Medical Image Identification Index

MU BIEN 4998

Team C54

Kyle Chang, Alexander Dums, Peter Moras

5/3/2019

# **Table of Contents**

[**Table of Contents**](#_63rog4sntg6i) **1**

[**Table of Figures**](#_f6lz2xblwrqt) **3**

[**Table of Tables**](#_aeiy9z1a2b6k) **4**

[**1. Product Description**](#_f2mqo5aqw178) **5**

[**2. Real World Context and Interoperability**](#_ssmuovwabnr4) **5**

[2.1 Real World Use Cases](#_hdhz3kbu986e) 5

[2.1.1 Actors](#_h0gd07a9dh63) 5

[2.1.2 Clinical Scenarios](#_v38ii2u0pmc1) 5

[2.1.3 Flow Charts](#_1s7u56hqceg2) 6

[**3. Data Model**](#_99cb46cij1fe) **7**

[3.1 International Standards](#_fvpr81nbis0g) 7

[3.2 Data Entity Relationship](#_4fg24gxlqkq2) 7

[**4. Transactions and APIs**](#_nzk254l7qu9a) **8**

[4.1 Web Browser](#_8iwve4bk2epy) 8

[4.1.1 Transaction Description](#_c8efac1r818s) 8

[4.1.2 Transaction International Standard](#_cir2mkclskpl) 8

[4.1.3 Transaction Diagram](#_ltwyezrygul2) 8

[4.1.4 Transaction Transport](#_yxwr3k9j0o1) 8

[4.1.5 Transaction Data Semantics](#_tk0u8hcslfg0) 8

[4.1.6 Transaction Instantiation](#_2dhta4s93qkg) 8

[4.1.7 Transaction Response](#_qivuxohdbffl) 8

[4.2 Upload File](#_32jk3hrun9it) 9

[4.2.1 Transaction Description](#_4795ih8exk1w) 9

[4.2.2 Transaction International Standard](#_39ci5898cuc) 9

[4.2.3 Transaction Diagram](#_bqoxdmxfi19c) 9

[4.2.4 Transaction Transport](#_58nvc83rl04g) 9

[4.2.5 Transaction Data Semantics](#_ca2owf7j53rv) 9

[4.2.6 Transaction Instantiation](#_tqcqc15hehgp) 9

[4.2.7 Transaction Response](#_cxfm11qsr639) 9

[**5. Software Design**](#_2066qqp9bvq6) **10**

[5.1 Licensing](#_81470e23fc73) 10

[5.1.1 Licensing Agreement](#_usuec72oh3jx) 10

[5.1.2 Licensing Dependencies](#_lyxmw61sx7x7) 10

[5.1 Source Code Repository](#_4p4pagdufe4) 10

[5.3 Software Languages](#_xnrlvd5iyp0c) 10

[5.4 Software Development Methodology](#_p7y5d7q42twz) 10

[5.5 Software System Attributes](#_c96xgq1dpiju) 11

[5.5.1 Scalability](#_abcpijnh7cj9) 11

[5.5.2 Serviceability](#_5abw3y7k2lnh) 11

[5.5.3 Monitoring](#_opklvgs7p54a) 11

[5.5.4 Upgradability](#_rqd9d14f8cc5) 11

[5.5.5 Uptime Goals](#_vvdv80mphj7x) 12

[5.5.6 Security](#_3gm83pdryxj6) 12

[5.7 Primary Software Modules](#_e7rhquq5i9u7) 12

[5.8 Software Flow](#_hrb5b6m8ecdv) 13

[5.9 Software State Transition Diagram](#_yl608id1kkn7) 16

[5.10 Software Architecture](#_326am0s52oov) 17

[5.11 Data Structures](#_qnajg797v0de) 17

[**6. Risk Assessment**](#_80jnbkiugxeb) **22**

[**7. Software Verification and Validation**](#_y2aj3fm3anrv) **23**

[**8. Recommended Design Changes**](#_wb0fepet7h7h) **23**

# 

# 

# **T**[**able**](#_2l4aqzm7mzfm) **of Figures**

Figure 2.1.3 - 1: Flow chart following a submission process 6

Figure 2.1.3 - 2: Flow chart following a searcher’s process 6

Figure 2.1.3 - 3: Flow chart following a site administrator’s approval process 6

Figure 3.2 - 1: Entity Relationship Diagram 7

Figure 4.1.3 - 1: Web browser transaction diagram 8

Figure 4.2.3 - 1: File transaction diagram 9

Figure 5.8 - 1: Login 13

Figure 5.8 - 2: Registration 13

Figure 5.8 - 3: Submitting Abstract 14

Figure 5.8 - 4: Accept Abstract 15

Figure 5.8 - 5: Upload Data set 15

Figure 5.8 - 6: Search 16

Figure 5.9 - 1: Abstract state transition diagram 16

Figure 5.10 - 1: Software Architecture Diagram 17

# **Table of Tables**

Table 2.1 - 1: Actor Definitions 5

Table 5.11 - 1: Abstracts Collection Data Structure in MongoDB 19

Table 5.11 - 2: AspNetUsers Collection Data Structure in MongoDB 21

Table 5.11 - 3: Notifications Collection Data Structure in MongoDB 22

Table 5.11 - 4: Repository 23

Table 6 - 1: Risk Assessment Table 23

# **1. Product Description**

The Medical Image Identification Index gives researchers access to high quality, high volume anonymized medical image data for the primary purpose of training AI systems.

# **2. Real World Context and Interoperability**

## **2.1 Real World Use Cases**

### 2.1.1 Actors

Table 2.1 - 1: Actor Definitions

|  |  |  |
| --- | --- | --- |
| **Actor Name** | **Definition** | **Examples** |
| Submitters | Those that have unique data that can be submitted | Radiologists, Universities, Hospitals |
| Searchers | Those looking for data | AI Researchers, Radiologists |
| Site Administrator | Those that curate the data being taken into the website and the permissions of those that view the data | ACR staff |

### 2.1.2 Clinical Scenarios

Larry is a radiologist that has found an interesting and unique study that he wishes to upload to the Medical Image Identification Index. He already has an account with the site, so he submits an abstract of his data where he confirms that he has de-identified all the images using DicomCleaner. After waiting a few days for the site administrators to review his abstract and either approve or deny his upload, he receives an email giving him the go ahead to upload his data. Returning to the site, he then uploads his data in accordance to the guidelines posted on the submission page.

Steve is an AI researcher searching for images of left lung nodules in middle aged males. Using the search bar provided on the Medical Image Identification Index website, he searches for the data he wants using the keyword, “lungs”, which displays all instances of images that were labelled as lungs. Due to the many instance of lung data sets, he decides to further refine his search by adding the keyword “males”, which narrows down his search even more. Finding a suitable data set that he wants for his algorithm, he clicks the download button, which redirects to the data set’s provided method of download.

Kristy is a site administrator who oversees the Medical Image Identification Index. Looking at her notifications tab, she sees that she has an abstract that is pending approval. Selecting the abstract, she reads over its content. Upon finding its contents suitable for upload, she decides to approve it and send an automated message to the account that submitted the abstract. If the abstract was not approved an email stating that it was not approved would be sent instead.

### 2.1.3 Flow Charts

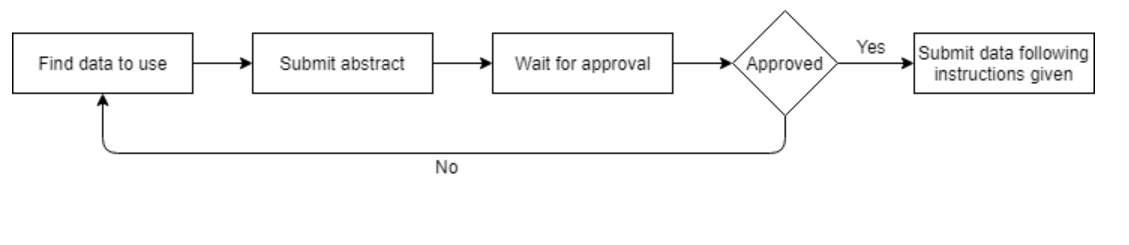


Figure 2.1.3 - 1: Flow chart following a submission process

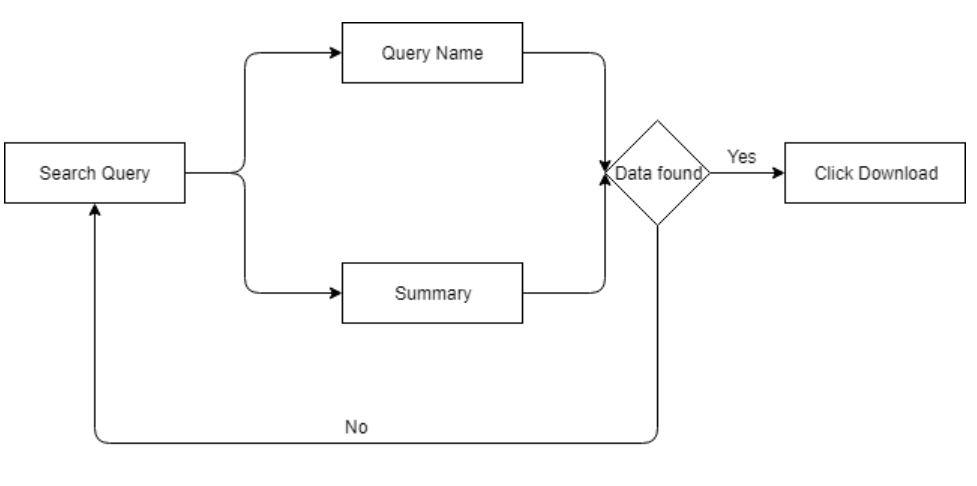


Figure 2.1.3 - 2: Flow chart following a searcher’s process

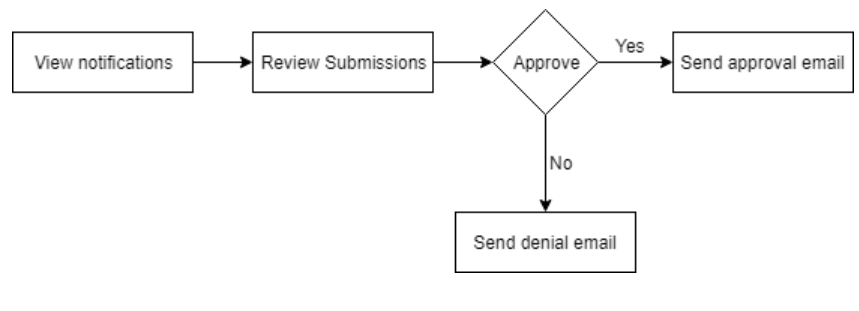


Figure 2.1.3 - 3: Flow chart following a site administrator’s approval process

# **3. Data Model**

## **3.1 International Standards**

* DICOM: <http://dicom.nema.org/medical/dicom/current/output/html/part03.html#table_C.7-1>

## **3.2 Data Entity Relationship**

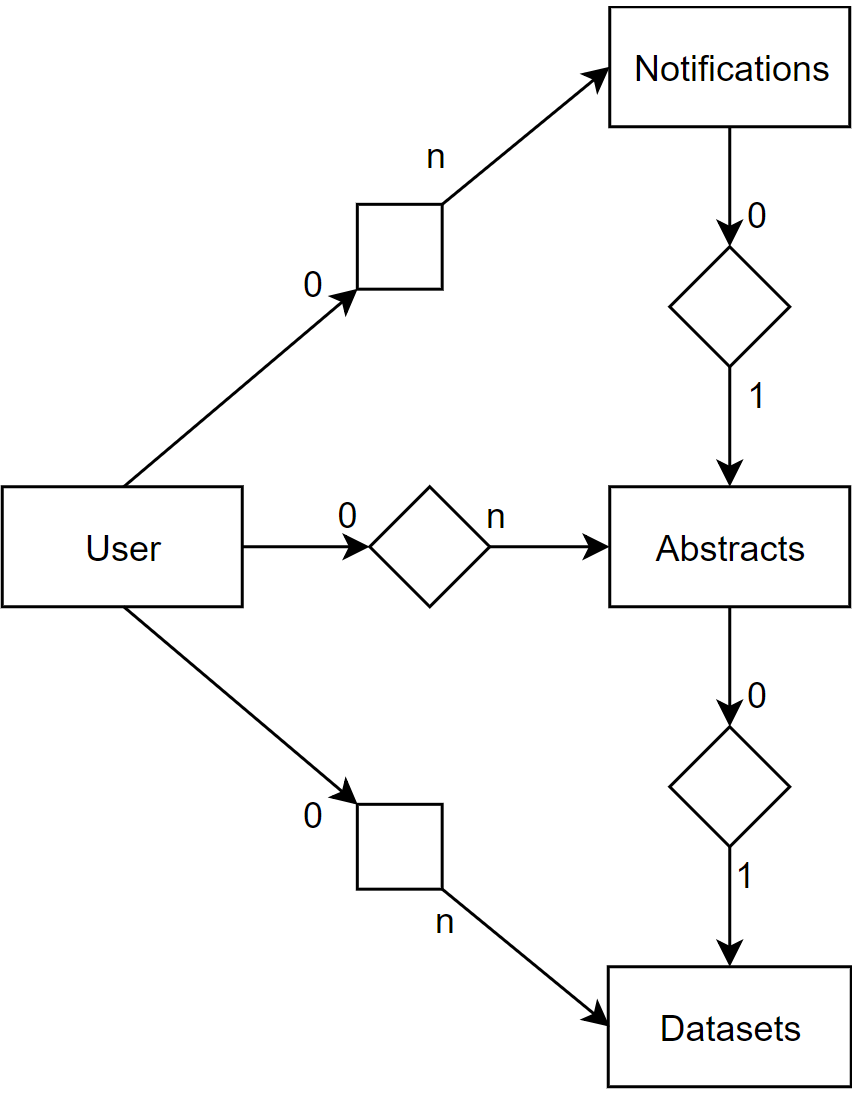


Figure 3.2 - 1: Data Entity Relationship

# **4. Transactions and APIs**

## **4.1 Web Browser**

### 4.1.1 Transaction Description

* The website provides standard HTTP requests for website data that interacts with the server through HTML pages.

### 4.1.2 Transaction International Standard

* HTML5: <https://www.w3.org/TR/html5/>
* CSS3: <https://www.w3.org/Style/CSS/Overview.en.html>

### 4.1.3 Transaction Diagram

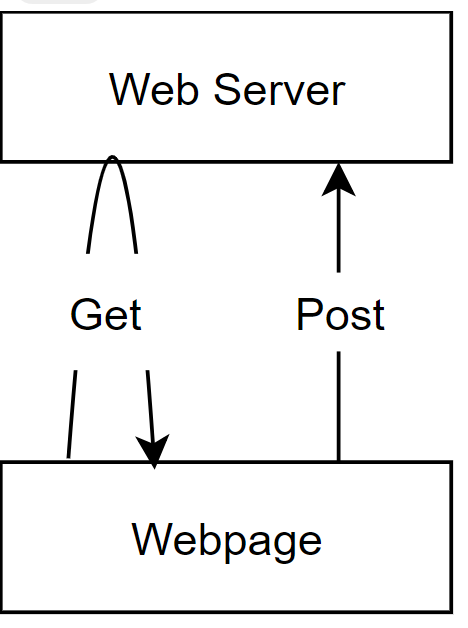


Figure 4.1.3 - 1: Web browser transaction diagram

### 4.1.4 Transaction Transport

* HTTP/2

### 4.1.5 Transaction Data Semantics

* HTML and JSON data is requested and sent using standard HTTP protocols.

### 4.1.6 Transaction Instantiation

* https://websitename.com/request

### 4.1.7 Transaction Response

* Action results in response to HTTP posts, resulting in redirects.
* JSON data in response to AJAX requests

## **4.2 Upload File**

### 4.2.1 Transaction Description

* Zip file is parsed to send individual DICOM header files to server. DICOM headers are stripped of image data before transport.

### 4.2.2 Transaction International Standard

* ISO/IEC 21320-1:2015 : <https://www.iso.org/standard/60101.html>
* DICOM 2019b : <https://www.dicomstandard.org/current/>

### 4.2.3 Transaction Diagram

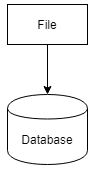
**

Figure 4.2.3 - 1: File transaction diagram

### 4.2.4 Transaction Transport

* HTTP/2

### 4.2.5 Transaction Data Semantics

* File type of ZIP is required to initiate process. ZIP folder is traversed through to find all DICOM or label recognized files. Other files types are ignored. Recognized files are extracted from ZIP and sent individually to web server to be stored in database Datasets. DICOM files have their images cut out before being sent, leaving only header data.

### 4.2.6 Transaction Instantiation

* Transaction is initiated with HTML input file type. Instantiation begins after valid zip file is submitted.

### 4.2.7 Transaction Response

* Displays total number of files needed to be sent along with the number of files sent so far.

# **5. Software Design**

## **5.1 Licensing**

### 5.1.1 Licensing Agreement

* Creative Commons Attribution 4.0 International
* <https://creativecommons.org/licenses/by/4.0/>
* The Maxiomtech MongoDB ASP.NET Identity provider v.1.0.7 was used to make the .NET Identity framework (which is configured for SQL databases) compatible with MongoDB: <https://github.com/maxiomtech/MongoDB.AspNet.Identity>

### 5.1.2 Licensing Dependencies

* MongoDB
* Microsoft Azure
* Maxiomtech MongoDB Identity provider
* Fellow-Oak DICOM parser

## **5.1 Source Code Repository**

* <https://github.com/senior-design-team-c54/Medical-Image-Identification-Index>

## **5.3 Software Languages**

* .NET framework 4.6: Architecture used by C#, provides backbone structure for back-end server.
* C# 7.0: Used for Programming the back-end web server that sends web pages and interfaces with the database.
* HTML5: Required for standard web page usage.
* CSS3: Required for standard web page usage.
* JavaScript: Provides extra frontend functionality for web pages. Primarily used to separate DICOM files for individual sending.

## **5.4 Software Development Methodology**

A hybrid Agile-Waterfall methodology was used. While seemingly conflicting, both were used throughout the design process. During the first half of the project, much of the project was planned out using many sequence diagrams, flow charts, and a Gantt chart. During the latter half of the project an Agile approach was taken, where many incremental changes were made.

## **5.5 Software System Attributes**

### 5.5.1 Scalability

* Scaling of users and data is based on database storage and server capacity, both of which are infinitely scalable. As the number of users increases, servers will inevitably have to be scaled both vertically and horizontally to accommodate the increased load. Similarly, data storage will need to be expanded to handle vast quantities of image data, likely well before scaling is required based on the number of users. This is done at the discretion of the system owners/administrators and is abstracted away from this application. In theory, the software itself will be able to handle an arbitrary number of datasets and users, but the hardware will require upgrades.
* The greatest bottleneck in scalability is the large amount of memory required to store huge sets of DICOM images.

### 5.5.2 Serviceability

To maintain the system, a common format error log would be used to monitor issues as they occur. This file would be available to administrators monitoring the system where they could then perform the necessary fixes in the areas specified by said file. A local copy of the server can be tested using Visual Studio and the ASP.NET framework in conjunction with the IIS express program.

### 5.5.3 Monitoring

IHE SOLE and ATNA provide system monitoring. Audit Trail and Node Authentication provides user accountability, while a Standardized Operational Log of Events would provide a logging system for user actions. If there were to be any unusual events, alerts to site administrators could be sent out as soon as the event occurred. Events that could be reported include, improper access of data by users, data outages, and server outages. Automatic monitoring is also provided by Google, Amazon, and Microsoft servers and databases, with the owners having comprehensive data about performance, usage, and error logs.

### 5.5.4 Upgradability

* Currently, only a single server is used to maintain online presence, and upgrading would temporarily force the server offline.
* For future upgradability, redundant servers would be used enable fixes without shutting down the entire system. By upgrading one redundant server at a time and hosting users there, ideally no downtime would exist when implementing changes.

### 5.5.5 Uptime Goals

* Ideally, uptime would be 24 x 7 x 365 using failover to provide continuous service even if one or more servers failed.

### 5.5.6 Security

* HIPAA (De-identification required)
* Database uses an IP whitelist to only allow certain machines to access it (even if they have the proper login credentials). Ideally, only the server(s) running the application and perhaps a database administrator would have login capability and whitelisted IP’s to provide an added layer of security. Furthermore, MongoDB automatically encrypts all stored data.
* User authentication and authorization is handled using Microsoft’s Identity framework in conjunction with MaxiomTech’s MongoDB provider for the Identity framework.
* All database access is handled by a single class in the application to minimize the exposure area in which leaks can occur.
* Do not allow people to access critical data without proper credentials
* Secure on the fly vs Secure on the disk

While all data input to the system is to be de-identified by the submitter, there is no guarantee that they have done so with 100% accuracy, as a result, the data on the server must be kept secure. As all users that wish to access data must have an account, their permissions will be set at the administrators discretion. The monitoring system in place should provide an alert in the event of unusual activity.

**5.6 Detailed Software Requirements**

*Gantt chart with listed requirements and time tables linked below.*

[*https://drive.google.com/file/d/18eO50pWd9kgIA1AMRAZSXRFdBOGqP6a-/view?usp=sharing*](https://drive.google.com/file/d/18eO50pWd9kgIA1AMRAZSXRFdBOGqP6a-/view?usp=sharing)

## **5.7 Primary Software Modules**

* User - Accesses the web server to read information from the database.
* Web Server - Intermediary to the database that serves web pages to users
  + Login Page - Input login information to become recognized user
  + Register Page - Provide valid registration information to obtain User account
  + Search Page - Lists matching Queried Datasets
  + Upload Page - Post abstract data to server for review.
  + DataSet Upload Page - Post DICOM dataset data and link it with abstract
* Database - Stores Dataset, Abstract, and User data for web server use.

## **5.8 Software Flow**

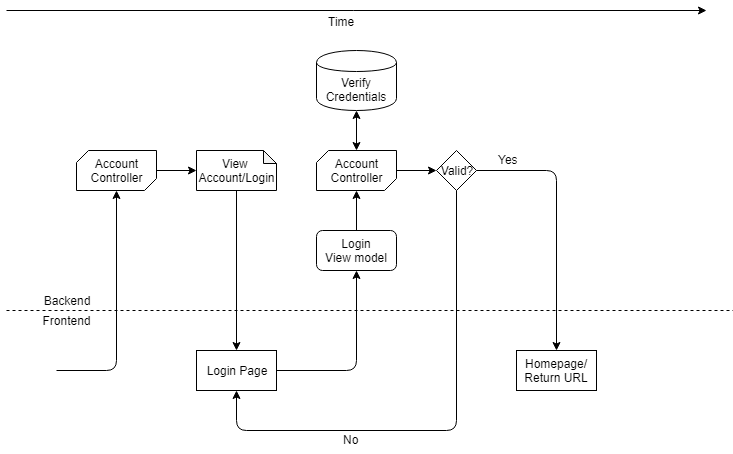


Figure 5.8 - 1: Login

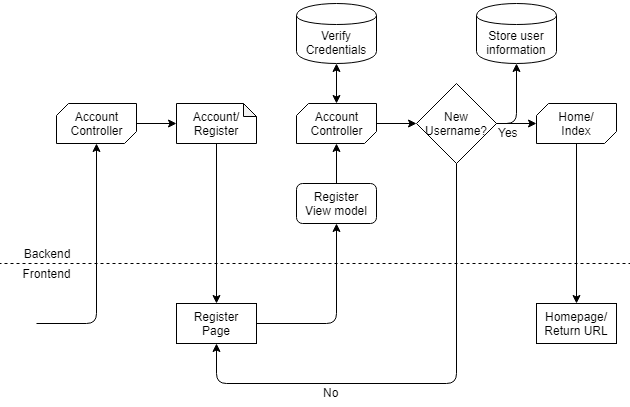


Figure 5.8 - 2: Registration

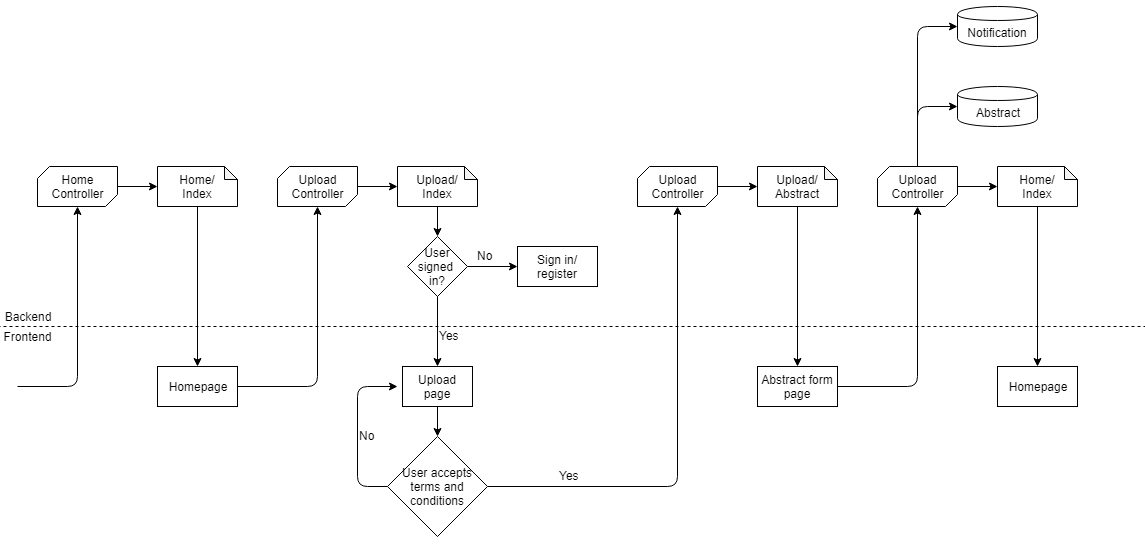


Figure 5.8 - 3: Submitting abstract

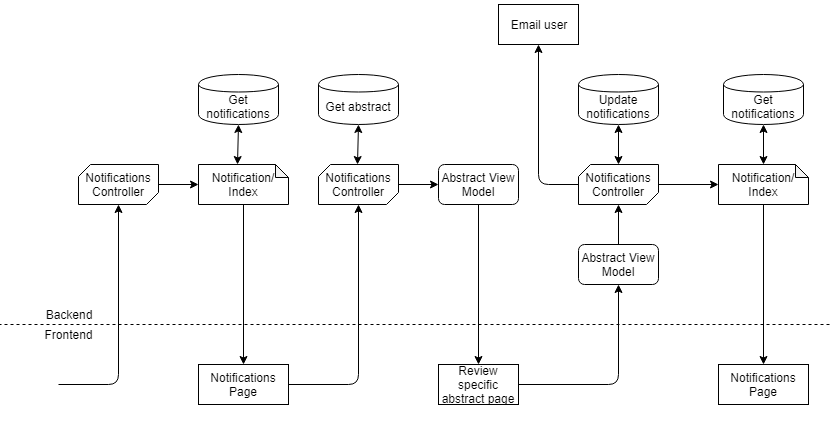


Figure 5.8 - 4: Accept abstract

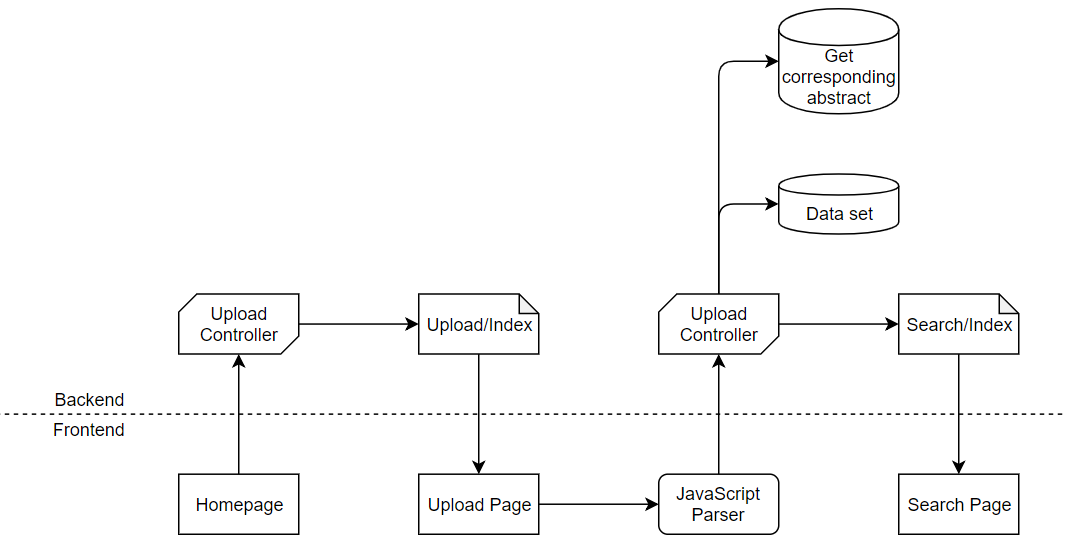


Figure 5.8 - 5: Upload data set

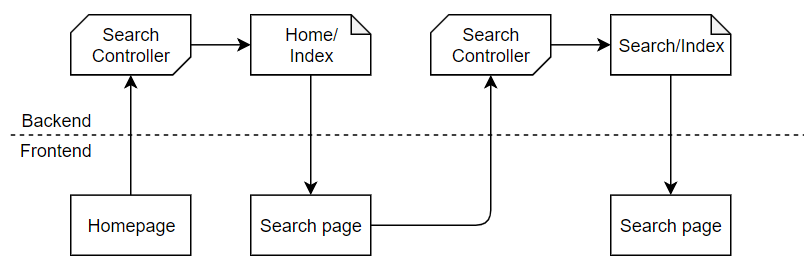


Figure 5.8 - 6: Search

## **5.9 Software State Transition Diagram**

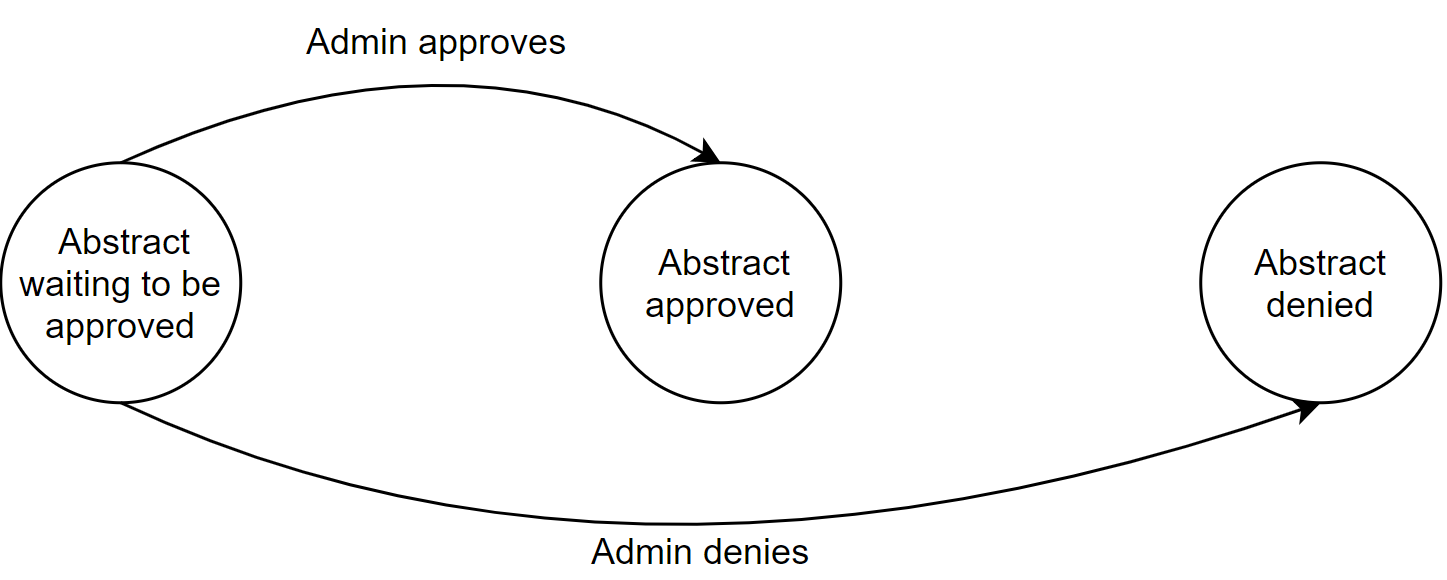


Figure 5.9 - 1: Abstract state transition diagram

## **5.10 Software Architecture**

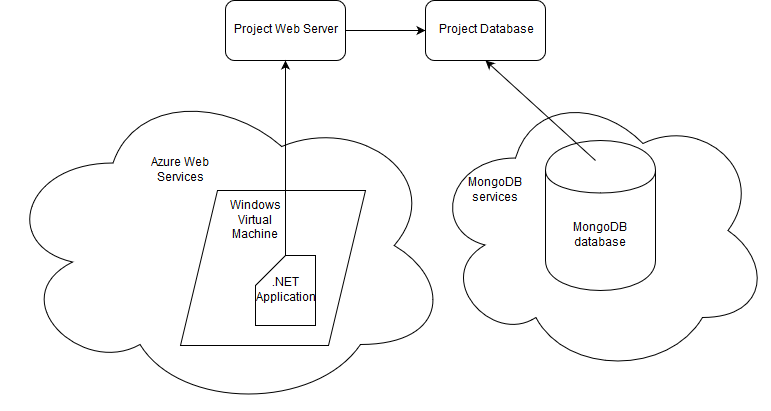
**

Figure 5.10 - 1: Software Architecture Diagram

## **5.11 Data Structures**

* Database is used for:
  + User login
  + Abstracts
  + Search
  + Administrative review

The MongoDB (schema-less) database backing the web application is currently configured with four collections: *Abstracts*, *Notifications, AspNetUsers,* and *Repository*. The *Abstracts* collection contains documents (formatted essentially as JSON) which each contain the series of fields that are mapped to the user inputs in the abstract form page. *Notifications* contains notifications mapped to users and abstracts. *AspNetUsers* contains user information such as login credentials. *Repository* stores dataset metadata.

Table 5.11 - 1: Abstracts Collection Data Structure in MongoDB

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Data Type** | **Description** |
| \_id | String or ObjectId | GUID for this particular document in the Mongo database |
| UserName | String | Email/username for the user who submitted the abstract |
| DatasetTitle | String | User-provided title for dataset (min. 10 characters) |
| Authors | String | User-provided list of authors of dataset |
| Source | String | User-provided institutional source of data |
| Url | String | User-provided URL at which dataset can be accessed |
| Rehost | Boolean | True if user specified that they want a particular dataset to be rehosted, false otherwise |
| PublicAccess | Boolean | True if user specified that a particular dataset should not be publicly searchable, false otherwise |
| Content | String | Additional information provided by user about dataset |
| Summary | String | Summary of relevant information about dataset provided by user |
| Approved | Boolean | True if admin has approved abstract via review process, false otherwise |
| Rationale | String | Feedback provided by admin during the abstract review process |
| Reviewed | Boolean | True if abstract has been reviewed by an admin, false otherwise |
| ReviewedBy | String | Email/username of admin who reviewed abstract |
| DateReviewed | DateTime | Timestamp for when admin reviewed abstract (approved or denied) |
| DateGenerated | DateTime | Timestamp for when user originally submitted abstract |
| DatasetId | String | GUID for Dataset document associated with this abstract (null until after user has submitted their data following abstract approval) |
| IsAnonymized | Boolean | True if user has certified that their data has been scrubbed of PHI, false otherwise |
| HowAnonymized | String | User-provided description of how data was de-identified |
| SameModalityAndManuf | Boolean | True if user specified that all images are from the same device and manufacturer, false otherwise |
| Manufacturer | String | User-provided manufacturer of imaging device used to generate dataset images |
| Modality | String | User-provided modality of imaging device used to generate dataset images |
| HasLabels | Boolean | True if user specified that the data has labels, false otherwise |
| LabelType | String | User-specified type(s) of labels |
| LabelFormat | String | User-specified format that labels are stored in |
| LabelsReviewed | String | User-provided information about whether or not labels have been peer-reviewed, can be have values of “Yes”, “No”, or “Unknown” |
| LabelLevel | String | User-provided level at which data labels are stored, can have values of “Series”, “Study”, or “Unknown” |
| ClinicalIssues | String | User-provided information about the key clinical issues encapsulated in the data |
| Country | String | User-specified country from which data originated |
| USRegion | String | User-specified region in the US from which data originated |
| State | String | User-specified state in the US from which data originated |
| PixelDataShifted | String | User-provided condition of pixel data shifting in the data, can have values of “Yes”, “No”, or “Unknown” |
| HowPixelDataShifted | String | User-provided description of how pixel data was shifted |
| PixelDataSynthesized | String | User-provided condition of pixel data synthesis in the data, can have values of “Yes”, “No”, or “Unknown” |
| HowPixelDataSynthesized | String | User-provided description of how pixel data was synthesized |
| MultipleModalitiesAndManuf | String | User-provided description of the devices and modalities if they specified that there were multiple |
| ModelsAndVersions | String | User-provided description of the scanner models and software versions used |

Table 5.11 - 2: AspNetUsers Collection Data Structure in MongoDB

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Data Type** | **Description** |
| \_id | ObjectId | GUID for this particular document in the Mongo database |
| UserName | String | Username of user, equal to their email address in this configuration |
| PasswordHash | String | Hashed/encrypted password for user |
| SecurityStamp | String | Security stamp used for cookie authentication |
| Roles | Array | List of user’s roles (e.g., “admin”) |
| Claims | Array | List of user’s claims (more granular version of roles, easier to customize for each user) |
| Logins | Array | List of third-party logins used for third-party authentication (e.g., Facebook, Google, etc.) |
| PhoneNumber | String | User phone number |
| Email | String | User email address (same as username in this configuration) |
| EmailConfirmed | Boolean | True if email address has been confirmed, false otherwise |
| LockoutEndDateUtc | DateTime | If lockout is enabled, specifies when the user will be able to login again |
| AccessFailedCount | Integer | Number of failed login attempts associated with this username/email, used for locking user out |
| LockoutEnabled | Boolean | True if user has been locked out due to too many failed access attempts, false otherwise |
| TwoFactorEnabled | Boolean | True if some form of two-factor authentication is supported for this user, false otherwise |

Table 5.11 - 3: Notifications Collection Data Structure in MongoDB

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Data Type** | **Description** |
| \_id | String | GUID for this particular document in the Mongo database |
| UserName | String | Username/email for user who caused notification to be generated (e.g., by submitting an abstract) |
| ForUserType | String | Specifies if notification is for admin review |
| NotificationType | String | Class of notification (e.g., new abstract, reviewed abstract, etc.) for server to cast to when deserializing |
| Content | String | Field for any additional relevant content |
| Attachment | String | GUID for the attached document in the Mongo database |
| AttachmentType | String | Class of attachment so that server knows which collection to look in and how to cast when deserializing the attachment |
| Resolved | Boolean | True if an admin has reviewed the notification, false otherwise |
| ResolvedBy | String | Username/email of admin who reviewed the notification |
| DateResolved | DateTime | Date and time that the admin reviewed/resolved the notification |
| DateGenerated | DateTime | Date and time when the notification was originally created by the server |

Table 5.11 - 4: Repository Collection Data Structure in MongoDB

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Data Type** | **Description** |
| \_id | ObjectId | GUID for this particular document in the Mongo database |
| Availability | Boolean | (deprecated/unused) |
| Publication\_Date: | DateTime | Date and time that dataset was uploaded |
| Name | String | Title of dataset (from abstract) |
| Summary | String | User-provided dataset summary from abstract |
| Authors | String | Author names for dataset (from abstract) |
| Url | String | URL at which dataset can be accessed (from abstract) |
| Tags | Object | List of DICOM tags |
| Label\_Type | String | Type of dataset labels |
| TotalFiles | Integer | Total number of DICOM files parsed |
| MetaData | Object | Abstract associated with this dataset |

# **6. Risk Assessment**

Table 6 - 1: Risk Assessment Table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Potential Issue** | **Probability** | **Severity** | **Risk** | **Mitigation Plan** |
| Cannot find image | 0.05 | 2 | 0.1 |  |
| Download link leads to the wrong image | 0.05 | 7 | 0.7 | Adding Hashes (SHA-256) to verify data |
| Image/Abstract mismatch | 0.01 | 1 | 0.01 |  |
| Private data publically accessible | 0.1 | 3 | 0.3 | Error On user side, give Admins ability to hide public Datasets |
| Data not de-identified | 0.5 | 10 | 5 | Admin hides data and notifies owner of data breach |
| User gains administrative powers | 0.01 | 9 | 0.09 | Limit Admin powers, potentially add more specialized roles |

# **7. Software Verification and Validation**

* Refer to April 10th document (Experimental Verification document)

# **8. Recommended Design Changes**

* Refer to April 10th document
* Additional Changes:
  + Update and delete data
  + Local storage of image data
  + API for normal functionality
  + More searchable fields
  + Additional metadata to upload such as labels either real or artificial