13/04/2021 (Sel) 15:15 (GMT+07)

AM sampai 2031-02-26 10:00 AM (GMT + 7). Selama waktu ini, Global Gaming permainan tidak akan tersedia. Kami memohon maaf atas ketic

How we create the Judi Online websites

Designing and building data driven dynamic web applications…

…the one web, domain driven, RESTful, open, linked data way

Explore the domain

This should be clear from the business requirements. It might be food or music or gardening…

Employ a domain expert

Get them to sketch their world

Sketch back at them

Model real (physical and metaphysical) ‘things’ *not* web pages - try to blank from your mind all thoughts of the resulting web site

Do this through the lifetime of the project as you refine your understanding

Identify your domain objects

As you chat and sketch with your domain expert you should build up a picture of the types of things they're concerned with

Make a list of these objects

Identify the relationships between your domain objects

- As your knowledge of the domain increases you'll build up a picture of how your objects interlink
- Sketch basic entity relationship diagrams with your domain expert
- Keep sketching until the picture clears
- The resulting domain model will inform the rest of your project and should be one of the few 'artifacts' your project ever creates

Check your domain model with users

Run focus groups

Speak to users - get them to sketch their understanding of the domain

Sketch back at them

Synthesise the expert model and the user model

User-centric design starts here - if you choose to model things and relationships between those things that users can't easily comprehend no amount of wireframes or personaes or storyboards will help you out

Check to see if your website already deals with some of your domain objects

If it does then reuse this functionality by linking to these pages

You don’t want to mint new URIs for existing objects

Having more than one page per object confuses users and confuses Google

Think of your website as a coherent whole; not as a collection of individual products

As ever, don’t expose your internal organisational structures through your website. Users don’t care about departments or reporting lines

Design your database

Translate your domain model into a physical database schema

Source your data

Check if there are business systems in your organisation able to populate your schema

Check if there are existing websites outside your organisation you can use to populate your schema

Give preferential treatment to any websites that offer their data under a liberal licencing agreement - you can buy in data to help you slice and dice your own data but if you do this you might not be able to provide an open data API without giving away the 3rd party’s business model

If your organisation AND an open data website can provide the data you need consider the danger in minting new identifiers for your own data - can you easily link out / can you easily get links in?

Pipe in your data

Whether you choose to use your business data or buy data or use open data you’ll need a way of piping it into your database schema

You’ll probably have to reshape it to make it suitable for publishing

Make your models

In an MVC framework your models should contain all your business logic

This mean they should capture all the constraints of your database schema plus all the extra constraints of your domain model

Design your URI schema

This should follow naturally from your domain model

This isn’t just about designing URIs for resources you link to - sometimes your pages will be made up of other transcluded resources - all of these subsidiary resources should be addressable too

By making every nugget of content addressable you allow other sites to link to it, improve your bookmarkability and increase your SEO - cf. an individual ‘tweet’

Bear in mind that some representations (specifically mobile) will need smaller, more fragmented representations with lower page weight - designing your subsidiary resources to be addressable allows you to easily deal with this requirement - transclude the content on a desktop machine, link to it on a mobile

The usual rant about persistence

It’s nice if URIs are human readable

It’s also nice if they’re hackable

It’s an absolute prerequisite that they’re persistent

Don’t sacrifice persistence for the sake of prettiness or misguided SEO

URIs are your promise to the web and your users - if you change them or change their meaning you break that promise - links break, bookmarks break, citations break and your search engine juice is lost

* Cool URIs don’t change *

Make hello world pages for your primary domain objects

For now all they need is an h1 with the title of the object

Make hello world pages for your primary aggregations

For now all they need is an h1 with the title of the aggregation and a linked list of things aggregated

Define the data you need to build each of your pages

For each URI define the data you need to build all representations of the object

This should cover all the representations you intend to make- just because the HTML representation doesn’t need to show the updated date doesn’t mean the RSS or Atom or RDF don’t need it

Some resources will transclude others. You don’t need to define the data required for these - just reference the transcluded resource

Build up your HTML pages and other representations

Now you know what data you need you can begin to surface this in your representations

If you’re working in HTML make sure you design your document to be semantically correct and accessible

Try not to think about page layout - that’s the job of CSS not HTML

In general your page should be structured into title, content, navigation - screen readers don’t want to fight through calendar tables etc to get to the content

Add caching and search sitemaps

Knowing what can be cached and for how long is a vital part of designing your user experience

Cache for too long and pages go stale

Don't cache for long enough and you send unnecessary traffic across the wires and place extra strain on your application

Cached pages will also be faster and smoother to render in a browser

And if your users are paying for data on a mobile every extra connection means bigger bills

An example: if you're creating a schedule page for today's TV you want to cache for performance reasons but you don't want to cache it for too long since schedules are subject to change. But you can cache yesterday's schedule more aggressively and last week's schedule more aggressively still

Creating XML search sitemaps helps search engines know which bits of your site have been updated. Which helps them to know which bits to re-index

Apply layout CSS

Add layout CSS to your HTML pages

Experiment with different layouts for your markup by moving elements around the page

Test and iterate

You should be testing with real users at every stage of development but it’s particularly important to conduct usability AND accessibility tests now

It’s like testing traditional wireframes but you’re testing on the real application with real application behaviours and real data (no lorum ipsum)

Sometimes the results of your testing will require changes to layout CSS, sometimes to markup, sometimes to the data you need to surface and sometimes to the underlying domain / data model

Bear in mind if you’re using data from existing business systems there may need to be heavy investment to make changes to that data model and employ the staff to admin those changes

Occasionally it might even mean renegotiating contracts with outside data providers - all design and usability issues are fixable - some just need more lawyers than others :)

Apply décor CSS

Over the top of your wireframe application you can now start to add visual design and branding

This is exactly the same process as taking a paper wireframe and applying design treatments over the top except you’re mainly working in CSS

Experiment with different treatments - see how far you can stretch the design with the markup given

Sometimes you’ll need to add additional markup to hook your CSS off

Now’s the time to add background imagery for headers, dividers, buttons, list items etc so best to open Photoshop / Illustrator to make your design assets

And test and iterate

Never stop testing

Remember that personas are just abstractions of people - it’s always better to use real people

Ideally you should be able to adjust your code / markup / CSS to respond to user requests

If you can afford the recruitment / developer time there’s no better way to test than with a user sitting alongside a developer - the developer can react to user requests, tweak the application and gain instant feedback without the ambiguity that sometimes comes from test reports

Again you should accessibility test - some of the design / décor changes may affect font sizes etc - make sure your users can still read the page

Add any JavaScript / AJAX

By designing your browsable site first and adding in Javascript / AJAX over the top you stand a better chance of making an accessible web site - one that degrades gracefully

As ever Google et al are your least able users - search bots don’t like forms or JavaScript - sites that degrade well for accessibility also degrade well for search engines

Making every subsidiary resource addressable and providing these resources serialised as XML or JSON makes adding AJAX relatively trivial

You’ll probably need to tweak your CSS to adjust to life with JavaScript / AJAX

And test and iterate

Again test your site for accessibility and usability with JavaScript turned on and off

Continue

Follow the same steps for each development cycle

Some development cycles will just be about surfacing new views of the existing domain model; some will require expanding your domain model

Now you know your domain model and have made each domain object addressable layering over new views and more subtle user journeys should be trivial

And keep testing!