PROJEKT FINAL-PAYMENT

(RUBY ON RAILS)

AGENDA

- Stand des Projekts
- Arbeit mit dem Framework
- Fazit

WIE WEIT IST DAS PROJEKT?

WIR SIND FERTIG.

- Alle Requirements implementiert
- Code auf Github: **http://tinyurl.com/finpay**
- Requirements mit Feature Tests abgedeckt (ca. 80% CC)
- Zeitaufwand (gesamt): ca. 70h

EINGESETZTE TECHNOLOGIEN (AUSWAHL)

- Ruby on Rails
- Template-Sprache: HAML
- Clientseitiges Scripting: CoffeeScript
- Templating-Framework: Bootstrap
- Über ein Dutzend Gems aus unterschiedlichen Bereichen

ETWA 1800 ZEILEN CODE
(TEMPLATES, RUBY SCRIPTS, COFFEESCRIPT)

ARBEIT MIT DEM FRAMEWORK

- (Subjektive) Einschätzungen/Erfahrungen
- Gut (+++) bis überhaupt nicht gut (---)
- (Wenig projektspezifischer Code, das kennt ihr schon!)
- Zeitangaben

TEMPLATING (+++)

- Template-Engines durch GEMs dynamisch austauschbar
- Wir verwenden HAML
  - Ultra kurze Notation von HTML-Templates
  - Intuitiv, sehr schnell zu lernen
  - Viele Features, schnelle Ergebnisse
  - Gemeinsam mit Bootstrap auch noch schick!
- Zeitaufwand Layout/Optik: etwa 5 Stunden

PERSISTENZ UND VALIDIERUNG (++)

- Nutzung von ActiveRecord und ActiveModel als ORM
  - Großer Funktionsumfang
  - Wir verwenden Migrations, Scopes, Validierung, Observer, Seeds, ...
  - Ausgezeichnete Dokumentation
- Unsere Engine: SQLite3
- Zeitaufwand Datenmodellierung: etwa 10 Stunden

ERWEITERBARKEIT: GEM-KOSMOS (++)

- Gut:
  - Es gibt für (fast) jedes Problem irgendwo ein Gem
  - Wiederkehrende Probleme haben Standardgems hervorgebracht
    (CanCan, Devise, Bootstrap, ...)
- Nicht so gut:
  - Man muss sich regelmäßig einen Überblick verschaffen
  - Einarbeitungsaufwand (z.B. CanCan-Autorisierung, ...)
  - "kleinere" Gems nicht immer up-to-date oder nicht mehr gepflegt

BOOTSTRAPPING UND INSTALLATION (+++)

- Bootstrapping: binnen Sekunden
- Installiert euch selbst eine Kopie der Software!
  1. Github Repository clonen
  2. `$> bundle install`
  3. `$> rake db:setup`
  4. `$> rails server`
  5. **localhost:3000**
  6. Mit *test1@example.com* und Passwort *123* einloggen

FORMULARE: BOOTSTRAP_FORM (++)

- angepasster Formbuilder für Bootstrap
- lediglich Definition der Felder und Typen nötig
- automatisches Labelling und Fehlermarkierung
- Standard Formbuilder etwas umständlicher

AUTHENTIFIZIERUNG: DEVISE (++)

- komplett durch Devise realisiert
- wenig Änderungen nötig
- alles was man braucht!

AUTORISIERUNG: CANCAN (++)

- einfache DSL
- gute Integration in die Controller
- verlässt sich viel auf Konventionen, Autorisierung mit verschachtelten Resourcen etwas umständlich

ROUTING (++)

- nur Restful-Resources benutzt
- bei dynamischen Segmenten muss Controlleraction und Routenname mit angegeben werden

SICHERHEIT (-)

- OOTB: SQL-injection, CRSF, Code-Injection, Mass-Assignment
- Secure by default! :D
- **ABER** gestern mal wieder: CVE-2013-0276 (5. dieses Jahr)
- Rails Magic schafft Angriffsvektoren :(

LOKALISIERUNG (-)

- Key-Value Aufzählung
- Muss manuell gepflegt werden
  - Aufwand zum sicherstellen dass die Anwendung übersetzt worden ist: <1h

TESTBARKEIT (++)

- RSpec: einfach und extrem lesbar
  - BDD: der Test ist die Spezfikation
  - ausgereiftes GUI testen
- ...jedoch ist die Ausführung etwas langsam

ERLERNBARKEIT (POSITIVES)

- relativ wenig Ruby-Kenntnisse erforderlich
- um produktiv zu arbeiten hat Folgendes ausgereicht
  - Syntax einprägen
  - die grundlegende Sprachkonstrukte verstehen
- Rails liest sich wie eine DSL für Web-Frameworks
- ausgezeichnet dokumentiert
- viele Lösungen im Internet zu finden

ERLERNBARKEIT (BEDINGT NEGATIVES)

- Convention over Configuration und Duck-Typing
  - Vorteilhaft für schnelles Schaffen von Ergebnissen
  - Aber nicht immer klar wie die Anwendung verdrahtet ist
  - Doku deckt aber alles ab

ZEITLICHES

Kategorie	Zeit [h]
Tickets	53,5
Sonstiges	15,5
davon Doku	16,5
Gesamt	69

BURNDOWN CHART

FAZIT

Webanwendungen mit Rails zu bauen ist hochgradig **effizient** , schnell **erlernbar** und **macht Spaß**.

Trotzdem gibt es einige **Fallstricke**, die besonderer Aufmerksamkeit bedürfen. Es schadet also nicht sich *trotzdem* tiefer in Ruby und Rails einzuarbeiten.