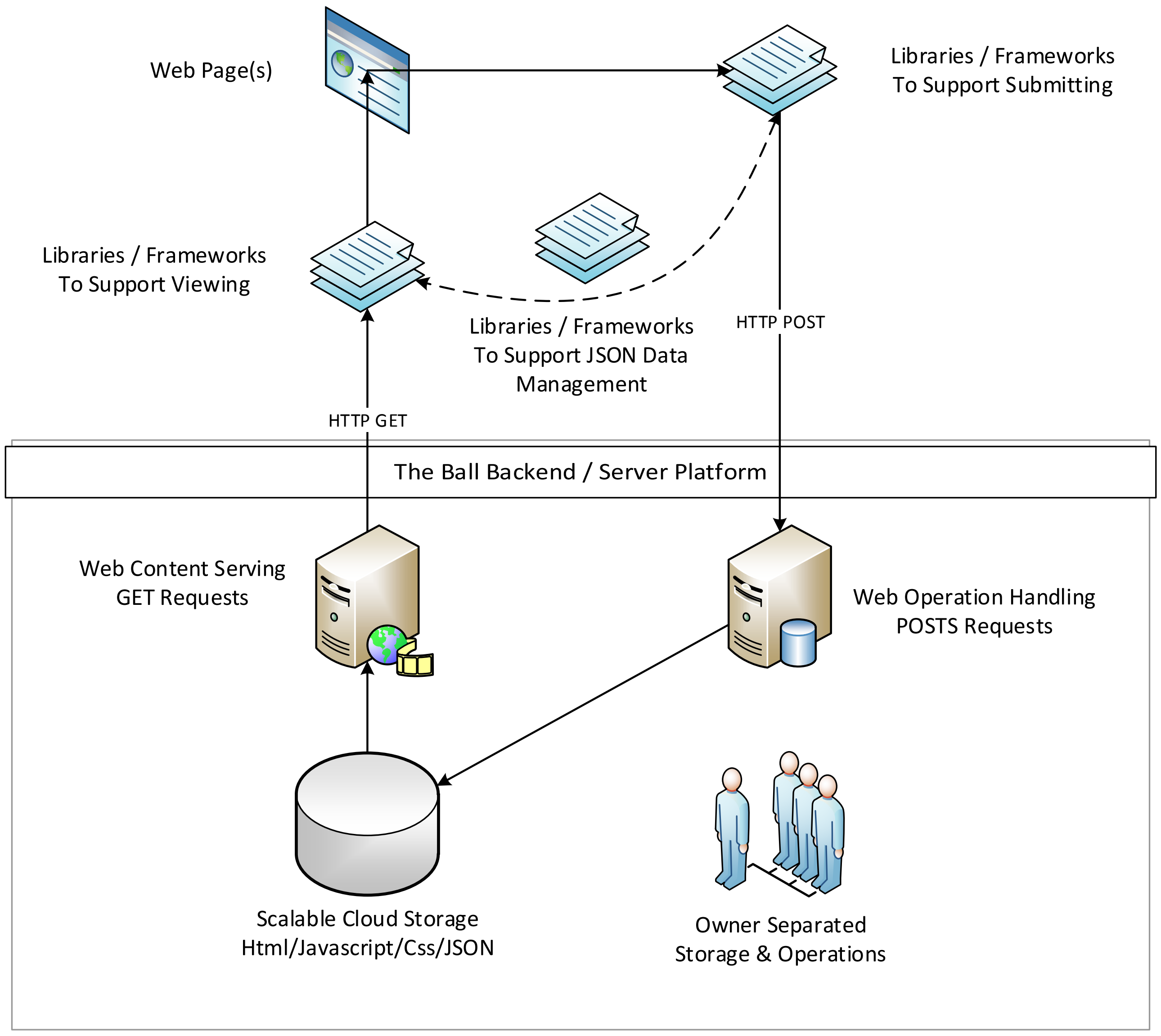
Building a Web Template on the OIP/the Ball

Architectural Overview



Web Page(s)
Libraries / Frameworks To Support Submitting
Libraries / Frameworks To Support Viewing
Libraries / Frameworks To Support JSON Data Management
HTTP POST
HTTP GET
The Ball Backend / Server Platform
Web Content Serving GET Requests
Web Operation Handling POSTS Requests
Scalable Cloud Storage Html/Javascript/Css/JSON
Owner Separated Storage & Operations

Asyncronous Operations and JSON

The Ball platform accepts normal html form submissions and JSON content (= ajax) submissions. There is assisting library (currently named as oiplib1.0) that provides the following methods for operations.

Object modifications

- CreateObject(domainName:string, objectName:string, dataContents:any)
- SaveObject(objectID:string, objectETag:string, dataContents:any)
- DeleteObject(objectID:string)

Operation Execution

- ExecuteOperationWithForm(operationName:string, operationParameters:any)
- ExecuteOperationWithAjax(operationFullName:string, contentObject:any)

On constructor of OperationManager it injects IFrame on bottom of the page to support asynchronous submissions – the IFrame is used as a target for form submissions.

The fetching is simply getting JSON objects from storage.

Template Constructs

The Ball architecture is not serving dynamic content in traditional means "dynamic"; it stores all its dynamic content as JSONs in scalable cloud storage. For web templates this means that all the back-end requests (even those that appear to be dynamic) are simply fetching JSON files from known locations.

Example for Dynamic User Performed Search

Example for completely dynamic request is user given query performed search against content. The Ball will count unique hash value for the search parameters and respond to the query with that hash-value named JSON name. The client will be checking for the result of that JSON or its ID to appear in status updates and display that to the user. In case such a search was previously performed against the same content, the JSON is ready to be served.

More traditional dynamic-alike queries such as "display latest 10 news on this topic" in this architecture turn out to be pre-rendering JSON for that result with specific/dedicated expiration time. This serves as a solid base for massive user base optimized queries where the end-result JSON is simply shared with every request who is looking for that data.

Template / Content Functionality and Caching

All the content in group is served as if it would appear from normal filesystem. This includes all the template artifacts and JSON files. By default the Ball is not assuming any lifecycle on any of the content, thus it will respond to caching settings as "private" and expiration time as "0 seconds". It will also include ETag and Last-Modified header information, so that browsers can send them with follow-up requests. The Ball will check on incoming requests for those headers and result in "304 Not modified" response for unmodified content.

For advanced performance usage and resource optimization (in case of millions of users) the Ball behavior is easy to adjust to have some kind of caching expiration time on html-template content and still have proper functionality on JSON data that must not be cached without verifying if it has been changed.

Main Template Additions

Main and default template for groups is named as "categoriesandcontent". When new group is created or platform general management and content management side is maintained, this is the template that is being modified and copied over to every group. The location for this main template is in template repository under following: OIPTemplates/UI/categoriesandcontent.

The modifications of that template structure is designed to apply to every group in the running the Ball instance.

Example

Implementing new admin side template such as “controlpanel” will consist of following:

1. Introducing new html file under the “html” folder in categoriesandcontent
2. Placing all the resources (css, js, images and so forth) under assets/controlpanel/ folder
3. Plugging in the template internal data structures to the Ball instance/group’s common JSON datamodel and files of the reference content
4. Modifying ajax actions (gets and posts) to map between the datamodels
5. Testing the template through WebStorm built-in server so that it reads the data properly from reference JSON

Independent Group Template

In addition to main template modifications that are instance-wide, groups can also have custom UI templates. Such templates are uploaded as independent ZIP-packages through group identity page. They follow the same relative path structure as do the main template alterations.