Setting up shop

- git clone https://github.com
  /Salamangkero
  /prosdev-basic-server.git

- javac *.java

- java BasicServer

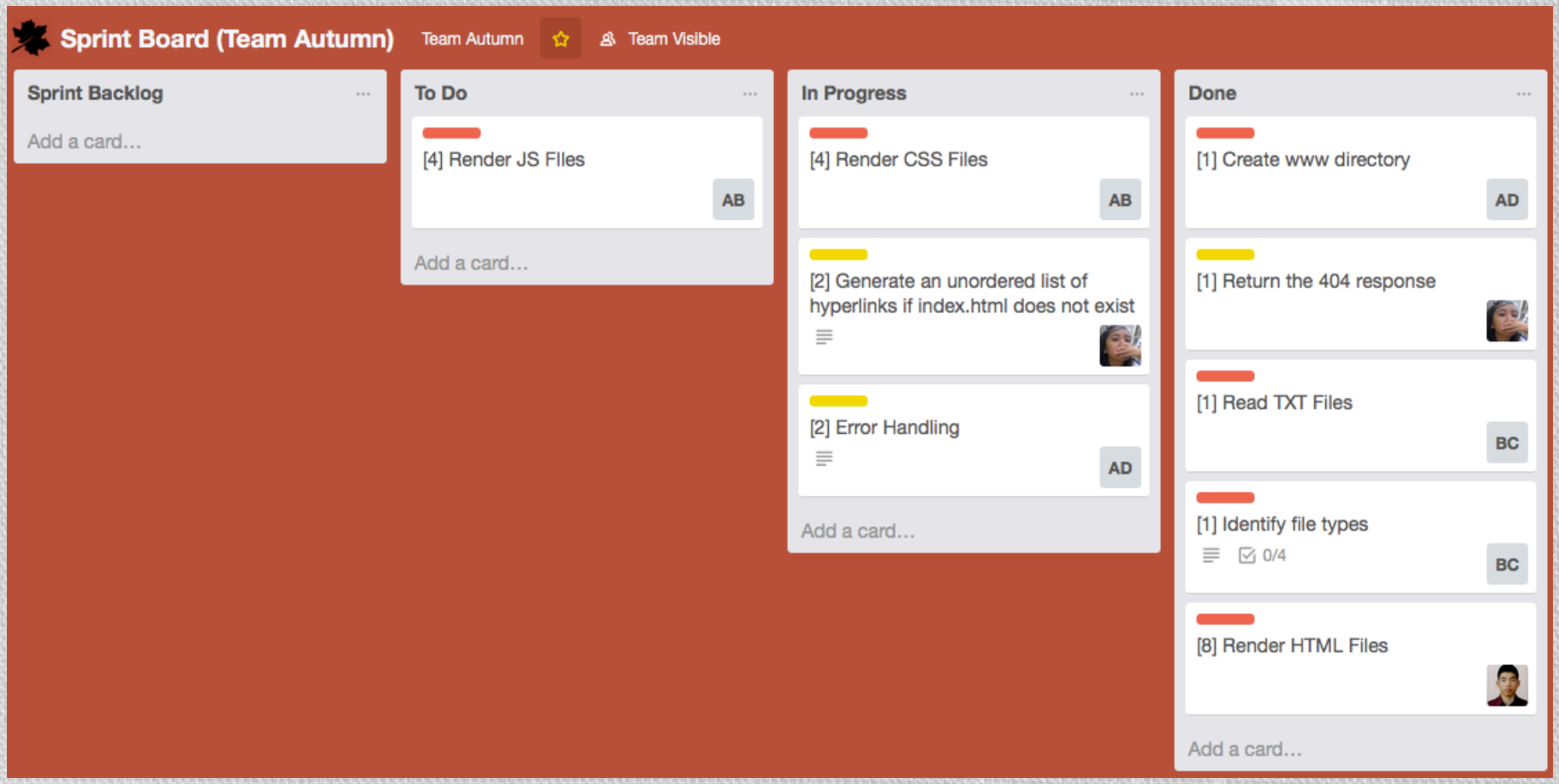
SCRUM LEVEL 1

Sprint Planning (Again)

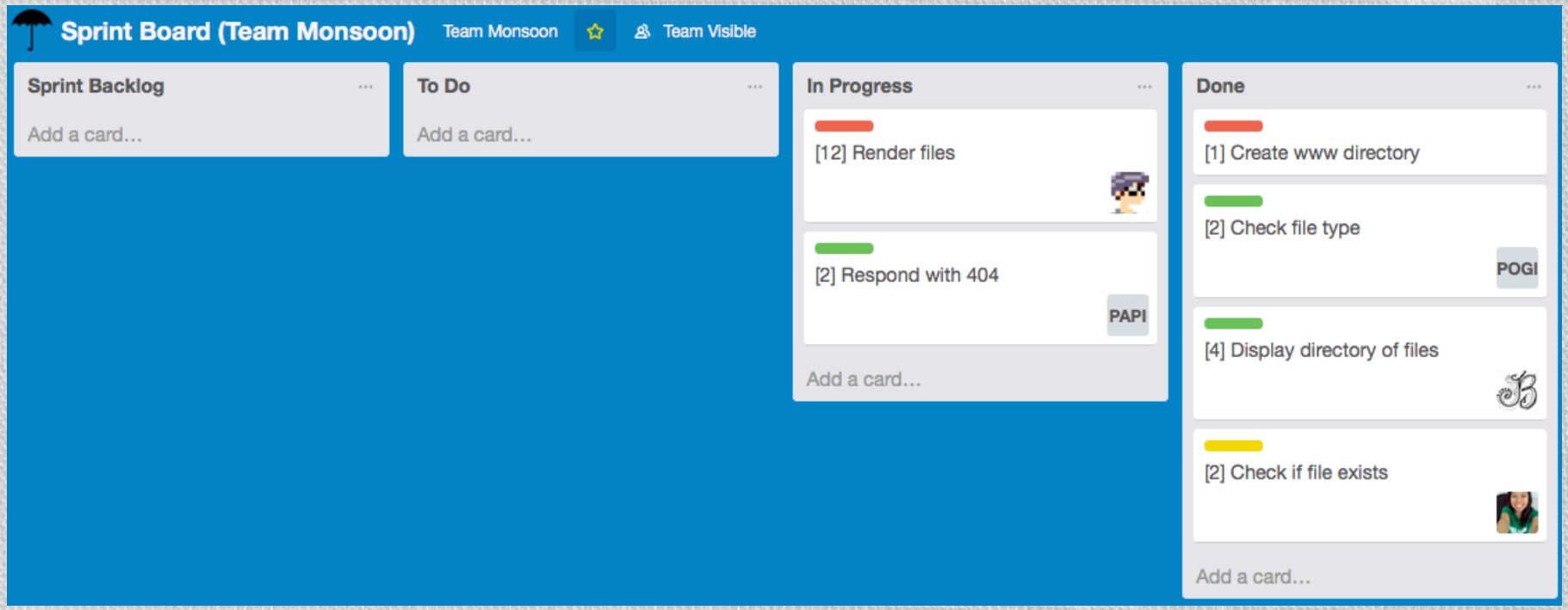
Sprint Planning

- Identifies scope of work available for one sprint
- Prepares sprint backlog by breaking down work into atomic stories
- As a team, estimate time / effort / complexity per story
- Identifies which stories are achievable by the end of the sprint
- Commits those stories by moving from “Sprint Backlog” to “To Do”

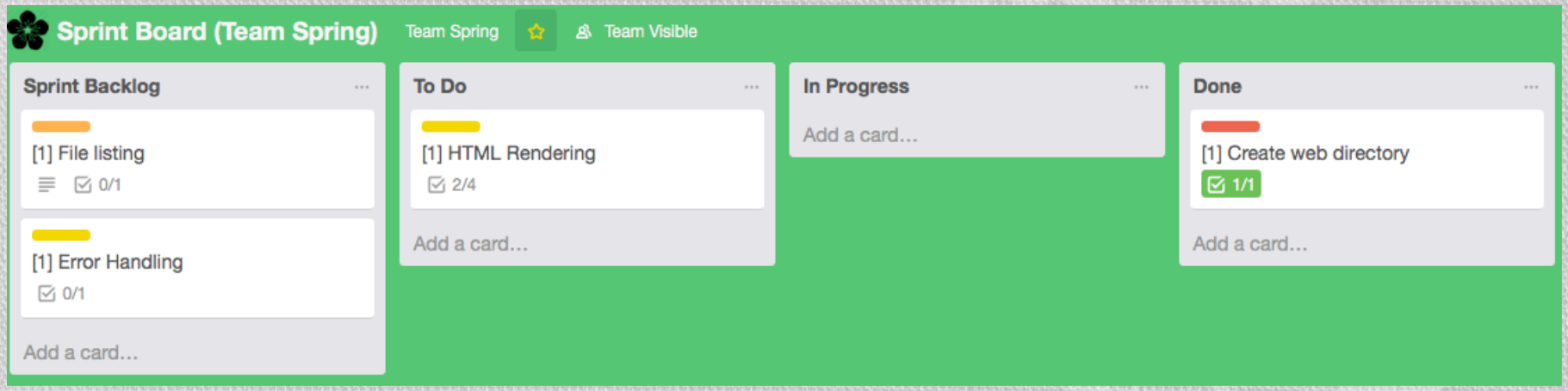
Team Autumn


Sprint Board (Team Autumn)
Team Autumn
Team Visible
Sprint Backlog
Add a card...
To Do
[4] Render JS Files
AB
Add a card...
In Progress
[4] Render CSS Files
AB
[2] Generate an unordered list of hyperlinks if index.html does not exist
[2] Error Handling
AD
Add a card...
Done
[1] Create www directory
AD
[1] Return the 404 response
[1] Read TXT Files
BC
[1] Identify file types
0/4
BC
[8] Render HTML Files
Add a card...

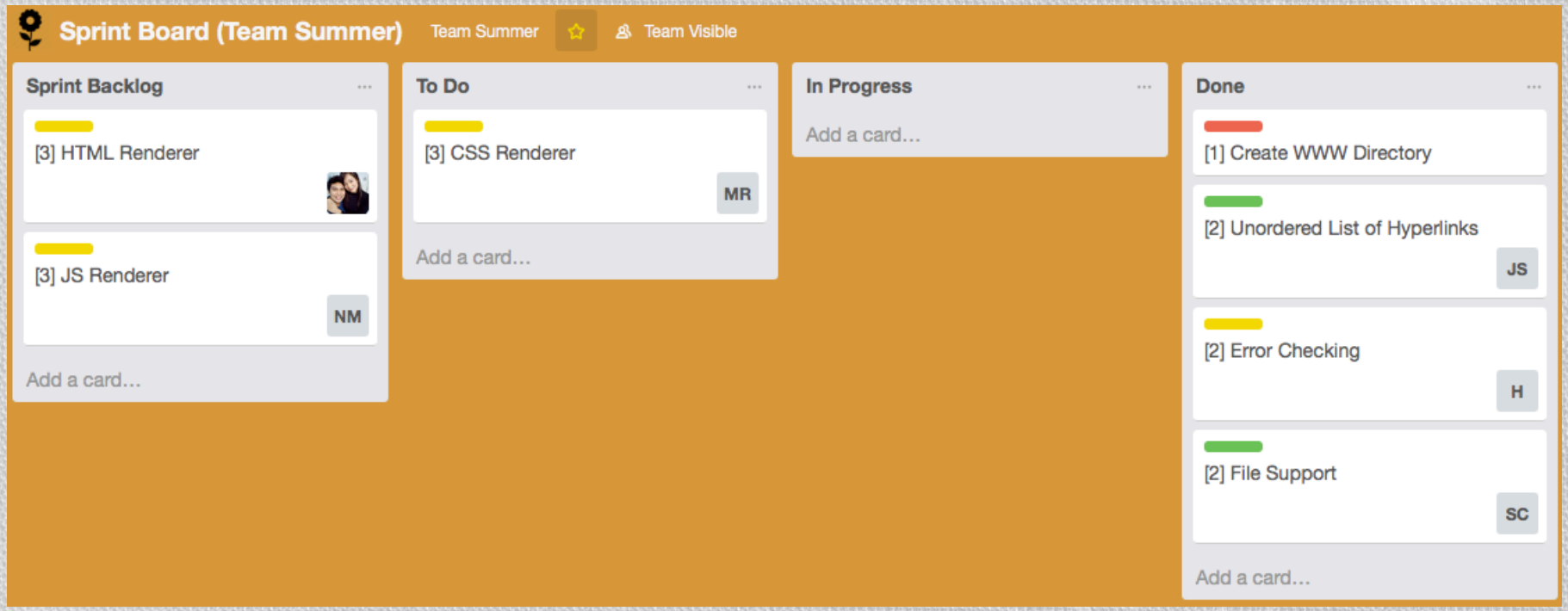
Team Monsoon



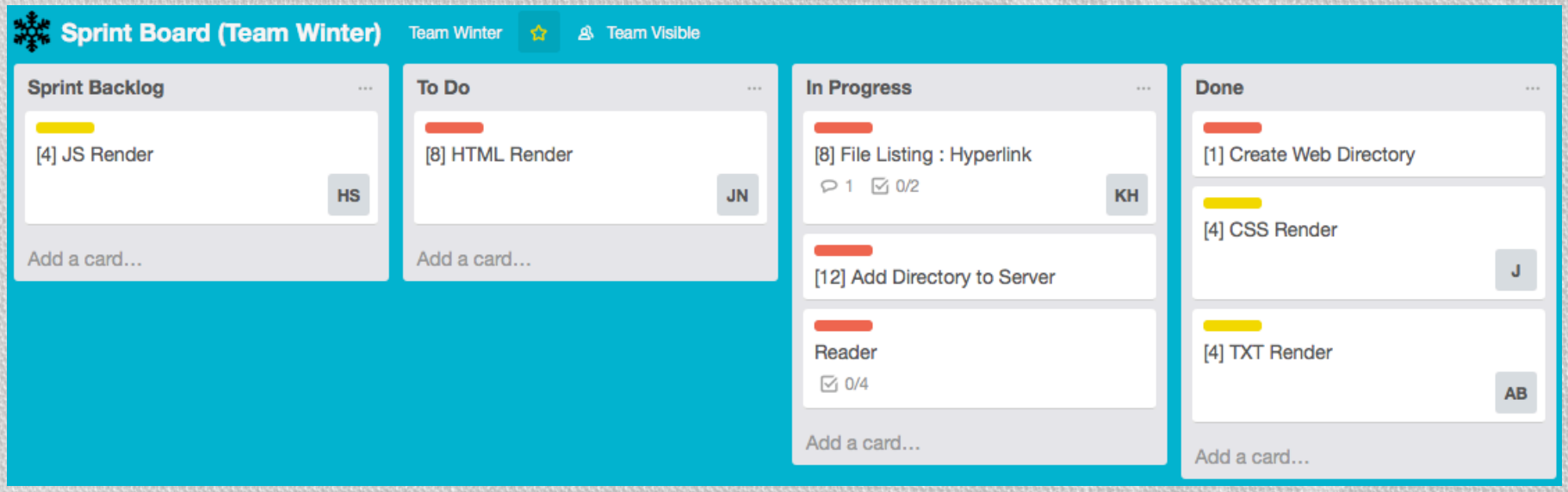
Team Spring


Sprint Board (Team Spring)
Team Spring
Team Visible
Sprint Backlog
[1] File listing
0/1
[1] Error Handling
0/1
Add a card...
To Do
[1] HTML Rendering
2/4
Add a card...
In Progress
Add a card...
Done
[1] Create web directory
1/1
Add a card...

Team Summer


Sprint Board (Team Summer)
Team Summer
Team Visible
Sprint Backlog
[3] HTML Renderer
[3] JS Renderer
NM
Add a card…
To Do
[3] CSS Renderer
MR
Add a card…
In Progress
Add a card…
Done
[1] Create WWW Directory
[2] Unordered List of Hyperlinks
JS
[2] Error Checking
H
[2] File Support
SC
Add a card…

Team Winter


Sprint Board (Team Winter)
Team Winter
Team Visible
Sprint Backlog
[4] JS Render
HS
Add a card…
To Do
[8] HTML Render
JN
Add a card…
In Progress
[8] File Listing : Hyperlink
1
0/2
KH
[12] Add Directory to Server
Reader
0/4
Add a card…
Done
[1] Create Web Directory
[4] CSS Render
J
[4] TXT Render
AB
Add a card…

GIT REFRESHER
GIT WORKFLOWS

L1: CENTRALIZED WORKFLOW

L2: FEATURE-BRANCH WORKFLOW

git-clone, git-pull, git-status

- Git is a DISTRIBUTED version control system

- git-clone creates a copy not of the code but of the entire repository
- origin vs. local vs. remote

- git-pull “pulls” changes (actually, commits) from remote to local
- It fetches changes from the remote and applies it to the local repo

- git-status
- HEAD vs. working copy

git-add, git-checkout, git-reset HEAD, git-rm

- git-add stages a set of changes for commit
- git-checkout will undo your changes, but only if they are not staged
- You will need git-reset HEAD to unstage changes
- Lastly, git-rm removes a file and stages its deletion for commit

git-commit, git-log, git-push

- git-commit “locks in” a set of changes
- It maps a unique identifier to it
- It also maps other data to it (eg. author, DESCRIPTION, timestamp)

- git-log just display the sequence of commits from newest to oldest

- git-push propagates your commits from local to remote
- This fails if your local is not “up-to-date” with the remote

- Remember pull-before-push policy

Centralized Workflow

- One “main” repository
- One “main” branch

- Obvious solution, simple, straightforward
- Perfect for teams transitioning from SVN

- Highly inefficient, creates A LOT of superfluous merge commits

Feature Branch Workflow

- Each “feature” is a set of commits that logically belong together
- The rule still stands, each commit should NOT break the system

- git-branch creates a “branch” of code, that is, a separate “timeline”
- Useful for keeping related commits together

- git-pull is also useful for merging branches, by the way

- Overwhelmed yet? Use git-help ;)

SCRUM LEVEL 2

Daily Scrum

Daily Scrum

- Also known as stand-up meetings, typically < 15 mins
- Always takes place at the same time and same place
- Each member briefly outlines:
  - What he/she did yesterday
  - What he/she will do today
  - What blockers are/were being encountered

- Take note, the goal is to inform others what you're up to
- DO NOT try to solve problems in this meeting, save it for later

Important: KEEP IT SHORT!!!

REST LEVEL 2

HEAD, PUT, POST and DELETE

GET

- Most HTTP Requests are use the GET HTTP method

- GET /users (give me a list of users)
- GET /users?age=17 (give me a list of users aged 17 yo)
- GET /users/21.xml (give me the details of user #21 in XML format)
- GET /users/tyrion (give me the details of user tyrion)

- GET /assets/style.css (give me the stylesheet named style.css)
- GET /favicon.ico (give me the favicon for this domain)

POST

- Semantically speaking, POST means “create a new object”

POST /users HTTP/1.1
Content-Length: 44
Content-Type: application/x-www-form-urlencoded

first_name=Cersei&last_name=Lannister&age=29
or
age=29&first_name=Cersei&last_name=Lannister

POST

- If a POST request succeeds, the proper response is 201 Created
- 200 OK will also be alright IF you are returning an HTTP body

- If it fails because of incorrect parameters, 400 Bad Request
- If the object already exists, 409 Conflict
- Generally speaking, though, 400 Bad Request is a safe catch-all

PUT

- A PUT request is an attempt to update or overwrite an object

```
PUT /users/eddard HTTP/1.1
Content-Length: 11
Content-Type: application/x-www-form-urlencoded

status=dead
```

PUT

- A PUT request is an attempt to update or overwrite an object

PUT /users/eddard HTTP/1.1

Content-Length: 18

Content-Type: application/json

{“status”: “dead”}

PUT

- A PUT request is an attempt to update or overwrite an object

```
PUT /users/eddard HTTP/1.1
Content-Length: 21
Content-Type: application/xml

<status>dead</status>
```

PUT

- A PUT request is an attempt to update or overwrite an object

```
PUT /assets/style.css HTTP/1.1
Content-Length: 46
Content-Type: text/css

.ninja {
  color: #000000;
  visibility: hidden;
}
```

PUT

- When a PUT request succeeds, respond with 204 No Content
- Again, 200 OK is fine IF Content-Length > 0

- If the incoming Content-Type is not supported or inappropriate, use 415 Unsupported Media Type
- If the object is not there, 404 Not Found
- If the parameters are otherwise valid, but would cause conflict, for example, putting role=king for /users/joffrey while /users/robert is still king, use 409 Conflict
- Again, if in doubt, 400 Bad Request

HEAD

- Functionally identical to GET, except it never returns an HTTP body
- Content-Length is still a positive amount
- Useful for saving bandwidth, especially if you already have a cached copy of the object

- Both GET and HEAD methods are idempotent
- PUT and DELETE are also idempotent (but not safe)

DELETE

- Self-explanatory, removes the object
- On success, return 204 No Content
- As usual, if the object does not exist, 404 Not Found

BUILDING A WEB SERVER

With a really demanding client

My needs are “few”… and “simple”…

- **GET**
  - Default index page, XML and JSON support
- **POST**
  - 201 Created, 409 Conflict, 415 Unsupported Media Type
- **PUT**
  - 204 No Content, 400 Bad Request
  - 404 Not Found, 415 Unsupported Media Type
- **HEAD**
  - 200 OK, 404 Not Found
- **DELETE**
  - 204 No Content, 404 Not Found

Tips

- Prefix your feature branches with <team_name>/feature/<desc>
- Your descriptions should be short but still descriptive

- Expected number of stories is AT LEAST 10
- It's ok if you have less, but you'll have some explaining to do
- If you pad with BS stories, you will also have to explain yourselves