DHBW
Duale Hochschule
Baden-Württemberg
Mosbach

Blog

Programmieren II - Programmentwurf

Prof. Dr. Helmut Neemann

11. Oktober 2017

Versionen

Datum	Stand
11.10.17	Initiale Version

Organisatorisches

- Die Abgabe des Programmentwurfs erfolgt spätestens am Freitag, den 24.12.2017 als gepackte Quellen im ZIP-Format per `pushci`.
- Die Bearbeitung der Aufgabe erfolgt in Gruppen von maximal drei Studierenden.
- Jede Gruppe fertigt eigenständig eine individuelle Lösung der Aufgabenstellung an und reicht diese wie oben angegeben ein.
- Die geforderte Funktionalität muss von Ihnen selbst implementiert werden.
- Sie können sich Anregungen aus anderen Projekten holen, allerdings muss in diesem Fall die Herkunft der Ideen bzw. von Teilen des Quellcodes im Kommentar Ihrer Quelldatei(en) vermerkt sein.

Viel Spaß und viel Erfolg bei der Bearbeitung!

Aufgabenstellung

Es soll ein einfaches Blog implementiert werden. Diese soll es ermöglichen, Artikel einzustellen welche über einen WEB-Server ausgeliefert werden.

Anforderungen

1. Nicht funktional
   1.1) Der Code soll im Paket `de/vorlesung/projekt/[Gruppen-ID]` liegen, damit alle Lösungen parallel im Source-Tree gehalten werden können.
   1.2) Es dürfen keine Pakete Dritter verwendet werden! Einzige Ausnahme sind Pakete zur Vereinfachung der Tests und der Fehlerbehandlung.
   Empfohlen sei hier `github.com/stretchr/testify/assert` und `github.com/pkg/errors`.
   1.3) Alle Source-Dateien und die PDF-Datei müssen die Matrikelnummern aller Gruppenmitglieder enthalten.
2. Allgemein
   2.1) Die Anwendung muss nur ein Blog verwalten können. Mehrere Blogs auf dem selben Server sind nicht erforderlich.
   2.2) Die Anwendung soll unter Windows und Linux lauffähig sein.
   2.3) Es soll sowohl IE 11 als auch Firefox, Chrome und Edge in der jeweils aktuellen Version unterstützt werden. Diese Anforderung ist am einfachsten zu erfüllen, indem Sie auf komplexe JavaScript/CSS „spielereien“ verzichten und ein 90er Jahre Look&Feel in kauf nehmen. ;-)
3. Sicherheit
   ~~3.1) Die Web-Seite soll nur per HTTPS erreichbar sein.~~
   ~~3.2) Der Zugang für die Autoren soll durch Benutzernamen und Passwort geschützt werden.~~
   3.3) Die Passwörter dürfen nicht im Klartext gespeichert werden.
   3.4) Es soll „salting“ eingesetzt werden.
   ~~3.5) Alle Zugangsdaten sind in einer gemeinsamen Datei zu speichern.~~
4. Ein Blog-Beitrag
   4.1) Ein Beitrag besteht aus dem eigentlichen Text,
   4.2) dem Datum der Erstellung,
   4.3) dem Namen des Autoren und
   4.4) den Kommentaren der Leser.
5. Ein Kommentar
   5.1) Ein Kommentar besteht aus dem eigentlichen Text,
   5.2) dem Datum der Erstellung und
   5.3) dem Nickname des Lesers.
6. WEB-Oberfläche, Autoren
   ~~6.1) Die Anmeldung soll über eine Web-Seite erfolgen~~.

- ~~Zur weiteren Identifikation des Nutzers soll ein Session ID Cookie verwendet werden.~~
- ~~Der Session Timeout soll 15 min betragen, und über Flags einstellbar sein.~~

6.2) Ein Autor soll sein Passwort ändern können.

6.3) Ein Beitrag soll erstellt

6.4) bearbeitet und

6.5) gelöscht werden können.

7. WEB-Oberfläche, Leser

7.1) Zunächst soll der neueste Beitrag zu sehen sein.

7.2) Es soll möglich sein, zu älteren Beiträgen zu navigieren.

7.3) Das Erstellungsdatum und der Autor des Beitrags soll ersichtlich sein.

7.4) Kommentare sollen ebenfalls angezeigt werden.

7.5) Anzeige nach Erstellungsdatum sortiert, neueste Kommentare oben

7.6) Das Erstellungsdatum und der „Nickname“ des Autors jedes Kommentars soll ersichtlich sein.

7.7) Ein Leser soll einen neuen Kommentar abgeben können.

7.8) Eine Authentifizierung ist nicht erforderlich.

7.9) Es soll ein „Nickname“ eingegeben werden können.

8. Storage

8.1) Jeder Beitrag soll incl. der Kommentare in einer Datei im Dateisystem gespeichert werden.

9. Konfiguration

~~9.1) Die Konfiguration soll komplett über Startparameter erfolgen. (Siehe Package `flag`)~~

~~9.2) Der Port muss sich über ein Flag festlegen lassen.~~

~~9.3) „Hart kodierte“ absolute Pfade sind nicht erlaubt.~~

10. Betrieb

~~10.1) Wird die Anwendung ohne Argumente gestartet, soll ein sinnvoller~~ „default“ ~~gewählt werden.~~

~~10.2) Nicht vorhandene aber benötigte Order sollen ggfls. angelegt werden.~~

~~10.3) Die Anwendung soll zwar HTTPS und die entsprechenden erforderlichen Zertifikate unterstützen, es kann jedoch davon ausgegangen werden, dass geeignete Zertifikate gestellt werden. Für Ihre Tests können Sie „self signed“ Zertifikate verwenden. Es ist nicht erforderlich zur Laufzeit Zertifikate zu erstellen o.ä..~~

Optionale Anforderungen

11. Kommentare
    11.1) Kommentare sollen zunächst nicht für Leser sichtbar sein.
    11.2) Autoren können Kommentare freischalten, so dass diese für die Leser sichtbar werden.
12. Nickname
    12.1) Der Nickname soll in einem Cookie gespeichert werden.
    12.2) Wird ein neuer Kommentar angelegt, soll der vorhanden Nickname im entsprechenden Feld voreingestellt sein.
13. Schlagworte Ansicht
    13.1) Für jeden Beitrag soll der Autor eine Anzahl von Schlagworten angeben können.
    13.2) Die Schlagworte sollen unter dem Beitrag angezeigt werden.
    13.3) Beim Click auf ein Schlagwort sollen alle Beiträge angezeigt werden, welche mit dem entsprechenden Schlagwort versehen sind.

WEB-Server in Go

Es gibt einige interessante Techniken, welche die Implementierung von WEB-Servern in Go unterstützen.

Der folgende sehr gelungene Vortrag sei empfohlen:

Mat Ryer: „Building APIs“
Golang UK Conference 2015
`https://youtu.be/tIm8UkSf6RA`

Eine weitere Quelle für Best-Practices findet sich in diesem Vortrag:
Peter Bourgon: „Best Practices in Production Environments“
QCon London 2016
`https://peter.bourgon.org/go-best-practices-2016/`

Abgabeumfang

- Der kompletter Source-Code incl. der Testfälle.
- Dokumentation des Source-Codes incl. Klassendiagramm
  Die Dokumentation setzt sich aus zwei Bestandteilen zusammen: Zum einen aus der Dokumentation des Sourcecodes. Diese erfolgt am geeignetsten im Sourcecode selbst. Zum anderen aus der Dokumentation der Architektur. Diese soll geeignet sein, einem Außenstehenden einen Überblick über das Gesamtprojekt zu verschaf-

fen, indem Fragen beantwortet werden wie: „Aus welchen Komponenten besteht das System?“ oder „Wie arbeiten die Komponenten zusammen?“. Bei Objektorientierten Projekten bietet sich z.B. ein Klassendiagramm an, um die Beziehungen zwischen den Klassen darzustellen, ist aber nicht ausreichend.

- Es soll **eine** PDF-Datei abgegeben werden, welche folgendes enthält:
  - Architekturdokumentation
  - Anwenderdokumentation
  - Dokumentation des Betriebs.
  - Jedes Gruppenmitglied ergänzt eine kurze Beschreibung des eigenen Beitrags zur Projektumsetzung ab (eine Seite reicht).
- Für die Bewertung werden nur die Sourcen und die eine PDF-Datei herangezogen.
- Alle Source-Dateien und die PDF-Datei müssen die Matrikelnummern aller Gruppenmitglieder enthalten.

Abgabe

Die Abgabe soll per `pushci` (`https://infprogs2.dhbw-mosbach.de`, User: `student`, PWD: `prog216`) erfolgen. Hierbei handelt es sich um einen Dienst, welcher aus dem Intranet und dem Internet erreichbar ist.

Dieser Dienst übernimmt Ihre Datei, packt sie aus, überprüft einige formale Kriterien, compiliert die Sourcen und führt alle Tests aus. Eine Datei kann nur abgegeben werden, wenn alle Tests „grün“ sind.

Wer bereits mit einem Continuous-Integration-System gearbeitet hat, ist mit diesem Vorgehen sicher vertraut. Für Studierende, die noch nicht mit einem CI-System gearbeitet haben, ist das möglicherweise ungewohnt. Es ist daher dringend angeraten, sich mit diesem Dienst frühzeitig vertraut zu machen. Es wäre sehr ungeschickt, erst am Tag der Abgabe um 23:55 das erste mal zu versuchen eine Datei „hochzuladen“.

Sie können beliebig oft eine Datei „hochladen“ und überprüfen lassen. Wenn diese Überprüfung erfolgreich war, kann die Datei abgegeben werden. Dies sollten Sie natürlich nur einmal mit Ihrer finalen Lösung tun.

Bewertungskriterien

Ihr Programmentwurf wird nach folgenden Kriterien (mit abnehmender Gewichtung aufgeführt) bewertet:

1. Abbildung der Anforderungen (s.o.)
2. Strukturierung und Konsistenz des Quellcodes
3. Ausreichende Testabdeckung des implementierten Codes. (gemessen mit `go test`

`-cover`)

4. Sinnhaftigkeit der Tests (Sonderfälle, Grenzfälle, Orthogonalität usw.)
5. Qualität von Kommentaren und Dokumentation
6. Benutzerfreundlichkeit der Schnittstellen (APIs)
7. Optionale Features
8. Das Design der Web-Seite spielt keine Rolle solange sie benutzbar ist. Es ist kein aufwendiges JavaScript/CSS erforderlich!