Cloud Computing

Google App Engine

Inhaltsverzeichnis

1. Vorwort

1.1. Google App Engine

Dieses Dokument dient als Begleitwerk zum Workshop „**Google App Engine**“. Die Arbeit ist in drei Teile gegliedert, den **Vortrag**, die **Demonstration** und den **Workshop**.

Der Aufbau dieses Dokuments richtet sich nach der Präsentation und dreiteilt ebenso den Inhalt. Hierbei ist das Kapitel zum Vortrag als verdichtete Informationsquelle zu verstehen, welche die Inhalte des Vortrages in der Präsentation tiefer erläutert und miteinander verbindet. Es soll die Präsentation insofern ergänzen, als dass zum Verständnis des Themas die Hintergründe über die Entwicklung und Begründung der Google App Engine fokussiert werden.

Das heißt, dass diese Dokumentation vorrangig das Ziel hat, dem Grundverständnis der strategischen Entwicklung der Google App Engine auf wissenschaftlicherer Basis als die Präsentation zu dienen. Die Präsentation dient hauptsächlich dem Grundverständnis der Google App Engine selbst, während hingegen die Dokumentation hier den Fokus auf Bestandteile legt, die den Rahmen der Präsentation sprengen würden, die zum tieferen Verständnis aber dienlich sind.

Das Kapitel zur Demonstration erklärt den Einstieg in die Arbeitsweise mit der „Google App Engine“, die das Thema dieser Arbeit ist. Es erläutert anhand von Bildern und Auszügen das Basis-Tutorial und dient als Grundvoraussetzung für den Workshop.

Das Kapitel zum Workshop ist zugleich Aufgabenstellung (sprich Übung) und Musterlösung zum Workshop in der Präsentation. Es arbeitet das in der Präsentation gestellte Thema beispielhaft auf und erläutert die Musterlösung zu den Aufgaben zweifelsfrei.

Nach dem Durchlesen dieser Dokumentation und der Präsentationsfolien soll der Leser einen hervorragenden Einblick in die theoretischen und praktischen Grundlagen der Google App Engine haben. Er soll sowohl die Vorteile als auch die Nachteile verstehen und abwägen können sowie sich mit der grundsätzlichen Arbeitsweise beispielhaft auskennen – er soll also über das Thema diskutieren können. Auf Basis der Grundlagen, die in diesem Dokument und der Präsentationsfolien fokussiert werden, soll der Leser selbstständig dazu fähig sein, sich in weiterführende Themen der Google App Engine einarbeiten zu können. Er soll selbst dazu in der Lage sein, Applikationen mit der Google App Engine zu schreiben und für die Öffentlichkeit bereit zu stellen.

2. Google App Engine

2.1. Problemstellung

Betrachtet man Google als Unternehmen, so kann man davon ausgehen, dass Google keine Entscheidung trifft, die nicht fundiert ist. Wie jedes andere Unternehmen ist auch Google daran interessiert, Macht und Geld zu erlangen. Die Entwicklung der Google App Engine ist daher ebenfalls eine Entscheidung von Google, die auf Tatsachen begründet sein muss – andernfalls wäre sie eine Verschwendung von Geld und Ressourcen.

Um die Gründe zu verstehen, die Google bei der Entwicklung der Google App Engine beeinflusst haben, müssen wir zumindest kurz wissen, was die Google App Engine eigentlich ist. Wenn diese Frage zumindest oberflächlich beantwortet ist, können wir auch die Frage klären, wie es dazu kam.

Daher vorneweg kurz die hier notwendige Erklärung, dass es sich bei der Google App Engine quasi um eine moderne Hosting-Plattform[1] handelt. Ferner noch ist es eine spezielle Hosting-Plattform, die für Webapplikationen[2] genutzt werden kann. Solche Webapplikationen lassen sich allerdings schon viel länger auch bei anderen Anbietern bereitstellen.

Um nun also die Unterschiede, Vor- und auch Nachteile der Google App Engine im Vergleich zu diesem bisherigen, klassischen Webhosting[3] zu verstehen, analysieren wir daher im ersten Schritt genau diesen bisherigen Zustand beim Webhosting.

1 einen Ort (im Internet), an dem Dinge (also Daten) abgelegt werden können
2 im Allgemeinen: Internetseiten, auch mit darüber hinausgehenden Funktionen
3 Webhosting: Service von Dienstleistern um Webseiten im Internet zu veröffentlichen

2.1.1 Bisheriger Zustand beim Webhosting

Das hier als klassische Webhosting bezeichnete Webhosting stellt im Allgemeinen einen Vertrag zwischen Kunde und Provider[4] dar. Der Kunde bestellt bei dem Provider seiner Wahl ein Hosting-Paket[5], welches fest definierten Rahmenbedingungen unterliegt.

So erhält der Kunde gegen Bezahlung einen Speicher festgelegter Größe, mit festgelegten Datenvolumina und festgelegter Adresse im Internet. Dieses kann der Kunde nun für seine Webseite bzw. -applikation nutzen und die von ihm gewünschten Angebote dort hinterlegen. Der Provider kümmert sich nun darum, dass die Daten seines Kunden bei möglichst hoher Verfügbarkeit im Rahmen der angebotenen Begrenzungen von der Öffentlichkeit auffindbar ist. Ist das Angebot des Kunden nun über den Rahmenbedingungen des Vertrages, so muss der Kunde entweder ein höherwertiges Paket bezahlen oder der Provider sperrt das Angebot des Kunden, bis die Rahmenbedingungen wieder eintreten – dies tut er im Sinne seiner Wirtschaftlichkeit.

2.1.1.1 Sicht des Betreibers

Der Betreiber[6] muss bei der Erstellung der Webhosting-Paket-Angebote für seine Kunden einige Dinge beachten. Für den Betreiber steht an erster Stelle die Wirtschaftlichkeit seines Angebotes. Somit muss er das richtige Gleichgewicht zwischen Kosten und Gewinn errechnen. Die Kosten stellen hierbei in erster Linie die Anschaffungskosten für Hardware[7], Personalkosten und Betriebskosten dar. Der Gewinn lässt sich insofern verändern, indem der Provider geringere Kapazitäten für höhere Kosten vertreibt.

Kostenpunkte für den Provider:

- Hardware (Server, Netzwerk, etc.)
- Software (Betriebssysteme, Applikations-Server, Management-Systeme)
- Personal (Installation, Instandhaltung, Service)
- Betrieb (Netzwerkgebühren, Strom, z. B. für Server, Kühlung, Gebäude)
- Infrastruktur (Gebäude, externe Netzwerkinstallation)

Der Provider erhält im Gegenzug von seinem Kunden Geld für dessen Nutzung seiner Ressourcen. Da eine Menge Kapital und Expertise in den Aufbau eines Rechenzentrums nötig ist, geht der Provider davon aus, dass die Wenigsten über eine solche Infrastruktur verfügen. Diesen Vorteil nutzt er für die Generierung seiner Paket-Angebote aus. Seine vorhandenen Ressourcen teilt er in kleine Einheiten und bietet diese für seine Kunden an.

Über den Aufbau und die Betreuung des Rechenzentrums gewinnt der Provider eine Menge Erfahrung im Umgang mit Servern, Rechner-Netzwerken und den Interessen seiner Kunden. Er kann sich zuletzt durch seine Erfahrung im Umgang mit dem Bereitstellen von Infrastruktur und Webhosting-Plattform anderen Dienstleistern gegenüber behaupten.

4 ein Dienstleister, der eine Dienstleistung bereitstellt, hier das Webhosting
5 Webhosting-Angebot; besteht aus Speicherplatz, Traffic und Verfügbarkeit
6 ist gleich der Provider
7 Computer, Infrastruktur, alle „harten“ Ressourcen

2.1.1.2 Sicht des Kunden

Der Kunde ist beim klassischen Webhosting darauf angewiesen, ein Webhosting-Paket-Angebot eines Providers zu mieten, um dort seine Webapplikationen zu hosten[8]. Dies kostet den Kunden dann eine gewisse vertraglich vereinbarte Summe, für die er eine gewisse Leistung erhält. So kauft der Kunde zunächst mehr Leistung für einen höheren Preis ein, als er benötigt, um für zukünftige Anforderungen seinerseits bereits ausgerüstet zu sein. Die Leistung erfolgt quasi auf Vorbehalt.

Nutzt der Kunde nun mehr Leistung als er bezahlt, so sperrt der Provider üblicherweise den Zugang aufgrund seiner wirtschaftlichen Interessen. Denn mieten verschiedene Kunden Leistung von dem gleichen Server, so kann ein Kunde die Leistung aller anderen Mitkunden durch zu hohe Auslastung der Leistungsreserven beeinflussen.

Benötigt der Kunde nun viel Leistung, so muss er mehrere Angebote des Providers miteinander kombinieren. Dies kann eine aufwändige Konfiguration nach sich ziehen, z. B. wenn mehrere physikalische Server in ein gemeinsames virtuelles Netz verbunden werden sollen. Der Kunde benötigt bei hohen Leistungsanforderungen daher selbst ein hohes Know-how um die einfachen Paket-Angebote seines Providers in eine für ihn optimale Lösung zu verbinden.

2.1.2 Zentrale Probleme des bisherigen Zustandes

Aus der bisherigen Analyse des Zustandes beim klassischen Webhosting lassen sich mehrere Probleme erkennen. Zentrale Rolle spielt die Frage: Wie können wir das bisherige Modell optimieren?

Optimieren bedeutet, dass der Provider geringere Kosten bei höherem Nutzen umsetzt – also den Gewinn maximiert. Für den Kunden bedeutet es, dass dieser mit geringstem Aufwand maximale Flexibilität erreicht.

Der Provider ist Zeit seines Daseins damit beschäftigt, Infrastruktur auszubauen, Pakete zu konfigurieren und die Kunden zu verwalten. Es bietet sich daher die Aufgabe an, diese Vorgänge soweit irgendwie möglich, zu automatisieren und den Aufwand auf Kundenseite abzuschieben. Weniger Konfigurationsaufwand auf Provider-Seite bedeutet im Umkehrschluss natürlich auch mehr Konfigurationsaufwand auf Kunden-Seite. Allerdings bedeutet es auch mehr Flexibilität, also Konfigurationsmöglichkeiten.

Der Kunde hat die Haupt-Problematik, dass er beim klassischen Webhosting auf zugeschnürte Pakete mit Leistungsgrenzen zurückgreifen muss. Hat sein Geschäft aber einen „Boom“ zu verzeichnen, könnte der Provider im entscheidenden Moment aufgrund der hohen Nutzerlast des Kunden sein Angebot sperren und damit die Geschäfte des Kunden beeinträchtigen. Andererseits würde eine Nicht-Sperrung ebenfalls bedeuten, dass kein Kunde mehr das Angebot nutzen kann, da die bereitgestellte Infrastruktur überlastet ist. Da aber dabei auch noch mehr Kunden des Providers betroffen wären, deren Geschäfte dann ebenfalls nicht zugreifbar wären, ist die einzige Option des Providers, den Kunden mit der Überlast zu beschränken. Für den Kunden ist dies eben fatal – auch deshalb, weil der Kunde meist nur diese eine Instanz der Webapplikation lauffähig hat.

8 hier: im Internet bereitstellen

Das bisherige klassische Webhosting-Modell braucht daher unbedingt eine Überholung der Art und Weise, wie es funktioniert.

Zusammengefasst lässt sich also sagen:

- Unflexible Nutzungsmöglichkeiten
- Schlecht Skalierbare Leistungsreserven
- Hohe Kosten für ungenutzte Ressourcen
- Nur rudimentäre Backup-Lösungen möglich
- Hoher Aufwand des Providers für Kundenbetreuung

2.1.3 Lösungsansatz für bisherigen Zustand

Es muss nun eine Möglichkeit geben, eine Art von Infrastruktur zu betreiben, die so skalierbar ist, dass eine Überlastung durch einzelne Kunden unmöglich ist. Eine Infrastruktur, die für den Provider zu jeder Zeit mit einfachsten Mitteln erweitert werden kann und die jegliche Arten von Auslastung intelligent verwaltet und vermittelt. Für den Kunden muss es eine Plattform geben, die sich automatisch an seine Auslastung angleicht und nur so viel berechnet, wie er auch von der Leistung nutzt. Dabei müssen sämtliche Flexibilitäten auf beiden Seiten maximiert werden, um sowohl Provider als auch Kunden ein maximales Einsatzgebiet der Infrastruktur zu ermöglichen.

Das mag jetzt dem Einen oder Anderen wie bloßes Wunschdenken klingen, aber für den Provider stellt es eine Herausforderung an sein Geschäftsmodell dar und bietet ihm bei Erfolg den Garant für die Marktposition gegenüber seiner Konkurrenten. Man stelle sich nur vor, wie klein und hilflos die Konkurrenz wirkt, wenn es einen Anbieter gibt, der dem Kunden exakt das liefert, was der Kunde wünscht – nicht mehr und nicht weniger.

2.1.3.1 Der Begriff Cloud Computing

In der Tat existiert die Antwort auf die Frage bereits: Cloud Computing.

Der Provider nutzt nun folgenden Trick, um seine Infrastruktur beliebig skalieren zu können. Nicht mehr physikalische Server werden in Einheiten unterteilt und vermietet, sondern die physikalischen Server dienen nur noch als Basis für virtuelle Maschinen, welchen die Art und Weise der darunterliegenden Hardware beinahe völlig egal ist.

Benötigt eine virtuelle Maschine nun mehr Leistung, dann wandert sie provider-intern auf eine physikalische Maschine mit mehr Leistungsreserven. Der Kunde bekommt diesen Vorgang nicht mit, außer der Tatsache, dass er auf Knopfdruck mehr Leistung besitzt. Wird nun noch mehr Leistung benötigt, erstellt der Kunde auf Knopfdruck weitere virtuelle Maschinen auf leistungsfähigeren Servern. Sind die physikalischen Server ausgelastet, stellt der Provider neue Server mit dem gleichen Basissystem unkompliziert nebendran, fügt sie mit einem vorgefertigten Installationsalgorithmus in das System mit ein, und „quasi“ auf Knopfdruck steht mehr Leistung zur Verfügung, auf der wieder bereits konfigurierte virtuelle Maschinen von Kundenseite aus erstellt und ausgeführt werden können.

2.1.3.2 Infrastructure as a Service

Das Spiel lässt sich beliebig weiter führen und wurde von der Gesellschaft als „Infrastructure as a Service“ getauft. Man erkennt, dass die Infrastruktur selbst automatisiert funktioniert und die Auslastung je nach Bedarf hoch- bzw. heruntergefahren werden kann. Auf physikalischen Maschinen wäre das nicht möglich, mit einer bestehenden Applikation mehr Leistung anzufragen... der Server müsste umgebaut werden. Aber die virtuelle Maschine wird einfach mit mehr Leistung konfiguriert und sucht sich automatisch eine bessere physikalische Maschine, falls die Aktuelle für die Konfiguration zu wenig Leistung bietet.

Der Provider hat nun also seine Expertise im Betreiben von Rechenzentren dazu eingesetzt, eine automatische Anpassung der Infrastruktur zu ermöglichen. Er spart sich damit – einmal richtig umgesetzt – die Kosten für teure Hardware-Konfigurationen und kann mit einfachen Mitteln beliebig viele Maschinen „blank“ bereitstellen, um diese vom Kunden dann mit seinen Anforderungen virtuell bestücken zu lassen.

Der Kunde hat nun die freie Wahl und kann seine virtuelle Maschine auf bedeutend mehr Arten konfigurieren, als das mit echter Hardware jemals möglich wäre. Er kann sich die Leistung jederzeit umkonfigurieren, die er aktuell benötigt und zahlt damit nur das, was wiederum seine Kunden bzw. er selbst auch nutzt.

2.1.3.3 Grenzen der Infrastructure as a Service

Für den Kunden bietet Infrastructure as a Service definitiv einen Mehrwert an Flexibilität, Skalierbarkeit und Konfigurationsmöglichkeiten. Problem bleibt allerdings, dass er sich mit virtuellen Maschinen und deren Konfiguration auseinandersetzen muss. Er kann selbstverständlich vorkonfigurierte virtuelle Maschinen, sog. Images seines Providers, nutzen und hat auf Knopfdruck lauffähige und in der Leistung skalierbare Webhosting-Pakete, die somit deutlich stärker erweiterbar sind, als fest zugeschnürte Pakete seines Providers.

Dennoch stellt der Kunde sich die Frage, wie er auch diesen Teil „vergessen“ kann. Die Vorteile, die die virtuellen Maschinen bieten, sind lediglich auf der Ebene der Infrastruktur zu sehen. Für die eigentliche Anwendung – das, worauf es ankommt – hat man bisher nur wenig Nutzen gewonnen, denn diese ist nach wie vor von einer (jetzt) virtuellen Maschine abhängig, die irgendwo ebenfalls Leistungsreserven besitzt. Nicht jede Applikation kann außerdem durch parallel laufende Maschinen verarbeitet werden und müsste umgeschrieben werden, sollten mehr als eine virtuelle Maschine verwendet werden.

Die Frage für den Kunden stellt sich also: Wieso bei Infrastructure as a Service aufhören? Kann man die Automatisation, Skalierbarkeit und Einfachheit nicht noch eine Ebene höher legen und von der Infrastruktur direkt auf die Webhosting Plattform zielen? Was ist mit der Ebene, direkt auf Anwendungen zu zielen?

Der Provider wird also wieder vor eine knifflige Aufgabe gestellt, die er lösen muss, um seine Kunden weiter von seiner Leistung überzeugen zu können. Denn das bedeutet schließlich Umsatz.

2.1.3.4 Platform as a Service

Selbstverständlich hat es unser Provider auf die nächst-höhere Stufe geschafft. Er bietet nicht mehr „nur“ virtuelle Maschinen in ihrer endlosen Skalierbarkeit und Automatisation an – er bietet eine zentrale Plattform an, die als Zugang vor dieses Netzwerk aus virtuellen Maschinen geschaltet ist.

Diese Plattform dient als das, was man beim klassischen Webhosting als Webspace bezeichnet hat – der Speicher, der mit den nötigen Server-Diensten so verknüpft ist, dass bei Abruf aus dem Internet die Applikation, die auf dem Speicher hinterlegt ist, ausgeführt und dargestellt wird.

Nur ist diese zentrale Plattform als Zugangspunkt nicht mehr auf eine physikalische Maschine angewiesen, sondern hat nun die vollständige Flexibilität an virtuellen Maschinen in der Hinterhand.

Das bedeutet, was der Kunde vorher von Hand konfiguriert hat, kann nun ebenfalls das System. Der Kunde entwickelt direkt auf der Plattform und die Plattform stellt sicher, dass die darunterliegende virtuelle Infrastruktur den Leistungsanforderungen genügt. Der Provider ist wieder nur damit beschäftigt, die physikalische Struktur durch blanke Server zu erweitern – einfachste Fließbandarbeit, für die er aber aufgrund des hohen Flexibilitätsnutzens auf Kundenseite eine Menge Geld verlangen kann.

Diese Architektur nennt die Gesellschaft „Platform as a Service“. Die Plattform erkennt hohe Kundenlast und vergrößert den Pool an virtuellen Maschinen, den sie unter sich liegen hat. Sie nutzt verteilte Algorithmen, um die gesamte Rechenkapazität mehrerer Server zu kombinieren. Sie stellt somit nicht nur die Kapazität der Infrastruktur aus ressourcentechnischer Sicht sicher, sondern sie verbindet diese Kapazität auch noch in einer Anwendung. Natürlich macht sie das für viele Anwendungen parallel und nutzt bereitgestellte Kapazität extrem flexibel und passgenau.

Der Kunde muss also nur noch die Applikation für die Plattform entwickeln und nutzt damit die kombinierte Rechenpower des Providers für sich selbst. Bzw. jeder Kunde macht dies. Die virtuelle Infrastruktur wird zu einer Art Wolke für die Plattform. Die physikalische Architektur zu einer Art Wolke für die virtuelle Infrastruktur.

Maximale Flexibilität und maximaler Nutzen für beide Seiten der Verantwortung sind erzielt.

2.1.3.5 Software as a Service

Würde man das Beispiel weiter spinnen, könnte man als Kunde noch auf die Idee kommen, nicht mal mehr konkrete Applikationen für eine Plattform zu entwickeln, sondern nur noch bereits entwickelte Software zu nutzen. Auch für den Provider ist diese Sicht sinnvoll, denn als Anbieter von Webapplikationen, die auf seiner Plattform laufen, kann er zusätzlich Nutzungsgebühren veranschlagen und der Kunde kann ohne Entwicklung bereits Speicherdienste, Nachrichtendienste, Websysteme und viele andere Dinge nutzen – alle mit der Kraft der Platform as a Service im Hintergrund.

2.1.3.6 Google

Solche Software as a Service bietet z. B. das amerikanische Unternehmen Google an. Google ist üblicherweise durch die Suche bekannt. Allerdings findet man dort auch Kartendienste wie Google Maps, Kollaborations-Tools wie die Google Docs, Kommunikationsdienste wie Google+ oder Videoplattformen wie YouTube.

All diese Applikationen nutzen im Hintergrund allerdings die Macht einer Platform as a Service. Denn durch diese Macht sind die Applikationen für Millionen Menschen gleichzeitig ohne Wartezeiten nutzbar. Permanente Verfügbarkeit, ständige Leistungsreserven und höchstflexible Nutzungsoptionen sind ohne Platform as a Service praktisch ausgeschlossen, da diese von Grenzen umzingelt sind. Man überlege sich einmal, wie viele Videos es auf YouTube gebe – ohne Infrastructure as a Service undenkbar zu verwalten. Oder wie viele Kartendienst-Anfragen ständig gestellt werden – ohne Platform as a Service nicht zu kontrollieren.

Also dient die Platform as a Service als der magische Baustein von schier grenzenloser Software wie der Google Dienste. Aufbauend auf der Infrastructure as a Service bietet er den Boden für gigantische Anwendungen, mit denen definitiv Milliarden Menschen erreicht werden können. Das Schöne an Google’s eigener Platform as a Service ist: Jeder kann sie nutzen und damit Software entwickeln!

2.1.3.7 Google App Engine

Der Name der Platform as a Service-Lösung, die Google entwickelt hat, lautet „Google App Engine“. Wir werden uns mit dieser Lösung nun beschäftigen – denn als Kunde können wir die geballte Rechenpower von Google für unsere Zwecke nutzen. Ein Komfort, der keine Alternativen zulässt.

Das besondere an der Google App Engine ist nämlich, dass man Zugriff auf all die Google Dienste bekommt, die Google selbst für seine Applikationen einsetzt.

Warum dies so vorteilhaft gegenüber anderen Anbietern von Platform as a Service ist, ob wir also hier den heiligen Gral der Software-Entwicklung gefunden haben und wieso Google überhaupt uns an seinem Stück vom Kuchen teilhaben lassen will – das sind die großen Fragen, die wir mit diesem Dokument beantworten werden.

2.2. WER

Die Google App Engine scheint eine geniale Konstruktion zu sein. Ohne jetzt schon genau zu wissen, was eigentlich die Google App Engine genau von anderen Platform as a Service-Lösungen unterscheidet, müssen wir uns aber die Frage stellen: Ist jeder Mensch auf Gottes Erden ein Interessent für die Google App Engine oder gibt es nur ein spezielles Klientel, welches dieser Macht würdig ist?

Wir beschäftigen uns daher mit einer Zielgruppen-Analyse und überlegen, wen Google eigentlich mit der Google App Engine ansprechen will, welche Nutzer vielleicht nichts davon haben und was eigentlich die Konkurrenz macht.

2.2.1 Zielgruppe der Google App Engine

Die Zielgruppe der Google App Engine lässt sich relativ schnell feststellen. Die ganze Zeit war davon die Rede, dass diese Platform as a Service-Lösung das klassische Webhosting revolutioniert. Es ist von daher selbstverständlich, dass im Allgemeinen jeder Mensch angesprochen wird, der eine Webapplikation im Internet bereitstellen möchte.

So einfach ist die Zuweisung dann allerdings doch nicht. Schließlich muss man sich die Frage stellen, ob man nicht eine Alternative zur Hand hat, die vielleicht sogar insgesamt besser als die Google App Enigne ist.

Um diese Frage detailliert beantworten zu können, müssten wir das Kapitel WAS vorziehen. Kurzum kann man vorweg nehmen, dass die Google App Engine natürlich nicht kostenlos zur Verfügung gestellt wird, sondern ab einer gewissen Auslastung auch eine Menge Geld kosten kann.

Jeder Nutzer hat pro Tag eine sog. „Free Quota“[9]. Überschreitet er diese Free Quota, wird jedes Bisschen an Nutzung berechnet. Das betrifft sowohl Speicher, wie Traffic und auch E-Mail-Versand.

Damit ist eine Sache klar: Power-User können aufgrund der niedrigen Free Quota-Limits relativ schnell eine hohe Rechnung sehen.[10] Aus dieser Tatsache lässt sich also feststellen, dass kleine Nutzer und frische Startups hervorragend damit beraten sind, ihre kleinen Projekte auf der Google App Engine laufen zu lassen. Mit ein wenig Glück bleibt alles innerhalb der Free Quota. Sollte diese mal überschritten werden, weil das Startup gut läuft und boomt, bleibt alles noch relativ bezahlbar. Schreiten die Nutzerzahlen aber in höchste Dimensionen, sind die Preise der Google App Engine aufgrund ihres fehlenden Mengenrabattes (und nur darauf kommt es bei Großnutzern an) schlichtweg zu teuer im Vergleich zu einem eigenen, klassischen Server-Netzwerk.

Großnutzer sind allerdings nicht nur aufgrund der hohen Kosten bei Google besser darin beraten, selbst die Infrastruktur für ihre eigene Cloud aufzubauen – auch das Thema Datensicherheit könnte eine tragende Rolle spielen. So flexibel und einfach die Google App Engine auf den ersten Blick wirkt: Dort gehostete Applikationen speichern auch alle ihre Daten bei Google direkt – das mag dem ein oder anderen Unternehmer ein Dorn im Auge sein.

Trotzdem bietet die Google App Engine aufgrund ihrer Skalierbarkeit und Macht für den kleinen bis mittleren Anwender eine sehr attraktive Plattform für seine Webapplikationen an. Man bekommt dort quasi ein kleines Stück vom großen Kuchen – und das zum Einstiegspreis von einer kostenlosen Registrierung!

9 diverse Limits, bis zu deren Erreichen die Nutzung kostenlos ist
10 aktuelle Limits und Preise entnehmen Sie bitte hier: https://cloud.google.com/pricing/

2.3. Konkurrenz

Der ein oder andere Kunde mag sich nun jedoch fragen: Wieso sollte ich zu Google gehen und meine Daten diesem Unternehmen anvertrauen, das doch jetzt schon so viele Daten besitzt? Gibt es keine Alternative, mit der ich besser beraten wäre?

Das lässt sich tatsächlich in zwei Sätzen beantworten:

- Nur die Google App Engine bietet die Algorithmen von Google an – und die haben es in sich, denn die haben Google zu der Größe gemacht, die es heute ist!
- Google nutzt nachweislich die gesammelten Daten zur Verbesserung unseres Lebens. Je mehr Daten Google also besitzt, desto mehr kann es uns das Leben erleichtern!

Selbstverständlich kann man das jetzt auch so verstehen, dass Google nur darauf wartet, uns alle zu versklaven und zu erpressen. Aber mal ehrlich: Das klingt nicht sonderlich professionell. Google hat seit jeher gezeigt, dass gesammelte Informationen der Öffentlichkeit zugänglich gemacht wurden. Google Earth/Maps oder Google Suche sind wohl die prominentesten Beispiele.

Schauen wir uns Konkurrenzprodukte wie z. B. „Amazon Web Service“, „Heroku“, „Cloudfoundry“ oder „Microsoft Azure“ an, so haben alle ihre Stärken und Schwächen. Keiner der Konkurrenten kann allerdings mit einer Präsenz und Macht im Internet zu Google konkurrieren. Allein aufgrund der Größe, die Google im Internet darstellt, lässt sich erkennen, wie wenig die Konkurrenz doch zu bieten hat.

Klar ist Amazon ein Name, den man kennt. Auch Microsoft ist durchaus eine große Firma. Aber groß geworden ist Google über das Internet. Menschen suchen und finden über Google alles. Und was könnte besser sein, als seine Webapplikation direkt dort zu hosten, wo Nutzer Antworten finden?

Man tut sich also selbst einen Gefallen, indem man seine Applikation bei Google hostet. Nirgendwo im Internet gibt es so viel Kraft auf einem Fleck, kein Unternehmen wird so oft von Nutzern frequentiert wie Google.

2.4. WARUM

Wir können nun also sehr präzise sagen, dass die Google App Engine Zugang zum größten Internetkonzern der Welt bietet. Aber da ist natürlich noch mehr hinter der Google App Engine.

Die eigentliche Frage, weshalb Google auf die Idee kommt, seine Ressourcen für die Öffentlichkeit zur Verfügung zu stellen, beantwortet sich aus zwei Gründen: Auf der einen Seite wissen wir bereits, dass Google daran interessiert ist, das Leben zu verbessern. Warum? Ganz einfach: Weil dadurch Google als Unternehmen mit seinen Produkten bei JEDEM Menschen einen Kunden hat. Was kann sich ein Unternehmen wünschen, als von allen Menschen auf der Welt gebraucht zu werden? Google wird von allen Menschen auf der Welt gebraucht. Täglich. Stündlich.

Google erreicht dies natürlich durch ein Angebot an Dienstleistungen, welches in jeglichen Lebensbereichen Verwendung findet. Als Beispiel sei das Smartphone-Betriebssystem Android von Google zu nennen.

Die Google App Engine ist nur der nächste logische Schritt in der Entwicklung von Google. So präsent wie Google im Internet ist, so viele Informationen wie es braucht, so wenig Möglichkeiten hatten Nutzer bisher, direkt bei Google Informationen in Form von Webseiten einzustellen. Google hat diese Informationen bisher nur aus dem restlichen Internet gesucht und eben einen Weg gefunden, solchen Informationen sinnvoll Herr zu werden.

Es ist nur sinnvoll, wenn Nutzer direkt bei Google zu dieser Informationsarchitektur beitragen können – und die Google App Engine bietet sowohl Zugang zur Informationsstruktur von Google als auch die Möglichkeit, selbst Informationen dort und überall weiterzugeben.

Um also noch mehr Informationen zu haben und die Welt noch besser zu verstehen, um noch besser ermitteln zu können, woran es in der Welt mangelt, ist die Google App Engine als Plattform eine sehr gute Idee, denn so werden Informationen direkt mit Google verknüpft und tragen somit zum großen ganzen Erlebnis von Google aktiv bei.

Zusätzlich spielen sämtliche Erfahrungen mit ein, die wir bereits in der Problemstellung des klassischen Webhostings gefunden haben. Google weiß, wie ihre Infrastruktur funktioniert. Als einmaliger Nutzenvorteil gegenüber allen anderen Wettbewerbern ist die Google Infrastruktur DAS Alleinstellungsmerkmal der Google App Engine. Wer will denn nicht von diesem großen Kuchen naschen?

Was ist besser, als wenn die Entwickler von Webapplikationen direkt bei Google auf deren App Engine programmieren? Was könnte angenehmer sein, als eine Infrastruktur nutzen zu können, die schier unendlich zu sein scheint?

Google hat mit der Google App Engine also ein Werkzeug geschaffen, welches eine perfekte Symbiose von interner und externer Welt im Internet schafft und so nur beiden Seiten Vorteile bringt, indem deren jeweilige Kompetenzen vereint werden.

Google ist also gut darin beraten, die Google App Engine auch in der Zukunft weiter auszubauen, um noch mehr Entwickler direkt mit ins Boot zu holen. Welch einfachere Möglichkeit gäbe es, Herr über die Informationen im Internet zu werden, wie wenn die Informationen direkt bei einem selbst liegen?

Untermauert wird diese Idealvorstellung von Google's Leitbild: „Tu nichts böses". Und wenn man diesem Satz Glauben schenkt, dann kann man auch nachvollziehen, weshalb Google auch Andere an ihrer Macht teilhaben lässt – nicht nur weil sie es können – sie wollen es.

2.5. WAS

Wir sind nun also in der Position, erkannt zu haben, wie Platform as a Service entstanden ist, wie dafür die Google App Engine entstanden ist, für wen sie entstanden ist und warum sie entstanden ist.

Jetzt können wir uns der eigentlichen Frage stellen: Was ist die Google App Engine jetzt im Besonderen?

Anders formuliert lautet die gleiche Frage: Welche Features und Vorteile bringt mir die Google App Engine als Kunde?

Man kann sich auch kritisch fragen: Bisher klingt ja alles zu gut um wahr zu sein – wo ist an der Google App Engine nun der Haken? Und das ist eine gute und berechtigte Frage!

Gehen wir die einzelnen Argumente für die Google App Engine durch, schauen wir dabei auch darauf, welche negativen Assoziationen damit verbunden werden und wieviel Wahres daran ist.

2.5.1 Hauptkritik

Über alle Vorteile und Argumente für die Google App Engine ist das Haupt-Gegenargument, dass die Daten bei Google hinterlegt werden. Einem Konzern, der so groß und mächtig ist, dass er sich praktisch alles ohne drastische Konsequenzen erlauben kann.

Die Vergangenheit hat gezeigt, dass durchaus auch „Schindluder" mit den Daten getrieben wurde. Als Beispiel sei die „WLAN-Affäre" zu nennen, bei der Google im Rahmen seiner Street-View Straßenansicht auch die WLAN Umgebung gescannt und mitgehört hat.

Andererseits kann man dadurch heute nun jederzeit über sein WLAN seinen Standort erfahren, da Google die WLAN-Netze mit Geodaten kombiniert hat – ein Vorteil, wenn das GPS nicht funktionieren will.

Es geht also um das Grundthema Vertrauen – hat man genug Vertrauen, um Google noch mehr Daten zu geben? Diese Frage muss jeder für sich beantworten. Aber wie auch das obige Beispiel gezeigt hat, nutzt Google die Daten nicht nur aus Eigeninteresse, sondern optimiert bzw. vereinfacht dadurch unser Leben.

Unter diesem Gesichtspunkt – möchte man Google in der Optimierung unseres Lebens helfen oder stellt man sich quer, um seine Daten vor allen Zugriffen zu schützen (was im Falle sicherheitsrelevanter Unternehmensinformationen eine absolut berechtigte Aktion darstellt) – sehen wir uns die Argumente gegen und für die Google App Engine weiter an.

2.5.2 Skalierbarkeit

Wie wir bereits festgestellt haben, läuft die Google App Engine als Platform as a Service-Dienst auf der Infrastruktur von Google selbst – sie ist ein Teil der Google Cloud Platform, einer Summe von verschiedenen Cloud Computing-Lösungen von Google. Diese ist mächtig – man schaue sich nur die Anzahl der Google Rechenzentren und deren Größe auf der Welt an.[11]

Damit ist klar, dass Webapplikationen, die auf der Google App Engine laufen, eine schier unbegrenzte Größe annehmen können. Berechnungen können die geballte Power der Google Infrastruktur nutzen. Hohe Anstürme von Nutzern sind problemlos verkraftbar.

Allerdings hat man als Nutzer standardmäßig nur seine „Free Quota", die einen natürlich nicht die ganze Google Infrastruktur nutzen lässt – sonst könnte das ja jeder zu jeder Zeit. Nein es kostet natürlich Geld, wenn man viele Ressourcen in Anspruch nimmt. Aber da man ein bereits fertiges und Langzeit-getestetes Netzwerk nutzen darf und nicht sein eigenes Netz erst teuer aufbauen muss, ist der Preis auch durchaus gerechtfertigt.

Selbst wenn man sein eigenes Cloud-Netzwerk aufbauen möchte, an die Größe und damit an die mögliche Skalierbarkeit, die auf der Google App Engine geboten wird, kommt man vermutlich selbst nie heran. Ein Punkt für Google also.

2.5.3 Effizienz

Weiterhin können wir uns die Frage nach der Effizienz der Google App Engine stellen, denn eine solch große Infrastruktur, wie die von Google, kostet natürlich eine Menge Energie. Google ist deshalb sehr stolz auf ihre effizienten Rechenzentren und bietet mit diesen eine ökonomische Grundlage für das Hosting von Webapplikationen an. Der PUE[12] der Google Rechenzentren ist ca. 50% geringer als der Durchschnitt üblicher Rechenzentren. Das heißt, dass Google Rechenzentren im Vergleich zu herkömmlichen Rechenzentren über überdurchschnittliche Energieeffizienten verfügen.

Effizienz bedeutet allerdings auch, dass die Software nur dann ausgeführt wird, wenn sie benötigt wird und nur so viele Ressourcen verbraucht, wie für den Betrieb notwendig. Die Google App Engine nutzt Monitoring, um diese Zuweisung von Ressourcen intelligent zu steuern und zu verwalten.

Weiterhin bedeutet Effizienz, dass die Software, die auf der Google App Engine läuft, von den Programmierern effizient geschrieben wird. Aufgrund der verfügbaren Google App Engine-APIs[13] lässt sich dies problemlos bewerkstelligen. Die Software ist demnach extrem effizient geschrieben und nutzt die zugewiesenen Ressourcen optimal aus.

11 siehe: http://www.google.com/about/datacenters/
12 Power Usage Effectiveness (Effektivität der Nutzung von Energie)
13 API = Application Programming Interface (Schnittstelle zur Anwendungsprogrammierung)

2.5.4 Zuverlässigkeit

Eine weitere sehr wichtige Eigenschaft der Google App Engine ist die Zuverlässigkeit. Wir haben bereits die Art und Weise der Funktionalität der Platform as a Service analysiert. Aufgrund der automatischen Verwaltung der Ressourcen einer gehosteten Webapplikation kann die Google App Engine über redundante Datenspeicherung und redundante Laufzeitsysteme die Zuverlässigkeit der Software maximieren.

Das bedeutet, dass Software, die von der Google App Engine z.B. 3-fach redundant abgelegt wurde, praktisch nicht ausfallen kann. Denn wenn eine Instanz der Software nicht mehr funktioniert, wird der Traffic automatisch auf eine der anderen Instanzen umgeleitet. Aus denen wird gleichzeitig wieder eine neue dritte Instanz erstellt, die dann wieder funktioniert.

Sog. Down-Times[14] können mit der Google App Engine quasi nicht auftreten – Ausnahmen bestätigen hier die Regel. Fällt eine zentrale Schnittstelle in der Google Cloud aus, so können kurzfristige Offline-Zeiten auftreten. Bisher ist dies jedoch fast nie vorgekommen.

2.5.5 Sicherheit

Die Google App Engine ist glücklicherweise auch sehr sicher gestaltet. Dies spielt mit dem vorherigen Punkt der Zuverlässigkeit zusammen – denn durch die hohe Sicherheit der Google App Engine kann überhaupt erst eine hohe Zuverlässigkeit entstehen.

Daten werden mehrfach redundant abgespeichert, d.h. Backup-Automatismen sind in der Google App Engine integriert und sorgen so für eine maximale Sicherheit der Daten gegen Zerstörung oder Korruption.

Aber auch der Zugriff auf die Software ist über die Google Authentification Mechanismen sehr sicher gestaltet. Die Applikationen sind an das eigene Google Konto geknüpft und sind somit hinter den dessen Sicherheitsmechanismen gegen unautorisierten Zugriff geschützt.

Zusätzlich zu der Sicherheit der Daten – die bei Google geschützt liegen – gibt es noch eine weitere Sicherheitsstufe in der Ausführung. Diese wird später besprochen.

2.5.6 Übersichtlichkeit

Der nächste Vorteil der Google App Engine ist die hohe Übersichtlichkeit, die sowohl die Konfiguration und Dokumentation wie auch die Verwendung der APIs beinhaltet.

Dank der übersichtlichen Konfigurations-Oberfläche in der Google App Engine Konsole, sind dem Anwender alle Optionen verständlich gegeben, die er für die Nutzung und Erstellung von Instanzen seiner Software benötigt. Er kann ohne lange Einarbeitung bereits die erste Webapplikation hosten und versteht die Stellschrauben, an denen er drehen kann, beinahe instinktiv.

14 Zeit, in der die Webapplikation nicht verfügbar ist

Dank der hervorragenden und übersichtlichen Dokumentation sind Fragestellungen und die Hilfe einfach und direkt erschließbar. Dies beinhaltet sämtliche Probleme mit der Entwicklung der Software und der Konfiguration des SDKs[15].

2.5.7 Einfachheit

Ebenfalls ein großer Vorteil der Google App Engine ist die Einfachheit, mit der Software entwickelt und deployed[16] werden kann. Dank der außergewöhnlich guten Dokumentation ist die Erstellung von Software sowohl unproblematisch als auch einfach.

In unserer Demonstration zeigen wir beispielhaft die Entwicklung einer Applikation und weisen damit auch auf die Einfachheit der Google App Engine-Entwicklung hin.

Gerade bei der Einarbeitung in neue Gebiete der Software-Entwicklung ist eine einfache Einarbeitung von absolutem Vorteil und von jedem Entwickler gerne gesehen.

2.5.8 Alles aus einer Hand

Man könnte diesen Punkt auch als „Abhängigkeit" betrachten – jedoch ermöglicht Google mit seiner Google App Engine einem Entwickler, sich auf diese eine Platform zu konzentrieren. Dank umfangreicher APIs und Features sowie der Erweiterbarkeit durch Third-Party-Applications[17] sind die Grenzen in der Entwicklung von Webapplikationen auf der Google App Engine sehr weit gedehnt.

Allerdings ist durch die „Abhängigkeit" der Google App Engine garantiert, dass die Software nicht durch Konflikte mit externen Anbietern lahmgelegt wird. Was auf der Google App Engine entwickelt wurde, läuft dort auch. Dafür kann Google aufgrund dieses Vorteiles Sorge tragen – denn sie haben alle Teile, die für die Applikationen nötig sind, unter einer Kontrolle und können so alles perfekt miteinander vernetzen.

2.5.9 Sandbox

Ein anderer Aspekt der Sicherheit bei der Google App Engine ist die Sandbox, in der jede Software standardmäßig läuft. Aufgrund dieser Sandbox ist die Software durch Zugriffe von anderer Software, die ebenfalls bei der Google App Engine läuft, geschützt. Keine andere Software kann der eigenen Software schaden – ein sehr wichtiger Punkt im Umgang mit der Entwicklung und dem Hosten von Webapplikationen und deshalb gesondert aufgeführt.

Es ist also ausgeschlossen, dass ein anderer Entwickler eine Software schreibt, welche die Daten unserer Software kompromittieren kann. Unter diesem Aspekt ist die Entwicklung von Software auf der Google App Engine sicherer denn je.

15 SDK = Software Development Kit (Sammlung von Entwicklungswerkzeugen)
16 veröffentlicht
17 Software von Drittanbietern, also nicht Google und nicht uns selbst

2.6. WIE

Wir kennen nun die großen Vorteile der Google App Engine und können nun tiefer in die Materie einsteigen und uns mit der Entwicklung unserer ersten Webapplikation auf der Google App Engine auseinandersetzen.

Um dies im nachfolgenden Kapitel – der Demonstration – problemlos bewerkstelligen zu können, schauen wir uns vorher noch ein paar Details der Google App Engine an, wie diese intern funktioniert, damit wir dann das Grundverständnis dafür haben, Software auf Basis der Google App Engine entwickeln zu können.

2.6.1 Features der Google App Engine

Bevor wir nun ernsthaft Software entwickeln, schauen wir uns kurz die zentralen Features der Google App Engine an.

- **Memcache**
  Ermöglicht In-Memory Calculations, also Berechnungen direkt im Hauptspeicher.
- **Task Querys**
  Ermöglicht die Verwaltung von Aufgaben-Warteschlangen
- **Blobstore**
  Ermöglicht die Datenspeicherung von Objekten
- **Bilderdienst**
  Dient zur Bild-Berechnung und -Verarbeitung
- **E-Mail**
  Dient zum Versenden von Mails mit Google
- **Multi-Instances**
  Ermöglicht grenzenlose Skalierbarkeit dank paralleler Instanzen
- **Authentification**
  Ermöglicht Nutzern die Anmeldung mit einem Google Konto
- **Third-Party Frameworks**
  Lässt die Erweiterbarkeit durch Drittanbieter-Software zu, z.B. Spring, Wordpress, etc.

Es gibt also in der Tat einige sehr interessante Features, mit denen über 90% der Webapplikationen problemlos gebaut werden können. Die restlichen 10% lassen sich dank Drittanbieter-Software und eigener APIs dann ebenfalls verwirklichen.

2.6.2 Entwicklung mit und für die Google App Engine

Prinzipiell ist die Funktionsweise bereits bekannt – wir spielen sie dennoch an einer kurzen Liste durch, damit wir erkennen, wie einfach die Entwicklung von Software für die Google App Engine doch ist.

1. Bei der **Google App Engine** anmelden
2. **SDK** herunterladen und installieren
3. Erste Software nach unserer Demonstration oder dem Google Tutorial https://developers.google.com/appengine/docs/java/gettingstarted/introduction entwickeln
4. Software lokal testen
5. Eine „**Application**“ bei Google erstellen, unter der die Software identifiziert werden soll
6. Die Software bei Google direkt deployen
7. Google führt die Software in einer Sandbox aus
8. Die Software ist im Internet direkt erreichbar

Man sieht also, dass die Entwicklung von Software sehr unproblematisch ist und die Erstellung einer Applikation „Straight-Forward“ erfolgen kann.

Entwickelt werden kann mit vier verschiedenen Programmiersprachen, die da wären:

- Java 6
- Python 2.6
- PHP 5.4
- Google Go 1

2.6.3 Abschluss

Wir haben nun den Punkt erreicht, in dem wir die Theorie der Google App Engine verstehen und nachvollziehen können. Im nächsten Kapitel erfolgt deshalb nun direkt die Entwicklung einer Software mit dem SDK der Google App Engine. Danach sind wir bereit, die Google App Engine für uns und unser Unternehmen gewinnbringend einzusetzen.

3. Demonstration

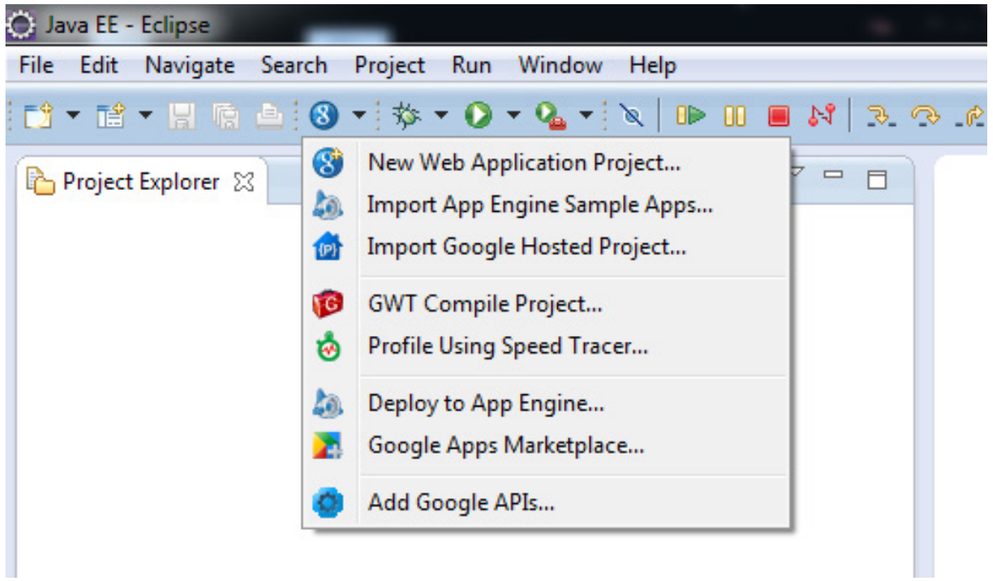
3.1. Überblick über die Projektstruktur

```
Guestbook/
    src/
        ...Java source code...
    META-INF/
        ...other configuration...
    war/
        ...JSPs, images, data files...
    WEB-INF/
        ...app configuration...
    lib/
        ...JARs for libraries...
    classes/
        ...compiled classes...
```

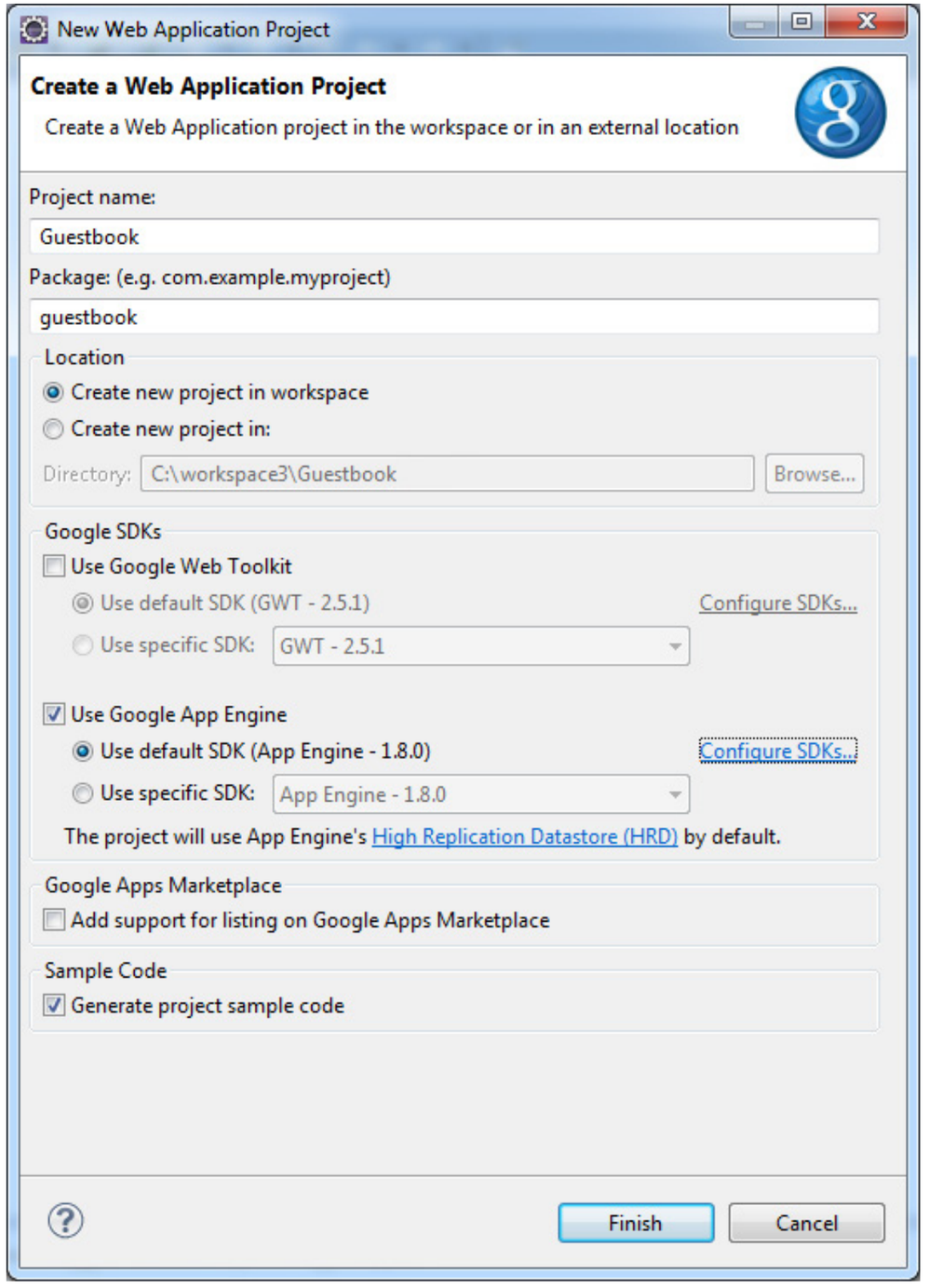
3.2. Umsetzung

3.2.1 Projekt anlegen

1. Neues Projekt anlegen (New Web Application Project) oder Google Button



2. Project name: **Guestbook**; Package: **guestbook**
3. Haken bei „Use Google Web Toolkit“ entfernen
4. Haken bei „Use Google App Engine“ setzen



3.2.2 Klassen-Übersicht

3.2.2.1 Die Servlet-Klasse

Die Java Anwendungen in der Google App Engine benutzen die Java Servlet API.

Die Java Servlet API ermöglicht sog. Servlets. Ein Servlet ist dabei ein kleines Java Programm, das innerhalb eines Web-Servers läuft. Servlets empfangen und antworten auf Fragen von Web Clients, hauptsächlich über HTTP (HyperText Transfer Protocol). Eine Java Klasse beinhaltet (`extends`) entweder die `javax.servlet.GenericServlet` oder die `javax.servlet.http.HttpServlet` Klasse.

Als Einstieg wird die erste Anwendung ein Hello World Programm sein:

1. Im Ordner /src/guestbook die Datei GuestbookServlet.java anlegen
2. Folgendes eintragen (meistens wird dieser Quellcode automatisch erzeugt):

```
package guestbook;

import java.io.IOException;
import javax.servlet.http.*;

public class GuestbookServlet extends HttpServlet
{
    public void doGet(HttpServletRequest req, HttpServletResponse resp)
                      throws IOException
    {
        resp.setContentType("text/plain");
        resp.getWriter().println("Hello, world");
    }
}
```

Um unsere Klasse `GuestbookServlet` von der Klasse `HttpServlet` erben zu lassen, muss man zuerst das `javax.servlet.http.*;` importieren. Des Weiteren benötigt man zur Absicherung die `java.io.IOException;` welche auch importiert wird.

Anschließend lässt man die Klasse `GuestbookServlet` von der Klasse `HttpServlet` erben und benutzt die im `HttpServlet` vorgefertigte Methode `doGet(HttpServletRequest req, HttpServletResponse resp)`, die zusätzlich eine `IOException` wirft.

Damit die Nachricht auch angezeigt werden kann, muss man zuerst einen `Content-Type` (Inhalts-Typ) festlegen; in diesem Fall soll ein einfacher Text angezeigt werden (Plain) `resp.setContentType("text/plain");`. Anschließend kann man die `getWriter` Methode aufrufen, um die Nachricht auszugeben `resp.getWriter().println("Hello, world");`.

3.2.2.2 Die Datei web.xml

Hier muss man nun einstellen, welche Servlet Klasse verwendet werden soll, wenn der Web Server eine Anfrage (Request) erhält:

1. Die Datei war/Web-Inf/web.xml öffnen
2. Folgendes eintragen:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE web-app PUBLIC
    "-//Oracle Corporation//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app xmlns="http://java.sun.com/xml/ns/javaee" version="2.5">
    <servlet>
        <servlet-name>guestbook</servlet-name>
        <servlet-class>guestbook.GuestbookServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>guestbook</servlet-name>
        <url-pattern>/guestbook</url-pattern>
    </servlet-mapping>
    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
    </welcome-file-list>
</web-app>
```

In der web.xml Datei wird das Servlet `guestbook` deklariert und auf die URL /guestbook verwiesen (mapping).

Wenn der Benutzer einen Pfad angibt, der mit keinem Servlet verbunden ist, wird die Datei index.html aufgerufen.

3.2.2.3 Die Datei appengine-web.xml

Die wichtigste Datei ist die appengine-web.xml. Sie ist eine Konfigurationsdatei und wird zum Deployen und Ausführen der Anwendung benötigt:

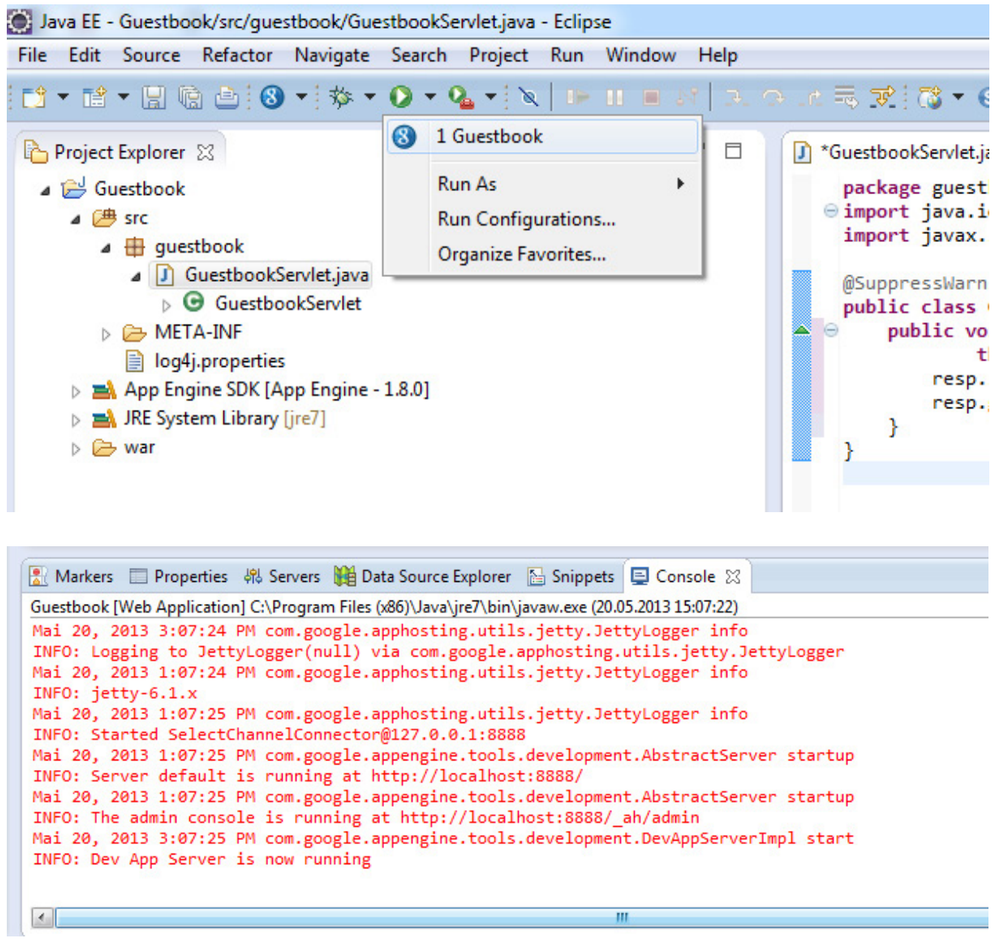
1. Die Datei war/Web-Inf/appengine-web.xml öffnen
2. Folgendes eintragen:

```
<?xml version="1.0" encoding="utf-8"?>
<appengine-web-app xmlns="http://appengine.google.com/ns/1.0">
    <application></application>
    <version>1</version>
</appengine-web-app>
```

Diese beinhaltet die `ID` für die Anwendung, die Versionsnummer, static files (z.B. CSS Dateien und Bilder) und resource files (z.B. JSPs). Alle Informationen zur appengine-web.xml Datei können hier nachgelesen werden: https://developers.google.com/appengine/docs/java/config/appconfig.

3.2.3 Starten des Projekts

Nun wird die Datei ausgeführt (Run => Guestbook) und gewartet bis in der Konsole **INFO: Dev App Server is now running** steht



Nun kann man im Browser unter http://localhost:8888/guestbook seine Anwendung aufrufen.

3.2.4 User-Service

Wird benutzt, um den Google User Account Service in eine Anwendung zu integrieren. So können sich die Benutzer mit ihrem Google Account in einer Anwendung anmelden. Dies ist für ein Gästebuch sinnvoll, da man den User-Namen zu jedem Gästebuch-Eintrag identifizieren kann.

Nun wird der User Service in das bereits bestehende Projekt **Guestbook** integriert, um eine persönliche Begrüßung auszugeben.

1. Datei src/guestbook/GuestbookServlet.java öffnen
2. Die Datei folgendermaßen erweitern:

```
package guestbook;

import java.io.IOException;
import javax.servlet.http.*;
import com.google.appengine.api.users.User;
import com.google.appengine.api.users.UserService;
import com.google.appengine.api.users.UserServiceFactory;

public class GuestbookServlet extends HttpServlet
{
    public void doGet(HttpServletRequest req, HttpServletResponse resp)
                      throws IOException
    {
        UserService userService = UserServiceFactory.getUserService();
        User user = userService.getCurrentUser();

        if (user != null)
        {
            resp.setContentType("text/plain");
            resp.getWriter().println("Hello, " + user.getNickname());
        }
    else
    {
        resp.sendRedirect(userService.createLoginURL(
                          req.getRequestURI()));
    }
}
```


3. Damit man mit Usern arbeiten kann, muss man zuerst drei Klassen, die aufeinander aufbauen, importieren:

   3.1. `UserServiceFactory`: Dient zum Erstellen und Zugreifen auf einen `UserService`.
   https://developers.google.com/appengine/docs/java/javadoc/com/google/appengine/api/users/UserServiceFactory

   3.2. `UserService`: Dient zum An- und Abmelden von Usern und zum Abfragen von Informationen über diese.
   https://developers.google.com/appengine/docs/java/javadoc/com/google/appengine/api/users/UserService

   3.3. `User`: Enthält Informationen über den User

   3.4. Anschließend legt man einen `UserService userService` an und kann durch diesen nun auf den aktuellen User zugreifen `userService.getCurrentUser();`

   3.5. Der aktuelle `User` wird nun in einem `User` Objekt `user` gespeichert

   3.6. Nun wird die Ausgabe des Textes noch so bearbeitet, dass, im Falle der `User` ist angemeldet (`user != null`), der Name des Users mit ausgegeben wird `resp.getWriter().println("Hello, " + user.getNickname());`

   3.7. Ist man bereits eingeloggt, würde sofort der Begrüßungstext angezeigt werden. Sollte man den Development Server neu gestartet haben, wodurch man logischerweise wieder ausgeloggt wird, wird man zur Login-Seite umgeleitet (`userService.createLoginURL(...)`). Vom Anmeldebildschirm wird man nun wieder zur App-Seite umgeleitet, da sich dieser die ursprüngliche URL gemerkt hat (`req.getRequestURI()`).

Nun kann man unter http://localhost:8888/ die Seite neu aufrufen:

1. Hat man den Server nicht neu gestartet, wird sofort die Begrüßung ausgegeben, da man noch eingeloggt ist. Andernfalls kommt man nun zu einem Login-Bildschirm. Hier muss man irgendeine Email-Adresse angeben, z.B. **hans@wurst.de**, und sich einloggen.

   Anmerkung: Würde man die App auf die Google App Engine laden, würde nicht dieser Anmeldebildschirm, sondern der Google Anmeldebildschirm angezeigt werden. Bei diesem muss man sich mit einer gültigen Googlemail Adresse anmelden.

2. Anschließend wird man von der Anwendung mit der eingegebenen Email-Adresse begrüßt.

3.2.5 JSP (Java Server Pages)

Damit der Nutzer nun auch einen Eintrag ins Gästebuch schreiben kann, wird nun ein User-Interface mit JSP implementiert, da dieses leichter zu warten ist. Theoretisch könnte man den HTML-Code auch direkt in das Java-Servlet schreiben, wodurch aber die Wartbarkeit leidet. Deshalb sollte man für das User-Interface ein Template verwenden.

Die Google App Engine kompiliert die JSP-Datei automatisch in den WAR Ordner als eine JAR Datei und verknüpft (mapped) den URL-Pfad.

Nun soll die bisherige Seite in das JSP geschrieben und von diesem umgesetzt werden:

1. Im Ordner war/ eine neue Datei guestbook.jsp anlegen
2. Diese soll folgenden Inhalt bekommen:

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ page import="java.util.List" %>
<%@ page import="com.google.appengine.api.users.User" %>
<%@ page import="com.google.appengine.api.users.UserService" %>
<%@ page import="com.google.appengine.api.users.UserServiceFactory" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>

<html>
    <body>

<%
    UserService userService = UserServiceFactory.getUserService();
    User user = userService.getCurrentUser();
    if (user != null) {
        pageContext.setAttribute("user", user);
%>
<p>Hello, ${fn:escapeXml(user.nickname)}! (You can
<a href="<%= userService.createLogoutURL(request.getRequestURI()) %>"
>sign out</a>.)</p>
<%
    } else {
%>
<p>Hello!
<a href="<%= userService.createLoginURL(request.getRequestURI()) %>"
>Sign in</a> to include your name with greetings you post.</p>
<%
    }
%>

    </body>
</html>
```


Damit man wieder mit den Usern arbeiten kann benötigt man die drei Klassen `UserServiceFactory`, `UserService` und `User`.

Für eingeloggte `User` soll ein Logout und für nicht eingeloggte `User` ein Login zur Verfügung stehen: `userService.createLogoutURL(request.getRequestURI()` und `userService.createLoginURL(request.getRequestURI())`.

Für mehr Informationen zur Syntax von JSP ist folgender Link hilfreich: http://www.jsptutorial.org/content/syntax.

Nun soll diese Datei beim Aufruf der Anwendung gestartet werden. Dies wird wie folgt umgesetzt:

1. Die Datei war/WEB-INF/web.xml öffnen
2. `<welcome-file>index.html` ändern

```
<welcome-file-list>
    <welcome-file>guestbook.jsp</welcome-file>
</welcome-file-list>
```

Startet man nun den Server neu, wird nach dem einloggen direkt das JSP aufgerufen.

Jetzt soll die Anwendung auch eine Oberfläche bekommen, damit der Benutzer seinen Eintrag verfassen kann. Das HTML-Form soll nun in das JSP geschrieben werden und zum Verarbeiten der Daten wird eine neue Servlet-Klasse, `SignGuestbookServlet`, verwendet.

Für den Anfang soll das Servlet zuerst getestet werden: Wird vom Benutzer ein Gästebuch-Eintrag eingegeben, wird dieser an das `SignGuestbookServlet` geschickt, dort verarbeitet und in der Konsole ausgegeben. Der Benutzer wird anschließend wieder zur guestbook.jsp umgeleitet.

1. Die Datei guestbook.jsp öffnen
2. Die Datei folgendermaßen erweitern:

```
...
<form action="/sign" method="post">
    <div><textarea name="content" rows="3" cols="60"></textarea></div>
    <div><input type="submit" value="Post Greeting" /></div>
</form>

    </body>
</html>
```

Anmerkung: Das Form soll oberhalb des schließenden body-Tags eingefügt werden.

Mit dem Form-Element wird hier nun eine Textarea und ein Button erstellt, um die Nachricht einzugeben und abzusenden.

Nun soll das `SignGuestbookServlet` umgesetzt werden.

1. Im Ordner src/guestbook eine Datei SignGuestbookServlet.java anlegen
2. Dieses soll folgenden Inhalt bekommen:

```
package guestbook;

import java.io.IOException;
import java.util.logging.Logger;
import javax.servlet.http.*;
import com.google.appengine.api.users.User;
import com.google.appengine.api.users.UserService;
import com.google.appengine.api.users.UserServiceFactory;

public class SignGuestbookServlet extends HttpServlet
{
    private static final Logger log =
        Logger.getLogger(SignGuestbookServlet.class.getName());

    public void doPost(HttpServletRequest req,
                       HttpServletResponse resp) throws IOException
    {
        UserService userService = UserServiceFactory.getUserService();
        User user = userService.getCurrentUser();

        String content = req.getParameter("content");
        if (content == null)
        {
            content = "(No greeting)";
        }
        if (user != null)
        {
            log.info("Greeting posted by user " +
                user.getNickname() + ": " + content);
        }
        else
        {
            log.info("Greeting posted anonymously: " + content);
        }

        resp.sendRedirect("/guestbook.jsp");
    }
}
```


Der `Logger` dient dazu, die vom Benutzer gesendete Nachricht auf der Konsole auszugeben. Wenn der Benutzer eingeloggt ist, wird sein Nutzername und seine Nachricht, ansonsten anonymously und die Nachricht ausgegeben.

Das Ganze geschieht in der Methode `public void doPost(HttpServletRequest req, HttpServletResponse resp)`. Sendet der Nutzer seine Nachricht wird genau diese Methode aufgerufen, um die Nachricht mit dem Nutzernamen zu verarbeiten.

Anschließend muss man die web.xml Datei bearbeiten, um das `SignGuestbookServlet` zu deklarieren und dieses anschließend mit der /sign URL zu verknüpfen.

1. Die Datei war/Web-INF/web.xml öffnen
2. Folgendermaßen erweitern:

```
<web-app xmlns="http://java.sun.com/xml/ns/javaee" version="2.5">

    ...

    <servlet>
        <servlet-name>sign</servlet-name>
        <servlet-class>guestbook.SignGuestbookServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>sign</servlet-name>
        <url-pattern>/sign</url-pattern>
    </servlet-mapping>

    ...

</web-app>
```

web.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" version="2.5">

    <servlet>
        <servlet-name>Guestbook</servlet-name>
        <servlet-class>guestbook.GuestbookServlet</servlet-class>
    </servlet>
    <servlet>
        <servlet-name>sign</servlet-name>
        <servlet-class>guestbook.SignGuestbookServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>sign</servlet-name>
        <url-pattern>/sign</url-pattern>
    </servlet-mapping>

    <servlet-mapping>
        <servlet-name>Guestbook</servlet-name>
        <url-pattern>/guestbook</url-pattern>
    </servlet-mapping>

    <welcome-file-list>
        <welcome-file>guestbook.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

Nun muss man noch eine Datei bearbeiten, um die Nachrichten in der Konsole anzeigen lassen zu können.

1. Datei war/WEB-INF/logging.properties öffnen
2. Folgendermaßen erweitern:

```
.level = WARNING
guestbook.level = INFO

...
```


Anmerkung: Das Servlet stellt Nachrichten mit dem **INFO log level** in der Konsole dar. Der **default log level** Wert ist **WARNING**, der die **INFO** Nachrichten in der Konsole unterdrückt.

Nun muss man den Server neustarten und die Anwendung unter http://localhost:8888/ aufrufen. Wenn man nun einen Text eingibt, erscheint der eingegebene Text in der Konsole in Eclipse.

3.2.6 Datastores (Datenspeicherung)

Die Google App Engine unterstützt 2 APIs:

- Java Data Objects (JDO)
- Java Persistence API (JPA)

JDO

- Persistente Speicherung von Java-Objekten
- Objektinformationen können in Datenbanken, Dateien o.Ä. abgespeichert werden
- Datenobjekte können ohne Kenntnisse der Speichermechanismen bearbeitet werden

JPA

- Schnittstelle zur Vereinfachung der Zuordnung und Übertragung von Objekten zu Datenbankeinträgen
- Lösung für objektrelationale Abbildung: Laufzeit-Objekte in relationale Datenbanken (nicht für objektorientierte Datenstrukturen vorgesehen) speichern

Nun sollen die eingegebenen Daten auch gespeichert werden, um z.B. mehrere Gästebuch-Einträge speichern und anzeigen zu können.

1. Die Datei src/SignGuestbookServlet.java öffnen
2. Die Datei folgendermaßen ändern

```
package guestbook;

import com.google.appengine.api.datastore.DatastoreService;
import com.google.appengine.api.datastore.DatastoreServiceFactory;
import com.google.appengine.api.datastore.Entity;
import com.google.appengine.api.datastore.Key;
import com.google.appengine.api.datastore.KeyFactory;
import com.google.appengine.api.users.User;
import com.google.appengine.api.users.UserService;
import com.google.appengine.api.users.UserServiceFactory;

import java.io.IOException;
import java.util.Date;

import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class SignGuestbookServlet extends HttpServlet
{
    public void doPost(HttpServletRequest req,
                       HttpServletResponse resp) throws IOException
    {
        UserService userService = UserServiceFactory.getUserService();
        User user = userService.getCurrentUser();

        String guestbookName = req.getParameter("guestbookName");
        Key guestbookKey =
            KeyFactory.createKey("Guestbook", guestbookName);
        String content = req.getParameter("content");
        Date date = new Date();
        Entity greeting = new Entity("Greeting", guestbookKey);
        greeting.setProperty("user", user);
        greeting.setProperty("date", date);
        greeting.setProperty("content", content);

        DatastoreService datastore =
            DatastoreServiceFactory.getDatastoreService();
        datastore.put(greeting);

        resp.sendRedirect(
            "/guestbook.jsp?guestbookName=" + guestbookName);
    }
}
```


Gibt nun der Benutzer eine Nachricht ein, wird diese in der `Entity greeting` gespeichert. Des Weiteren werden der Name des Benutzers und das Datum abgespeichert. Nachdem nun alle benötigten Daten für eine Nachricht in `greeting` gespeichert wurden, wird die `Entity` nun in den `DatastoreService datastore` gespeichert.

Alle Nachrichten werden nun in dieselbe `Entity` Gruppe gespeichert (jedes `Greeting` hat denselben Elternteil), wodurch der Benutzer nach dem Abschicken die Nachricht sofort lesen kann. Würde man diese Anwendung nun hochladen und viele Besucher würden diese nutzen, würde man ein Problem bekommen, da man in dieselbe `Entity` Gruppe nur einen Eintrag/Sekunde schreiben kann.

Nun muss man die guestbook.jsp noch ändern, um die Nachrichten aus dem Datastore anzuzeigen. Außerdem muss ein Formular eingefügt werden, damit der Benutzer seine Nachricht eintippen und abschicken kann.

1. Die Datei war/guestbook.jsp öffnen
2. Die Datei folgendermaßen abändern:

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ page import="java.util.List" %>
<%@ page import="com.google.appengine.api.users.User" %>
<%@ page import="com.google.appengine.api.users.UserService" %>
<%@ page import="com.google.appengine.api.users.UserServiceFactory" %>
<%@ page import="com.google.appengine.api.datastore.
                 DatastoreServiceFactory" %>
<%@ page import="com.google.appengine.api.datastore.
                 DatastoreService" %>
<%@ page import="com.google.appengine.api.datastore.Query" %>
<%@ page import="com.google.appengine.api.datastore.Entity" %>
<%@ page import="com.google.appengine.api.datastore.FetchOptions" %>
<%@ page import="com.google.appengine.api.datastore.Key" %>
<%@ page import="com.google.appengine.api.datastore.KeyFactory" %>
<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<html>
    <head>
        <link type="text/css" rel="stylesheet"
            href="/stylesheets/main.css" />
    </head>

    <body>
```

```
<%
    String guestbookName = request.getParameter("guestbookName");
    if (guestbookName == null)
    {
        guestbookName = "default";
    }
    pageContext.setAttribute("guestbookName", guestbookName);
    UserService userService = UserServiceFactory.getUserService();
    User user = userService.getCurrentUser();
    if (user != null)
    {
        pageContext.setAttribute("user", user);
%>

<p>Hello, ${fn:escapeXml(user.nickname)}! (You can
<a href="<%= userService.createLogoutURL(request.getRequestURI()) %>"
>sign out</a>.)</p>

<%
    }
    else
    {
%>

<p>Hello!
<a href="<%= userService.createLoginURL(request.getRequestURI()) %>"
>Sign in</a> to include your name with greetings you post.</p>

<%
    }
%>

<%
    DatastoreService datastore =
        DatastoreServiceFactory.getDatastoreService();
    Key guestbookKey =
        KeyFactory.createKey("Guestbook", guestbookName);
    // Run an ancestor query to ensure we see the most up-to-date
    // view of the Greetings belonging to the selected Guestbook.
    Query query = new Query("Greeting", guestbookKey).
        addSort("date", Query.SortDirection.DESCENDING);
    List<Entity> greetings = datastore.prepare(query).
        asList(FetchOptions.Builder.withLimit(5));
    if (greetings.isEmpty())
    {
%>

<p>Guestbook '${fn:escapeXml(guestbookName)}' has no messages.</p>
```



```
<%
    }
    else
    {
%>

<p>Messages in Guestbook '${fn:escapeXml(guestbookName)}'.</p>
<%
        for (Entity greeting : greetings)
        {
            pageContext.setAttribute("greeting_content",
                    greeting.getProperty("content"));
            if (greeting.getProperty("user") == null)
            {
%>
<p>An anonymous person wrote:</p>
<%
            }
            else
            {
                pageContext.setAttribute("greeting_user",
                    greeting.getProperty("user"));
%>
<p><b>${fn:escapeXml(greeting_user.nickname)}</b> wrote:</p>
<%
            }
%>
<blockquote>${fn:escapeXml(greeting_content)}</blockquote>
<%
        }
    }
%>

<form action="/sign" method="post">
    <div><textarea name="content" rows="3" cols="60"></textarea></div>
    <div><input type="submit" value="Post Greeting" /></div>
    <input type="hidden" name="guestbookName"
        value="${fn:escapeXml(guestbookName)}"/>
    </form>

  </body>
</html>
```

Anmerkung: Damit ein Benutzer keine bösartigen Skripte in die Textbox eingeben kann, um z.B. anderen Usern zu schaden, muss man die JSTL (JSP Standard Tag Library) Funktion `fn:escapeXml` nutzen, um dies zu unterbinden.

Beispiel:

```
<c:set var="string1" value="This is first String."/>
<c:set var="string2" value="This <abc>is second String.</abc>"/>
```

Ausgabe mit `escapeXml()`:

```
string (1) : This is first String.
string (2) : This <abc>is second String.</abc>
```

Ausgabe ohne `escapeXml()`:

```
string (1) : This is first String.
string (2) : This is second String.
```

Mit dem im JSP entwickelten `Query` und der `List<Entity> greetings` kann man nun die eingegebenen Nachrichten abfragen und nach Datum sortiert absteigend anzeigen lassen.

Der lokale Development Server nutzt eine lokale Datei als Datenspeicherung. Will man nun die bisherigen Nachrichten nicht mehr anzeigen lassen, muss man die Datei local_db.bin löschen:

1. Den Pfad workspace\Guestbook\war\WEB-INF\appengine-generated öffnen
2. local_db.bin löschen

Für genauere Angaben zum `DataStore` ist dieser Link sehr hilfreich: https://developers.google.com/appengine/docs/java/datastore.

3.2.7 Statische Dateien

Will man nun noch ein bisschen am Aussehen der Anwendung feilen, muss man natürlich auch andere Dateien einbinden, z.B. Bilder, CSS Stylesheets, Javascript Code, Filme oder Flash-Animationen.

Als kleines Beispiel wird nun eine CSS-Datei eingebunden:

1. Im Verzeichnis war/ ein neues Verzeichnis stylesheets anlegen (new => folder)
2. Im Verzeichnis stylesheets einen neue Datei main.css anlegen
3. Diese soll folgenden Inhalt haben:

```
body
{
    font-family: Verdana, Helvetica, sans-serif;
    background-color: #FFFFCC;
}
```

4. Die Datei war/guestbook.jsp öffen und die CSS zuweisen

```
<html>
    <head>
        <link type="text/css" rel="stylesheet"
            href="/stylesheets/main.css" />
    </head>

    <body>
        ...
    </body>
</html>
```

Nun kann man die Anwendung unter http://localhost:8888/ neu laden und sieht die Anwendung nun mit dem verwendeten CSS.

4. Workshop

4.1. Einführung

Die Aufgabe dieses Workshops ist es, einen tiefer-führenden Einblick in die Programmierwelt der Google App Engine zu geben.

Hierfür werden wir ein Forum auf Basis der Google App Engine bauen. Es begrüßt seine Besucher zunächst mit einer Übersicht der bereits vorhanden Threads. Meldet sich der Benutzer an, so kann er selbst neue Threads erstellen.

Wird auf einen Thread geklickt, so sieht man die Beiträge in chronologisch sortierter Reihenfolge. Die Verfasser sollen ein Bild zu jedem Beitrag hochladen können, das unterhalb des Beitrags angezeigt werden soll.

Ebenso gibt es die Möglichkeit, sich per E-Mail über neue Beiträge informieren zu lassen.

Das klingt nach viel Arbeit – jedoch erleichtert uns die Google App Engine diese.

4.2. Vorbereiten der Arbeitsumgebung

Damit wir unser Forum aufbauen können müssen wir zunächst unsere Arbeitsumgebung vorbereiten. Dafür gibt es zwei Möglichkeiten.

1. Wir verwenden eine virtuelle Maschine
2. Wir konfigurieren unser eigenes System

Im Folgenden werden wir uns für den Weg über die virtuelle Maschine entscheiden.

Eine einfache Schritt-für-Schritt-Anleitung, wie wir unsere eigene Arbeitsumgebung (Windows und OSX) einrichten, finden wir hier: https://developers.google.com/appengine/docs/java/gettingstarted/?hl=de.

Der einfachste Weg die Arbeitsumgebung vorzubereiten, besteht darin, das fertige VMWare-Image von https://mega.co.nz/#!bF52WZQZ!AqdEYV-5kc3_bwKmMOzNADG1d7QhFfSl5E-L5eAmq5ZI herunterzuladen. Es gibt verschiedene Möglichkeiten, die Maschine zu starten:

Windows XP oder neuer:

- VMWare-Player (gibt es kostenlos bei VMWare)
- VMWare Workstation 9 (die kommerzielle Variante)

OSX 10.7 oder neuer

- VMWare Fusion 5 (kommerziell)
- Parallels Desktop 8 (kommerziell)

4.3. Der erste Start

Haben wir nun eine Virtualisierungssoftware installiert und die VM heruntergeladen bzw. entpackt (ZIP-Datei, lässt sich unter Windows und Mac OS X standardmäßig entpacken), können wir diese mit einem Doppelklick auf **GAEUbuntu.vmx** bzw. **GAEUbuntu.vmware** starten.

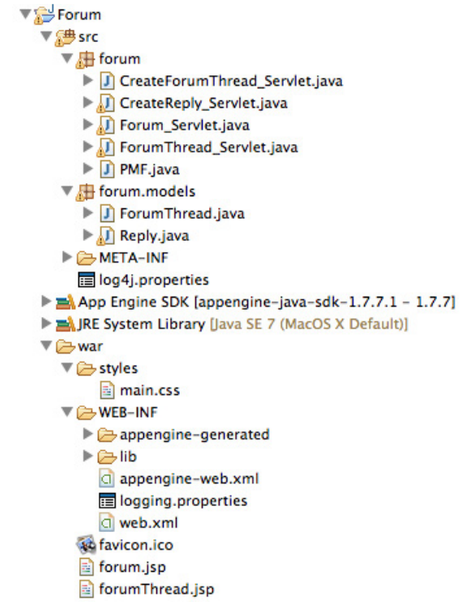
Nachdem wir uns bei Ubuntu eingeloggt (Benutzer: GAE, Kennwort: Engine) haben, starten wir Eclipse, indem wir auf die Verknüpfung auf der linken Leiste der Arbeitsfläche klicken.

4.3.1 Das Projekt

Nachdem wir Eclipse das erste Mal gestartet haben, sehen wir, dass bereits ein Projekt existiert. Es beinhaltet schon einige Dateien, die uns die Arbeit etwas erleichtern sollen.

4.3.1.1 Die Struktur

Die grobe Struktur wurde ja bereits im Beispiel-Kapitel erläutert. Daher werden hier nur die Klassen, die für unser Forum wichtig sind, kurz erklärt.



Unser Projekt besteht aus mehreren Teilen: Der GAE-Teil und der vorgefertigte Teil, der uns bei unserem Vorhaben unterstützen soll. Das Package „forum“ beinhaltet die Klassen, die für die Logik zuständig sind. Im Package „models“ sind die Datenmodelle für unser Forum untergebracht.

4.3.1.2 Vorhandene Dateien kurz erklärt

Fangen wir an uns die vorhanden Klassen etwas genauer anzusehen:

Die Dateien CreateForumThread_Servlet und CreateReply_Servlet beinhalten die Logik, um einen Thread, bzw. einen Beitrag zu erstellen. Sie verarbeiten die Eingaben des Benutzers und speichern sie in der Datenbank.

Forum_Servlet und ForumThread_Servlet sind für das Auslesen der einzelnen Informationen aus der Datenbank zuständig. Die Informationen werden dann an die jeweiligen JSP-Dateien weitergeleitet.

Die JSP-Dateien forum.jsp und forumThread.jsp sind nur zum Anzeigen der Informationen gedacht. Sie verpacken die Informationen in HTML-Code.

4.4. Auf dem Weg zum eigenen Forum

Die erste Aufgabe besteht darin, den Server in Eclipse zu starten. Ist dies geschafft, öffnen wir Chromium und rufen die Seite http://localhost:8888/ auf.

Wie wir sehen, gibt es noch nicht viel! Zu Beginn müssen wir die Datei bestimmen, die aufgerufen werden soll, sobald man unsere Internetadresse eingibt.

Lösung:

Im Projektnavigator wechseln wir zu der Datei web.xml und tauschen `index.html` mit `forum` aus.

```
<welcome-file-list>
    <welcome-file>forum</welcome-file>
</welcome-file-list>
```

Danach den Server neu starten und die Adresse erneut aufrufen.

4.4.1 Statische Dateien einbinden

Nachdem wir unser Forum nun zum ersten Mal gesehen haben, stellen wir fest, dass es nicht wirklich hübsch ist. Glücklicherweise lässt sich das leicht ändern – mit CSS. In unserem Projekt gibt es schon eine vorgefertigte Datei main.css. Binden wir diese nun ein.

Lösung:

In den beiden JSP-Dateien müssen wir die CSS-Datei im Header-Bereich einbinden.

```
<head>
    <link type="text/css" rel="stylesheet" href="/styles/main.css" />
</head>
```

4.4.2 Benutzer

Wenn wir nun das Forum starten, fällt schnell auf, dass wir keine neuen Threads erstellen können.

Wir benötigen erst den `UserService`. Durch ihn bekommen wir dann den (aktuellen) `User`, mit dem wir uns einloggen können.

Lösung:

```
UserService userService = UserServiceFactory.getUserService();
User user = userService.getCurrentUser();
```


Hier haben wir nun den aktuellen Benutzer. Ist der Benutzer nicht angemeldet, wird ein null-Objekt zurück gegeben.

```
if(user != null)
    userURL = userService.createLogoutURL(req.getRequestURI());
else
    userURL = userService.createLoginURL(req.getRequestURI());
```

Dieser Code muss in allen Servlets eingebaut werden.

Nun können schon die ersten Threads erstellt werden. Sollte der Thread nach dem Erstellen nicht sofort angezeigt werden, muss die Seite neu geladen werden. Dies ist eine Limitierung der Offlinevariante von GAE.

4.4.3 Eingabe speichern

Wir können uns nun endlich anmelden und neue Threads erstellen. Jedoch werden Beiträge, die wir schreiben, nicht gespeichert! Also müssen wir diese nun in der Datenbank ablegen, damit sie angezeigt werden können!

Lösung:

Wir brauchen zunächst den `PersistenceManager`, um eine Verbindung zum Datenspeicher herzustellen. Wir holen uns dann aus dem Request die `ThreadID`, in dem wir unseren neuen Beitrag erstellen möchten. `KeyFactory` wandelt uns den `String` in einen von der GAE intern verwendeten `Key` um.

Mit diesem bekommen wir den gewünschten Thread und können dort den neuen Beitrag einfügen.

Mit `close()` wird die Datenbankverbindung geschlossen. Änderungen an Objekten, die während einer offenen Datenbankverbindung durchgeführt wurden, werden automatisch gespeichert.

```
PersistenceManager pm = PMF.get().getPersistenceManager();
Key k = KeyFactory.stringToKey(threadID);

forumThread = pm.getObjectById(ForumThread.class, k);

try
{
    Reply reply = new Reply(user, title, content, date, forumThread);
    forumThread.addReply(reply);
}
finally
{
    pm.close();
}
```

4.4.4 Email-Benachrichtigung

Nun haben wir schon ein einfaches, funktionierendes Forum erstellt. Wir wollen aber gerne einen Thread abonnieren, damit wir benachrichtigt werden, wenn dort ein neuer Beitrag verfasst wurde. Dazu wird (der Einfachheit halber) jedes Mal eine Email an alle Abonnenten verschickt, sobald ein Beitrag erstellt wurde.

Lösung:

Wichtig ist, dass wir ein Eingabe-Feld haben, in dem wir festlegen können, ob wir per Email benachrichtigt werden möchten.

```
<div class="formEntry">Emailbenachrichtigung</br>
    <select name="emailNotify">
        <option value="DONT_CHANGE">Nicht ändern</option>
        <option value="NOTIFY">Benachrichtige mich</option>
        <option value="DONT_NOTIFY">Benachrichtige mich nicht</option>
    </select>
</div>
```

Mit `sendMail()` werden die Benutzer, die sich für dieses Thema angemeldet haben, per Email benachrichtigt. Dies geschieht durch die Java Mail API, die von der GAE bereitgestellt wird.

```
private void sendEmail()
{
    for(User user : forumThread.getUsersToNotify())
    {
        Properties props = new Properties();
        Session session = Session.getDefaultInstance(props, null);

        String msgBody = content;

        try
        {
            Message msg = new MimeMessage(session);
            msg.setFrom(new InternetAddress("xyz@abc.com",
                                            "Thomas Winkler"));
            msg.addRecipient(Message.RecipientType.TO,
                             new InternetAddress(user.getEmail(), "Hallo"));
            msg.setSubject("Im Forum: " + forumThread.getTitle() +
                           " ist ein neuer Eintrag geposted worden");
            msg.setText(msgBody);
            Transport.send(msg);
        }
        catch(AddressException e)
        {}
        catch(MessagingException e)
        {}
        catch(UnsupportedEncodingException e)
        {}
    }
}
```


Am Ende müssen wir noch zwei Methoden nach dem Erstellen eines neuen Beitrags aufrufen. Hier sehen wir die zusätzliche Methode `setMailNotify()`. Diese ist bereits erstellt und muss nur noch vor `sendMail()` aufgerufen werden.

```
try
{
    Reply reply = new Reply(user, title, content, date, forumThread);
    forumThread.addReply(reply);
    setMailNotify();
    sendEmail();
}
```

4.4.5 Bilderupload

Unser Forum ist schon fast fertig. Wir möchten es jedoch noch etwas aufpeppen. Am einfachsten geht das mit Bildern. Jeder Beitrag-Ersteller soll auch ein Bild hochladen können. Dieses wird dann jeweils unterhalb des Beitrags angezeigt.

Lösung:

Für den Bilderupload ist der `BlobStoreService` von Google zuständig. Dieser wird in die JSP-Datei forumThread.jsp eingebunden. Dies muss unbedingt das erste Kommando sein, da es sonst zu einer Fehlermeldung kommen kann. Dies liegt an der Offline-Version der GAE.

```
<body>
<div id="main">
<%  BlobstoreService blobstoreService =
                        BlobstoreServiceFactory.getBlobstoreService();
```

Dann müssen wir unser Formular anpassen und ein neues Feld für das Bild hinzufügen.

```
<div class="formEntry">Bitte wähle ein Bild für den Beitrag aus:</br>
    <input type="file" name="image" accept="image/*">
</div>
```

Danach müssen wir noch die URL, die aufgerufen wird, anpassen, damit der `BlobStoreService` das hochgeladene Bild als `Blob` verarbeiten kann. Der `EncodingType` wird auf `multipart/form-data` festgelegt.

```
<div id="form">
    <form action="<%= blobstoreService.createUploadUrl("/createReply") %>"
                                        method="post" enctype="multipart/form-data">
```

Jetzt muss das Servlet, dass für die Erstellung des Beitrags zuständig ist, den `Blob` auslesen und den `BlobKey` zusammen mit dem Beitrag speichern. Der Teil für das Anzeigen wurde bereits vorgefertigt.

```
private BlobstoreService blobstoreService =
                        BlobstoreServiceFactory.getBlobstoreService();
private BlobKey blobKey;
public void doPost(HttpServletRequest req, HttpServletResponse resp)
                                                throws IOException
{

    Map<String, List<BlobKey>> blobs = blobstoreService.getUploads(req);
    List<BlobKey> blobKeys = blobs.get("image");
    blobKey= null;
    if(blobKeys != null)
        blobKey = blobKeys.get(0);
```

Dann müssen wir den `BlobKey` noch mit der `Reply` speichern. Dafür gibt es bereits einen Konstruktor!

```
try
{
    Reply reply = new Reply(user, title, content, date, blobKey, forumThread);
    forumThread.addReply(reply);
```

4.5. Bei Google veröffentlichen

Wir haben endlich unser Forum fertig. Wäre doch nun schade, wenn es niemand zu Gesicht bekommen würde! Richtig, oder?

Also machen wir uns nun ans Veröffentlichen. Wir brauchen dazu ein Google-Konto (Google-ID). Dann besuchen wir die Seite https://appengine.google.com/start. Dort klicken wir auf „Create Application“. Je nach Google-ID müssen wir uns zunächst per SMS (oder Telefonanruf) authentifizieren. War dies nun erfolgreich, folgen wir einfach der Beschreibung auf dem Bildschirm. Google leitet uns durch diesen Prozess.

Danach klicken wir in Eclipse rechts unten auf „Sign in to Google...“. Zum Schluss müssen wir in der Datei appengine-web.xml unser Forum benennen => `<application>NAME</application>`. Geben wir dafür unseren vorhin eingerichteten `application-identifier` an.

Jetzt können wir unsere App auf AppSpot von Google veröffentlichen. Damit ist unsere Applikation im Internet global verfügbar und kann von jedem auf der Welt genutzt werden.