Software Engineering
ECE444

Project Requirements and Suggestions

© 2019

Project requirements subset

1. Implementation of all backend matter must be Ruby-on-Rails.
2. HTML5-based responsive website interface for all front-end matter.
3. The Website has to run out of Heroku, a cloud service that runs on AWS.
4. GitHub must be used for managing all source code;
5. the Master branch must always be runnable, and no other are branches allowed.
6. The (email-)id used to access GigHub must be a UofT email address. TA's and prof must have access via their emails
7. 5. Each module/service must have an externalizable RESTful API

© 2019

Project requirements subset II

8. All access to application Web pages should be via https, and any first visit via http should be automatically redirected to https.
9. Privacy of users must be maintained, so some information should only be displayed, and some actions should only be allowed after user habe properly authenticated themselves.
10. Userids must always be email addresses. Standard “forgot password” functionality required.
11. All captured data must be stored in a database; all displayed info must be obtained from that database.
12. There must be a description of the service and general marketing material easily visible to the general public (i.e., to someone not logged in).

© 2019

Project requirements subset III

13. Each Web page must have links to:
14. an “About” page that describes the service and provides the history of your app (i.e., explain why it is great).
    - a page containing the terms of service
    - a page describing the privacy policy
    - a page with a site map
    - a “Contact Us” page
    - a page with a FAQ
    - twitter, facebook, Instagram (for marketing purposes).
15. When a user is logged in, his/her name should be visible on the Webpage, and there should be a logout button.
16. When a visitor is not logged in, there should be a “My Account Login” and "Create Account" button/link clearly visible near the top.

© 2019

Perhaps consider: Internationalization

Probably too ambitious given our timeframe, but

- consider users understanding other languages, in different timezones and having different standards:
- Ruby i18n gem to, e.g.,
  - put static text into locale files, not directly into views
  - In config/application.rb, leave the app's timezone as UTC, store all times as UTC, and use ActiveSupport::TimeWithZone to display correctly
  - Use strftime and currency helpers to display dates, times, and currency
- ASCII ➔ Unicode: assumptions break
  - #bytes != #chars in string
  - In regexes, `\w` is not the same as `[A-Za-z]`

© 2019

Assumptions about names are often wrong

- People have a first & last name
  ➔ Madonna, Cher, . . .
- Only certain characters appear in names (e.g. for validations)
  ➔ 
- Much more: https://www.kalzumeus.com/2010/06/17/falsehoods-programmers-believe-about-names/

Make your app's Web or mobile UI look professional

- Make your app UI responsive. (Will be taught later. . .)

- getbootstrap.com (CSS + JavaScript)
  - Lots of pre-made CSS styles
  - Line things up in grids, make links that look like buttons, mobile-friendly layouts
- iui-js.org
  - CSS & JS to mimic iOS & Android UIs for your app
- phonegap.com
  - distribute your HTML5+JS mobile app as a "real" iOS or Android app (requires XCode, etc)

Other approaches to mobile app

- To HTML for mobile or go native?
- HTML5 + JS app—*preferred unless specifically lacking in performance or native features*
- Native iOS or Android app?
  - RubyMotion: Ruby compiler down to iOS native code using iOS SDKs (payware)
  - Teacup/MotionKit: DSL for creating views
  - Nitron: iOS MVC-like app framework
  - No free lunch: still need to learn iOS programming model & basic APIs, and purchase/learn iOS dev env

Authentication and Authorization

- Not the same thing!
- Consider using:
  - `omniauth` gem: implements OAuth by adding small number of new routes
    - *Strategies* target different auth providers
    - Provides "developer" strategy & testing support
  - `devise` gem: more complex to integrate, but offers:
    - User model,
    - “email lost password”,
    - confirmation email support
    - session timeouts
    - lockout after too many invalid login attempts
    - etc
  - `CanCan` gem provides *authorization/capabilities*

Keeping secrets

- e.g., for API keys:
- See https://blog.arkency.com/2017/07/how-to-safely-store-api-keys-in-rails-apps/ for possible solutions
- General idea: store encrypted in environment variables:
  - consider using figaro and GPG
  - See: http://saasbook.blogspot.com/2016/08/keeping-secrets.html

Payment processing and sending emails

- Payment processing:
  - Stripe: really easy and nicely integrates with site
  - Paypal: more awkward integration, but works better in international settings.
  - *Strong recommendation:*
    Don't touch a credit card number, ever
- Sending email
  - Mailgun: insulates you from dealing with email directly, works as Heroku addon
  - MailChimp: popular service

Generate Fake Data. . .

To make your demo and testing more realistic:

1. Manually create fake data, then snapshot copy of `db/development.sqlite3`
   - only works on dev, not production (Heroku)
   - brittle if DB schema changes

2. use faker gem and FactoryBot (formerly known as FactoryGirl) gem.
   - create a rake task to generate data
   - search StackOverflow for other similar techniques

Check views with Middleman

- Middleman app/gem can instantiate your Rails views without you running the Rails app
  - just set the variables each view looks for!
- Great for working on views during BDD steps when need view before writing controller methods

Some useful resources

- Heroku Add-ons page
- rubygems.org
- GitHub
- Google search
- StackOverflow