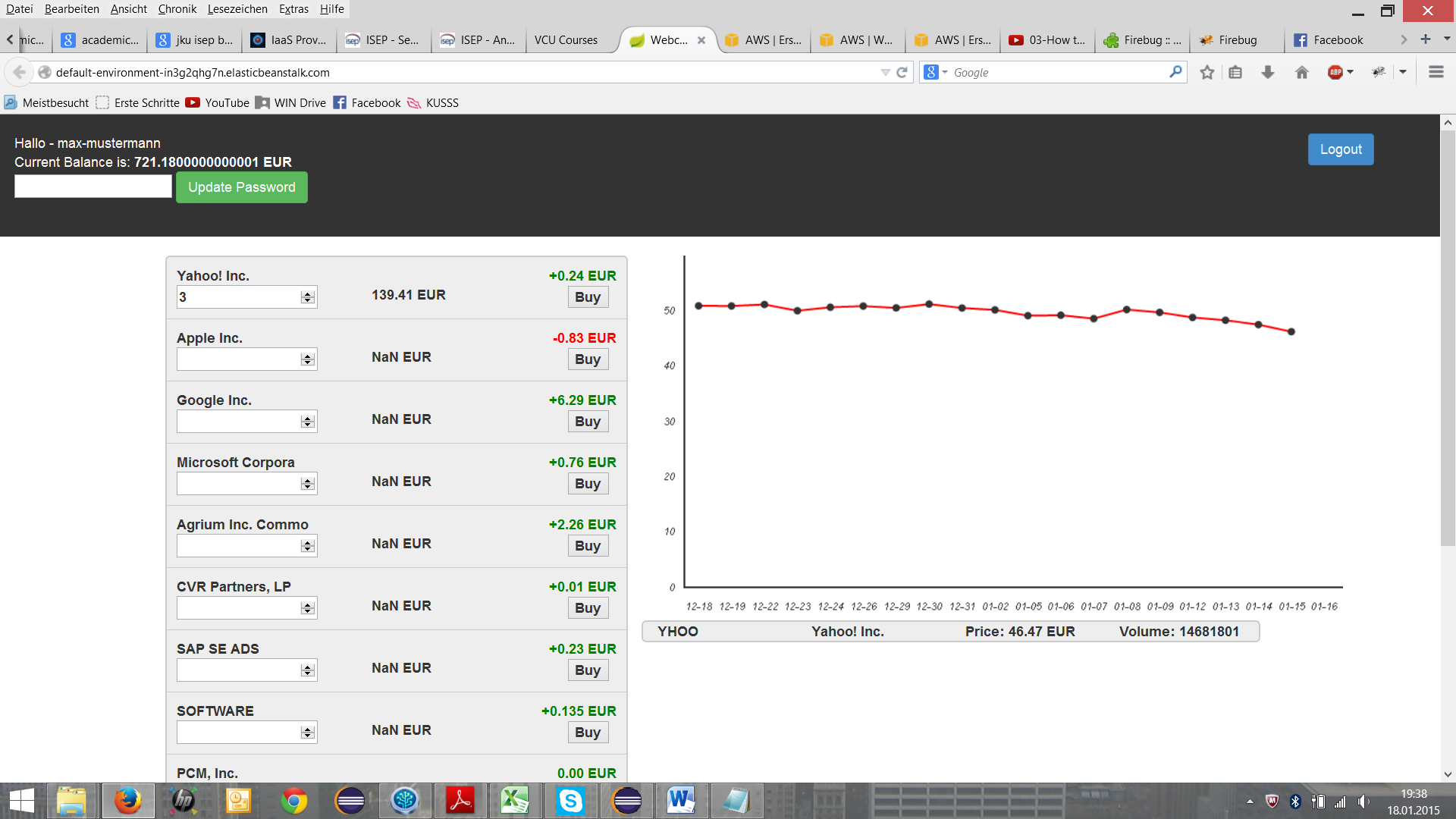


Abbildung 8: Logout

# Fehlersuche

Mithilfe von Firebug, einer Erweiterung für den Webbrowser Mozilla Firefox, haben wir eine stickprobenartige Fehlersuche auf unserer Webseite durchgeführt. Dies hat zu folgenden Ergebnissen geführt:

## Client-Server-Kommunikation

Nachstehend werden Ausschnitte der Requests und Responses bzw. Beispiele gezeigt, da z.B. GET history andere Requests/Responses liefert wenn ein User eingeloggt ist.

* StockOverview (GET stocks):
  + Request:
    - GET /finance/stocks HTTP/1.1
  + Response:
    - [{"symbol":"YHOO","name":"Yahoo! Inc.","volume":14681801,"lastTradedPriceOnly":46.47,"daysHigh":46.59,"daysLow":45.85,"change":"+0.24"},{"symbol":"AAPL","name":"Apple Inc.","volume":78513344,"lastTradedPriceOnly":105.99,"daysHigh":107.58,"daysLow":105.2,"change":"-0.83"}, etc.
* StockHistory (GET history):
  + Request:
    - GET /finance/stocks/YHOO/history HTTP/1.1
  + Response:
    - [{"symbol":"YHOO","date":"2015-01-16","value":46.47},{"symbol":"YHOO","date":"2015-01-15","value":46.23}, etc.
* Register (POST users?un=Alex2&pw=testuser0815):
  + Request:
    - POST /users?un=Alex2&pw=testuser0815 HTTP/1.1
  + Response:
    - Reload the page to get source for: http://default-environment-in3g2qhg7n.elasticbeanstalk.com/users?un=Alex2&pw=testuser0815
* Login:
  + Hier wird nichts geschickt, wird über Cookies geregelt
* Logout:
  + Hier wird nichts geschickt, wird über Cookies geregelt
* updateUser (PUT Alex3?pw=testuser0816):
  + Request:
    - PUT /secured/users/Alex3?pw=testuser0816 HTTP/1.1
    - Authorization: Basic QWxleDM6dGVzdHVzZXIwODE1
    - Cookie: authdata=%22QWxleDM6dGVzdHVzZXIwODE1%22
  + Response:
    - {"transaction":[],"portfolioStock":[],"appUser":{"idAppUser":12,"username":"Alex3","balance":1000.0}}
* BuyStocks (POST transactions?stocksymbol=GOOG&amount=1):
  + Request:
    - POST /secured/finance/transactions?stocksymbol=GOOG&amount=1&type=false HTTP/1.1
    - Authorization: Basic QWxleDM6dGVzdHVzZXIwODE2
    - Cookie: authdata=%22QWxleDM6dGVzdHVzZXIwODE2%22
  + Response:
    - {"transaction":[{"idTransaction":10,"stock":"GOOG","amount":1,"date":1421607500893,"price":508.08,"transactionType":"BUY"}],"portfolioStock":[{"symbol":"GOOG","name":"Google Inc.","amount":1.0,"balance":508.08}],"appUser":{"idAppUser":12,"username":"Alex3","balance":491.92}}
* SellStocks (POST transactions?stocksymbol=GOOG&amount=1):
  + Request:
    - POST /secured/finance/transactions?stocksymbol=GOOG&amount=1&type=true HTTP/1.1
    - Authorization: Basic QWxleDM6dGVzdHVzZXIwODE2
    - Cookie: authdata=%22QWxleDM6dGVzdHVzZXIwODE2%22
  + Response:
    - {"transaction":[{"idTransaction":10,"stock":"GOOG","amount":1,"date":1421607500893,"price":508.08,"transactionType":"BUY"},{"idTransaction":11,"stock":"GOOG","amount":1,"date":1421607830198,"price":508.08,"transactionType":"SELL"}],"portfolioStock":[],"appUser":{"idAppUser":12,"username":"Alex3","balance":1000.0}}
* getUserTransactions (GET transactions):
  + Request:
    - GET /secured/finance/transactions HTTP/1.1
    - Cookie: authdata=%22QWxleDI6dGVzdHVzZXIwMTU%3D%22 (nur nach Login, nicht nach Register)
    - Authorization: Basic QWxleDM6dGVzdHVzZXIwODE1
  + Response:
    - {"transaction":[],"portfolioStock":[],"appUser":{"idAppUser":11,"username":"Alex2","balance":1000.0}}
* addTransaction:
  + besteht aus BuyStocks bzw. SellStocks

## Client-Server-Kommunikation (Error-Checking)

* double registration (POST users?un=Alex2&pw=testuser0815): Error-Code 500 Internal Server Error
  + Request:
    - POST /users?un=Alex2&pw=testuser0815 HTTP/1.1
  + Response:
    - {"timestamp":1421608705197,"status":500,"error":"Internal Server Error","exception":"org.hibernate.exception.ConstraintViolationException","message":"could not execute statement"}
* wrong username: analog zu wrong password
* wrong password (GET transactions): ErrorCode 401 Unauthorized
  + Request:
    - GET /secured/finance/transactions HTTP/1.1
    - Authorization: Basic QWxleDI6YXNkZg==
  + Response:
    - HTTP Status 401 - Unauthorized
* spend more money than available (POST transactions?stocksymbol=GOOG&amount=2): Error-Code 500 Internal Server Error
  + Request:
    - POST /secured/finance/transactions?stocksymbol=GOOG&amount=2&type=false HTTP/1.1
    - Authorization: Basic QWxleDp0ZXN0dXNlcjA4MTU=
    - Cookie: authdata=%22QWxleDp0ZXN0dXNlcjA4MTU%3D%22
  + Response:
    - {"timestamp":1421609034788,"status":500,"error":"Internal Server Error","exception":"service.misc.TransactionError","message":"Not enough Money to buy!"}