Application Architecture Concepts

Enterprise Applications

Abstract

Describe application architecture to take advantage of new technologies, DevOps processes, microservices, and true agile, customer focused design.

Lyons, Christopher W.

Christopher.w.lyons@usps.gov

Introduction

This document is intended to describe an enterprise application architecture to take greater advantage of the technologies and infrastructure that USPS has installed over the past few years. Many of the benefits of this infrastructure and technology is achieved through use across the enterprise through cross-functional teams. Having been on multiple teams, both internal and external, I have seen how many of these technologies could benefit the organization overall and I prepared this document as a reference to how I have approached this infrastructure and as a document for use and adoption by others.

I will try to describe my reasoning for this approach and what caused me to come to this current setup. I will also try to describe how I think this could be fit into how applications are developed and maintained within USPS but this will be limited by what I have seen and worked with and most likely does not cover all scenarios.

The concepts and designs presented in this document were reached in response to actual USPS application design and development issues, not theory or 3rd party products. They were created to meet real business and developer needs with tangible benefits and promising future results.

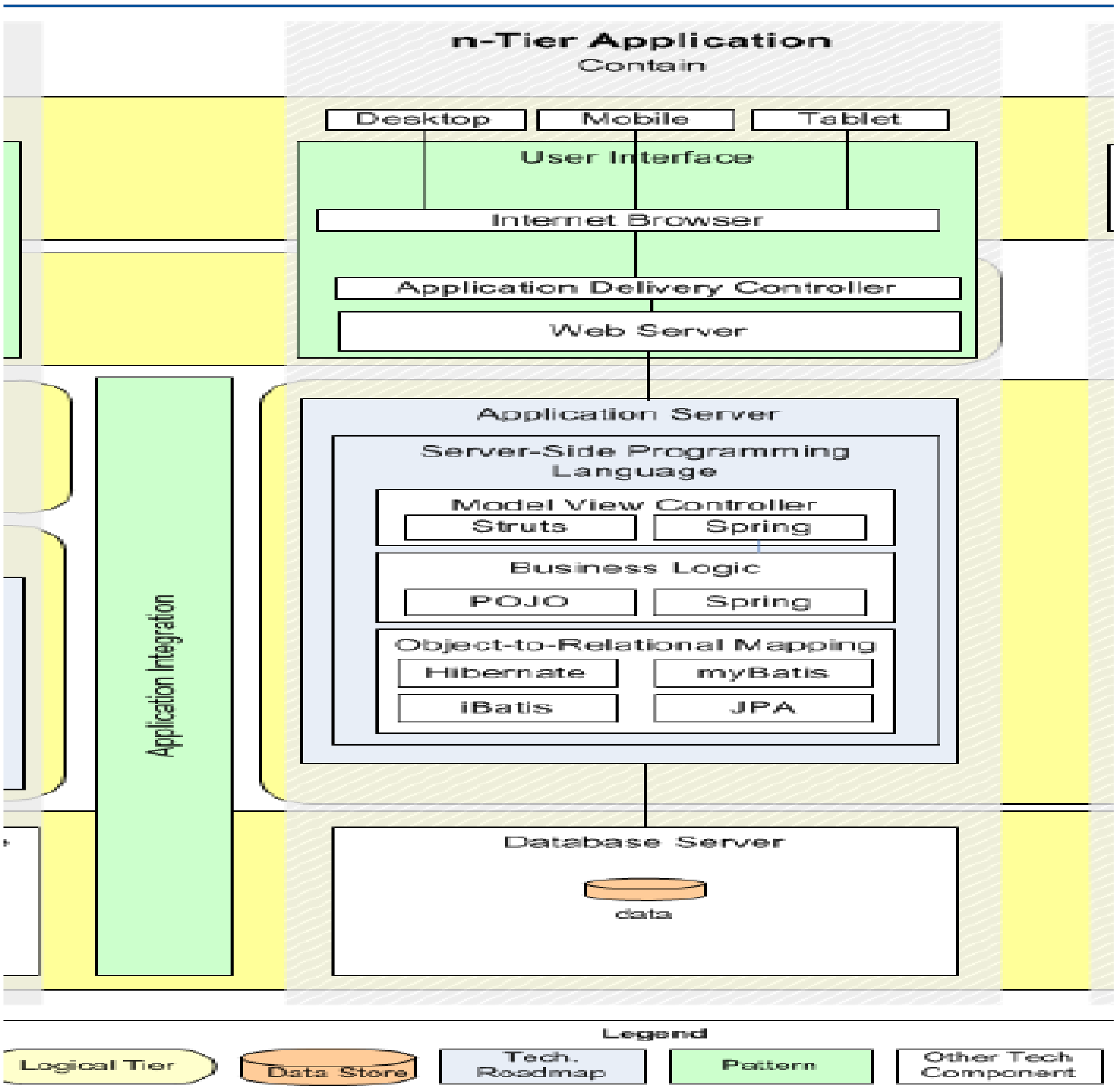
Contents

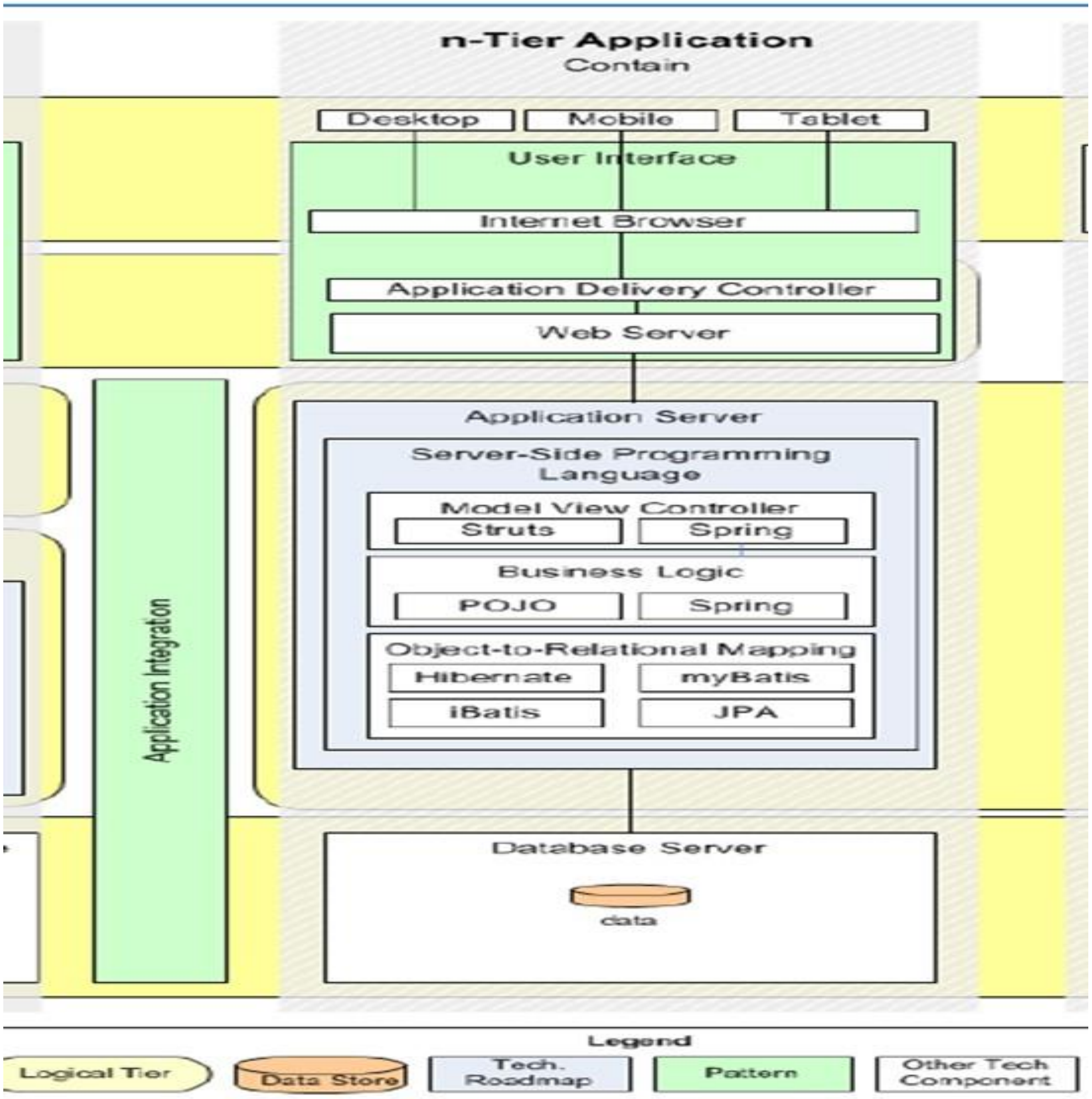
Overview

From a high level perspective there are 3 high level components to describe.

- UI/UX design, infrastructure, architecture, deployment, process
- Application services / microservices design, deployment, process, and architecture
- DevOps integration including testing, monitoring, and CI process

Most applications are an implementation of the following design (From USPS Reference Architecture):

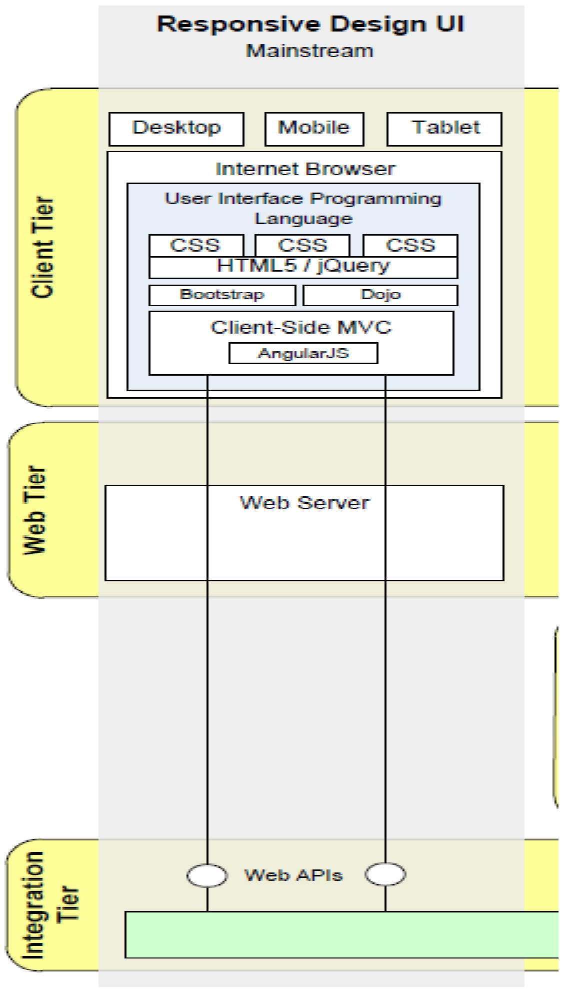


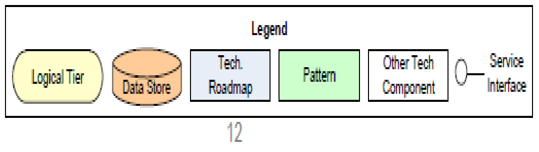


This architecture provides for an easy to execute, easy to scale implementation but is limiting in many ways. Generally this pattern requires an application to be designed as a “silo”, the application team designs and develops the entire stack; the views/pages, the application code/integration layer, and the database design. Many application teams will use frameworks which assist in some ways but limit options in many others. An application team that chooses the struts/tiles framework is generally not able to work with a separate UI/UX team as the framework is intended to work with specific view technology that is not used by UI/UX design resources. Many struts/tiles applications use server-side HTML generation via JSP use.

Design/front end development teams will not work with JSP technology as it is not appropriate to do so. JSP, at its core, is a java servlet and requires a java environment to execute while front-end design teams are required to build designs for browsers using HTML/CSS/and javascript. These are fundamentally different approaches and the process of accepting a "wireframe" deliverable from a design firm and integrating it into a struts/tiles based application vs including a design team as a part of the application team in order to use a true Continuous Integration process. A struts/tiles based application necessarily requires one deliverable to be complete BEFORE the other can begin; the wireframes must be complete before the application code can be written or the application code must be written or complete for the design team to write the UI/UX code based on the application code.

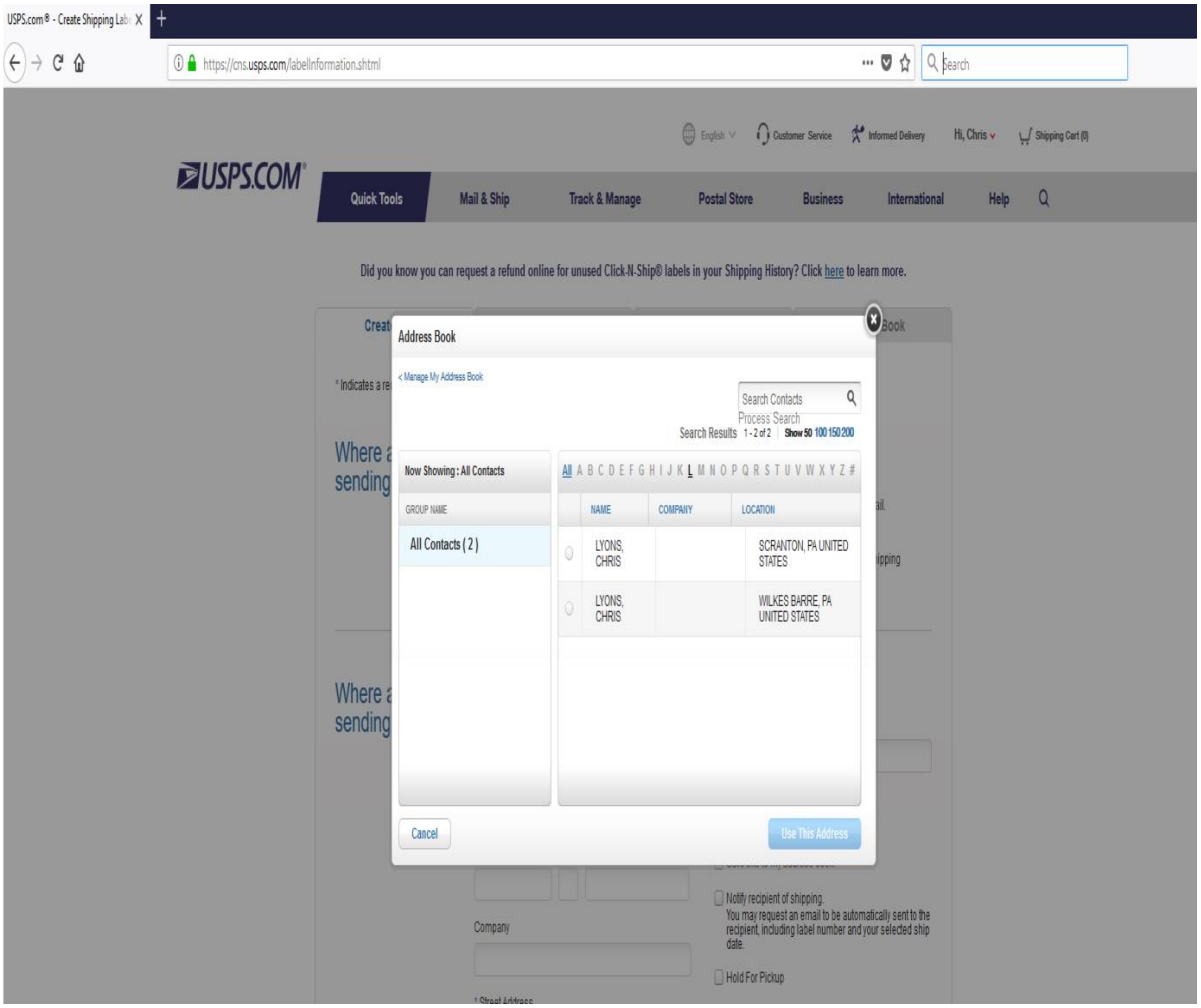
The solution for many of these problems are ALSO described in the USPS Reference Architecture, this design:





The way to read the solutions in this provided architecture is to view the server-side generated content as a "service" to the browser based content, which can then be built using a responsive, rich client experience. The "API's" that the "client tier" in the above design calls is the application server-side code and both teams can build their components of the system based on that spec at the same time thus delivering presentable solutions much faster. From that point optimization/CI/DevOps approaches reducing the complexity of both parts, the browser/client tier, and the server-side application code design.

Microservices are an extension of this approach as the application the end user sees is the browser web page but that webpage can be composed of different services from different applications. A simple USPS example would be Click-N-Ship (CNS).



Did you know you can request a refund online for unused Click-N-Ship® labels in your Shipping History? Click here to learn more.

NAME	COMPANY	LOCATION
LYONS, CHRIS		SCRANTON, PA UNITED STATES
LYONS, CHRIS		WILKES BARRE, PA UNITED STATES

You may request an email to be automatically sent to the recipient, including label number and your selected ship date.

In this application the familiar CNS content underneath the dialog box is loaded from the cns.usps.com website but a developer familiar with the browser debug tools could investigate and find that the AddressBook content comes from gab.usps.com. By expanding this approach USPS applications could be developed as mashups of UI/UX content developed and maintained separately from the server-side code that provides the data. This approach could also free application teams unfamiliar with UI/UX development from developing inferior websites that don't meet USPS standards as the browser content could and should be developed by developers with the appropriate expertise.

Technical Details – UI/UX

The overview described application designs that can take advantage of newer application design techniques, I would like to now describe some of the details of the UI/UX side of the application.

Infrastructure

UI/UX content and the design firms that build it professionally work with static content types. Generally this means HTML/CSS/JS but could be expanded to include anything that is a "static" resource. That is a resource that is sent unchanged by the server that serves that resource. It could mean video and other media, flash files, office documents, PDF's, SVG content, and more. USPS currently operates multiple products for service static content to end users:

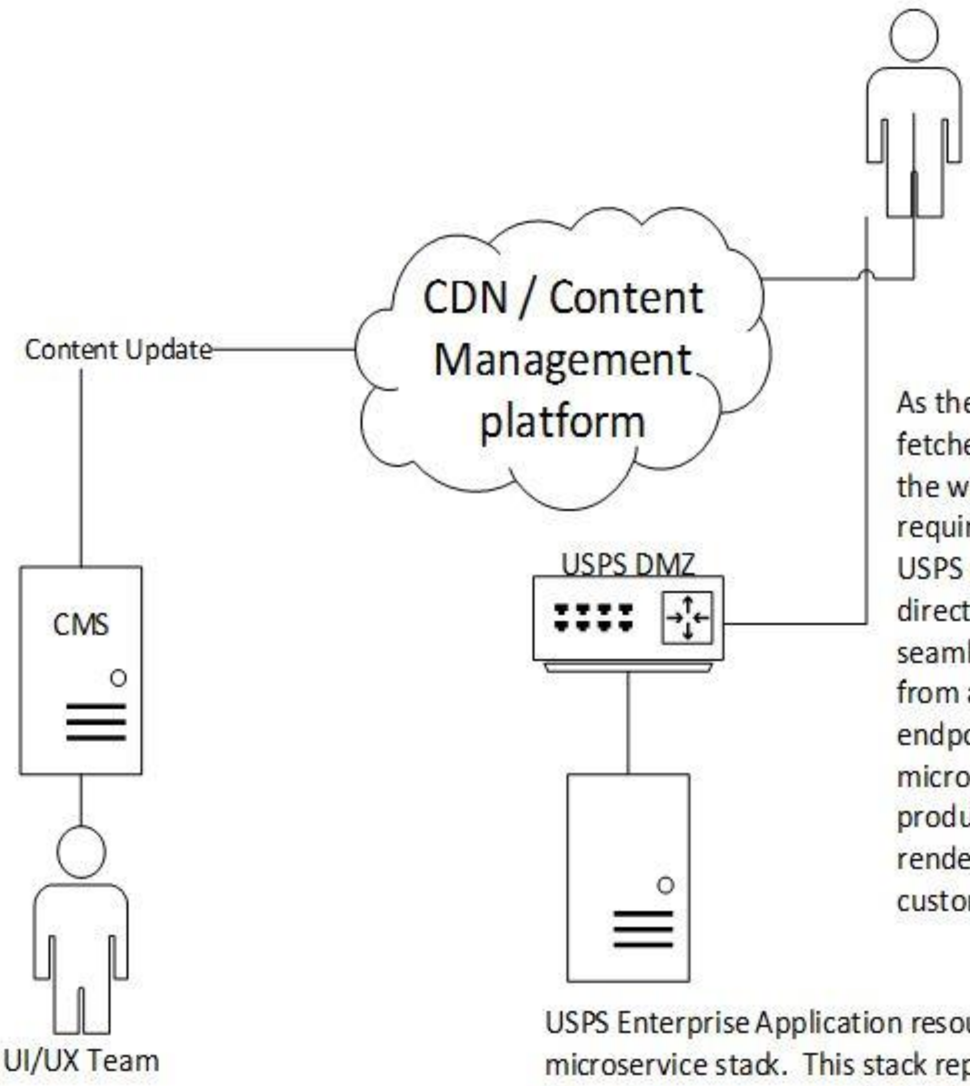
- Apache/IBM IHS/Microsoft IIS webservers – These webservers are the main product by which USPS developed/operated applications serve static content to end users and are a component of the reference architecture.
- Teamsite – This is a product used for static content for blue.usps.gov, liteblue.usps.gov, about.usps.com, and usps.com websites and probably more.
- Akamai CDN – A more recent addition to USPS static content infrastructure. Akamai provides many functions beyond content hosting.

Browsers have been made much more functional, standards compliant, faster, and secure and are a very important, if not the most important, part of an enterprise application. Standards compliance, functionality, and security has improved to a point where javascript and other browser technologies are first class languages treated the same way as java, c#, and c++. That also means that the UI/UX components should be worked by experts and not just any group.

The complexity in website design has encouraged the development of many frameworks and resources to assist engineers in much the same way that frameworks were created for J2EE applications. This is another reason to separate the server-side application code team from the UI/UX development team as there is no enterprise wide standard "platform" for UI/UX code as there is for J2EE applications. The platform is the browser and having multiple teams creating individual environments does not create opportunities for reuse and often creates the problem of websites of widely varying quality which in turn creates a bad experience for the most important people to the enterprise, the customer.

The choice of UI/UX framework / architecture is not as important at the moment as the design of enterprise applications. A proper enterprise content management setup will allow for rapid enhancement and change based on the customer's environment (browser technology). A UI/UX team should have the responsibility of creating and managing ALL content across ALL applications.

Sample Enterprise Architecture using existing USPS products and service



Customer. Exter
accesses a USPS
The USPS.com v
accessed by the
browser via an a
(https://cdn.usp
product).

As the customers browser
fetches and loads content
the website some content
required to be loaded fron
USPS microservice endpoi
directly. The customer's b
seamlessly loads the conte
from a USPS microservice
endpoint (https://
microservice.usps.com/
productjson) and via javas
renders the content for th
customer.

USPS Enterprise Application resource microservice stack. This stack represents just the resources necessary to host 1 microservice. It might consist of 1 or more WebSphere, jBoss, or Tomcat application servers. Further discussions should be made but a webserver (IIS, Apache, etc) should NOT be necessary or used in this case as this resource will ONLY return data in the form of XML or JSON. Using a webserver in unnecessary and wastes resources and security can be provided by existing network devices.

The UI/UX team is responsible for creating and updating all of the website content across all of USPS.com along with integrating the microservice data into that content. The microservice API's are the only thing that needs to be known by both the enterprise application team and the UI/UX team. The website content can be changed at any time without involvement from the application team.

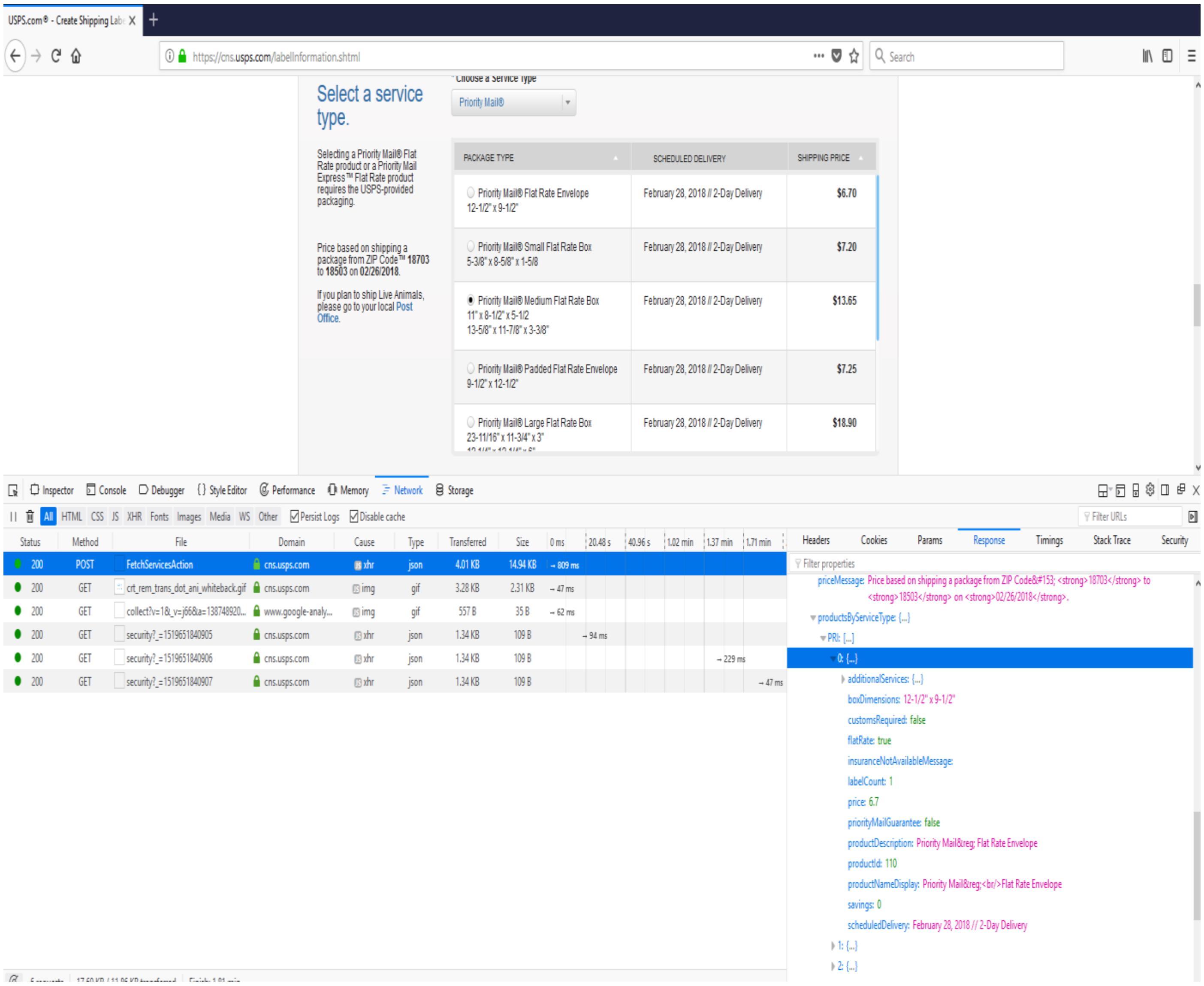
A permanent UI/UX group should be created and this group responsible for all development and maintenance of the USPS “website”. View the entirety of the USPS presence as 1 application. This will create the environment and focus that is needed to reduce duplication, increase standards compliance, and speed changes.

I would strongly recommend a Content Management System (CMS) be used as this will allow for permission management and coordination between the groups since everything should be run through the CMS. USPS does NOT currently have an enterprise Content-Management-System so options would be worth reviewing. Cloud-based products should be given strongest priority as the infrastructure described above allows for static content to be flexibly worked from anywhere. The data from the microservices is the only secure piece of the enterprise and the microservices themselves should be the only thing hosted on private infrastructure.

Technical Details – Server-Side Application

Service Oriented Architecture has been around for a while and microservices are the natural evolution of enterprise services. By treating the browser as a first class component of an enterprise application architecture it becomes apparent that server-side code should be the "services" that the browser client uses. These are basic microservices.

For example, Click-N-Ship(CNS) has a "FetchServicesAction" that the users browser calls when the user wants to get rates and products for their desired label.


USPS.com® - Create Shipping Lab
https://cns.usps.com/labelInformation.shtml
Search
Choose a Service Type
Priority Mail®

Select a service type.

Selecting a Priority Mail® Flat Rate product or a Priority Mail Express™ Flat Rate product requires the USPS-provided packaging.

Price based on shipping a package from ZIP Code™ **18703** to **18503** on **02/26/2018**.

If you plan to ship Live Animals, please go to your local Post Office.

PACKAGE TYPE	SCHEDULED DELIVERY	SHIPPING PRICE
Priority Mail® Flat Rate Envelope 12-1/2" x 9-1/2"	February 28, 2018 // 2-Day Delivery	$6.70
Priority Mail® Small Flat Rate Box 5-3/8" x 8-5/8" x 1-5/8	February 28, 2018 // 2-Day Delivery	$7.20
● Priority Mail® Medium Flat Rate Box 11" x 8-1/2" x 5-1/2 13-5/8" x 11-7/8" x 3-3/8"	February 28, 2018 // 2-Day Delivery	$13.65
Priority Mail® Padded Flat Rate Envelope 9-1/2" x 12-1/2"	February 28, 2018 // 2-Day Delivery	$7.25
Priority Mail® Large Flat Rate Box 23-11/16" x 11-3/4" x 3"	February 28, 2018 // 2-Day Delivery	$18.90

Status	Method	File	Domain	Cause	Type	Transferred	Size	0 ms
200	POST	FetchServicesAction	cns.usps.com	xhr	json	4.01 KB	14.94 KB	→ 809 ms
200	GET	crt_rem_trans_dot_ani_whiteback.gif	cns.usps.com	img	gif	3.28 KB	2.31 KB	→ 47 ms
200	GET	collect?v=1&_v=j66&a=138748920...	www.google-analy...	img	gif	557 B	35 B	→ 62 ms
200	GET	security?_=1519651840905	cns.usps.com	xhr	json	1.34 KB	109 B	→ 94 ms
200	GET	security?_=1519651840906	cns.usps.com	xhr	json	1.34 KB	109 B	→ 229 ms
200	GET	security?_=1519651840907	cns.usps.com	xhr	json	1.34 KB	109 B	→ 47 ms

```
priceMessage: Price based on shipping a package from ZIP Code&#153; <strong>18703</strong> to
              <strong>18503</strong> on <strong>02/26/2018</strong>.
▼ productsByServiceType: {...}
  ▼ PRI: [...]
    ▼ 0: {...}
      ▶ additionalServices: {...}
        boxDimensions: 12-1/2" x 9-1/2"
        customsRequired: false
        flatRate: true
        insuranceNotAvailableMessage:
        labelCount: 1
        price: 6.7
        priorityMailGuarantee: false
        productDescription: Priority Mail&reg; Flat Rate Envelope
        productId: 110
        productNameDisplay: Priority Mail&reg;<br/>Flat Rate Envelope
        savings: 0
        scheduledDelivery: February 28, 2018 // 2-Day Delivery
    ▶ 1: {...}
    ▶ 2: {...}
```

This call was made by the browser to a microservice hosted on the CNS infrastructure. The microservice returns all of the information needed by the page (labelInformation.shtml) to render the response to the user. This microservice could be called by ANY application or ANY page. If a change was desired to this page and a robust CMS system was in place, this change can be made by the UI/UX team without any interaction with the CNS team at all. Only if the microservice API needs to change does the CNS team need to be involved and depending on the change the UI/UX team may not need to be involved.

Microservices should return JSON for their format. JSON is smaller over the wire than XML, more efficient, easier to use and understand, more flexible, and can be read by both servers and browsers natively. If JSON cannot be used for some reason XML is acceptable as it can also be used by both browser and server-to-server communication.

Microservices do NOT need struts or spring and it would be strongly advised to not use either of them. Modern application servers, even free-open source ones, contain enough infrastructure to make these libraries unnecessary. These frameworks overly complicate application code and increase the ramp up time of new developers to join a team and begin productive work. These services should be written as thinly as possible to allow for simple migration and deployment via DevOps/CI operations.

Microservice Rules and Recommendations

Microservices should be developed carefully to provide maximum benefit at a low cost. The following are principals used to that effect:

- Use jax-rs framework or .net equivalent (for .net applications)
- Do not use session or any other server state mechanism
- Output should be JSON or XML.
- Cross-domain services should enable and configure CORS but if careful analysis is performed JSONP is acceptable in cases where no private data is contained. In a previous example “FetchServiceAction” contains private data and should not return JSONP. CNS should enable CORS for cross-domain requests
- Microservices cannot contain or return UI content of any kind.
- Should be designed to be as platform independent as possible. Use configuration if needed. Do not use the filesystem or OS unless that is a specific function of the microservice.
- Document microservice API’s as a regular API.
- Always deploy through scripting and automation infrastructure. This is VERY important to build and maintain flexibility within the organization. White glove deployments are not scriptable and degrade the environment for everyone.
- The microservice team members should not be on the UI team for the app they are working on. This creates laziness in the code as developers can write the microservice and the UI that interacts with it and cut corners.
- Always rotate teams. Microservice code should be small and focused on a specific task. Having a microservice so specialized that only 1 person is familiar with and comfortable working on has been designed wrong. Individual developers should be able to pick up a microservice comfort level within 2 weeks. Rotate developers every 6 – 12 months, or more or less often based on desires but rotation should always occur after a period of time.