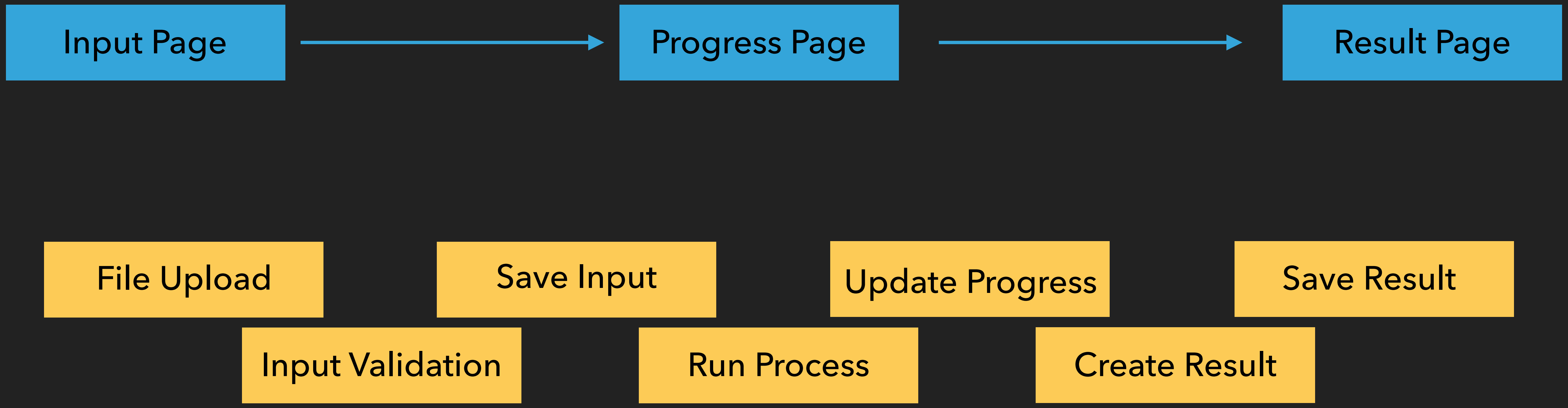
HOW IT WORKS:

JAVA WEB JOB

WHAT DOES THIS JAVA WEB JOB DO?



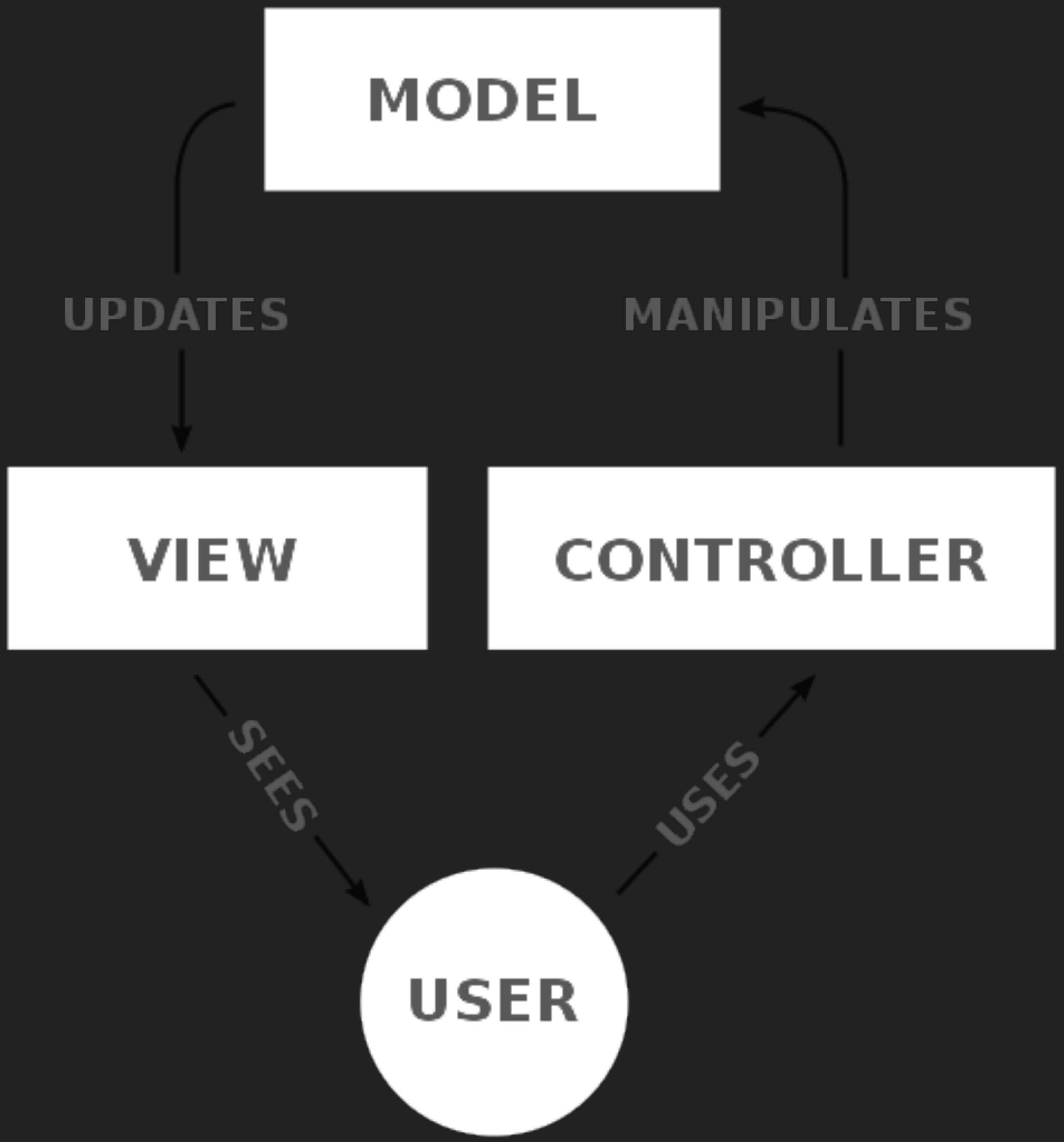
- On the surface, there are three pages; Input, Progress, and Output.
- Beneath that, there are different actions done to produce the result from the entered data.

OVERVIEW

A FURTHER BREAKDOWN

MODEL VIEW CONTROLLER* (MVC) ARCHITECTURAL PATTERN

- View - What the user of the program can see. Displays relevant information in a readable way.
- Model - Stores both data entered by the user and data entered for use in application processes beforehand.
- Controller - Updates the Model with user input and provides the View with data from the Model.



* More specifically, the Model 2 implementation of MVC is used.

MVC IN JAVA WEB JOB

View

HTML, CSS, Javascript, JSP

Controller

Tomcat Web Container,

Servlets, View Controllers

Model

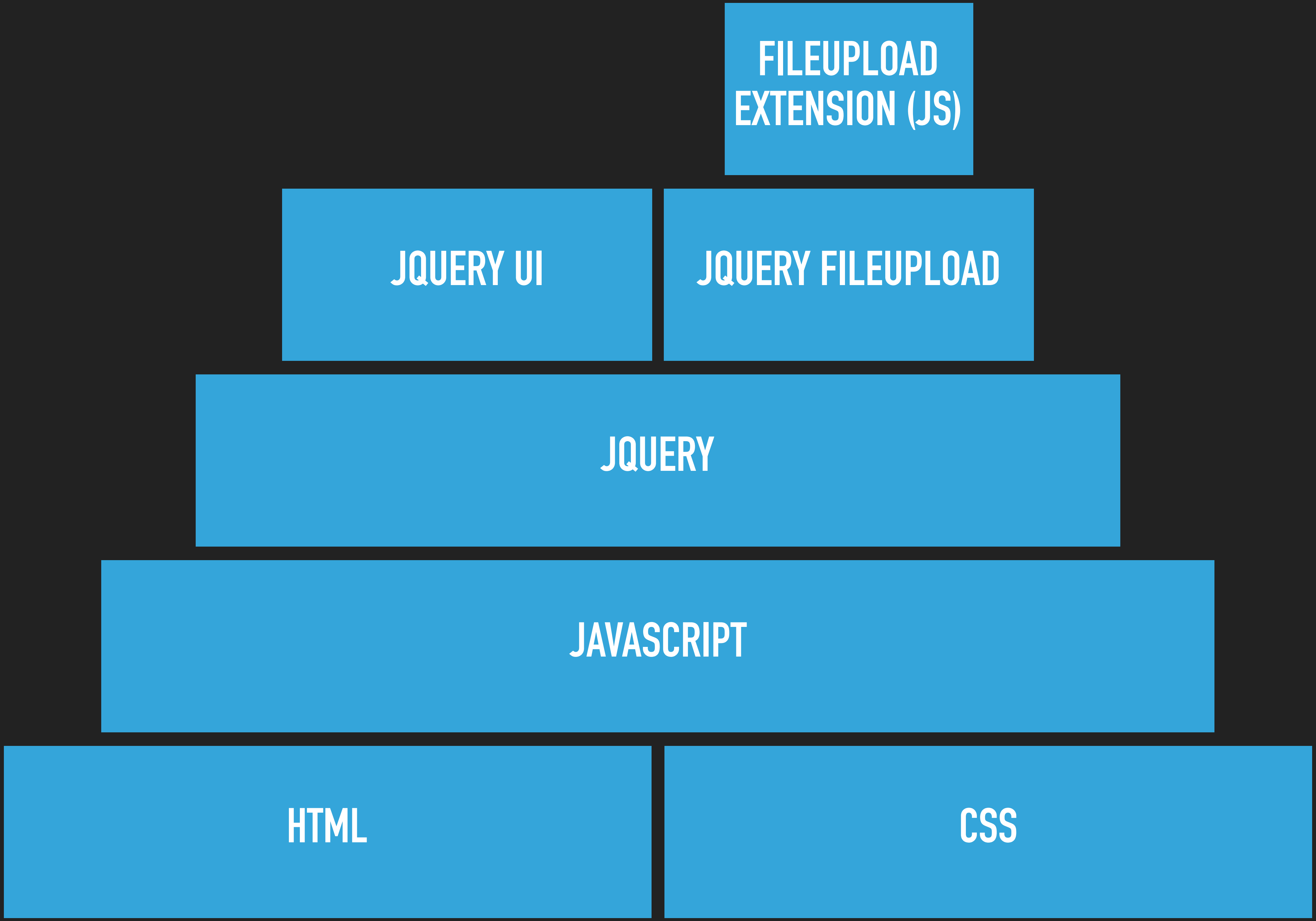
Java Beans, Database

ANOTHER LENS, THE THREE PARTS OF THE PROGRAM

- The MVC Pattern closely corresponds to another way of organizing the functions of this project:
  - Web Browser/ Client - Everything that relates to what the user sees and can interact with.
  - Web Server - Everything that is generated server-side and links the Web Browser to the Database
  - Database - Storage and retrieval of data related to the project.
- MVC and the Browser/Server/Storage categorizations are very closely related.

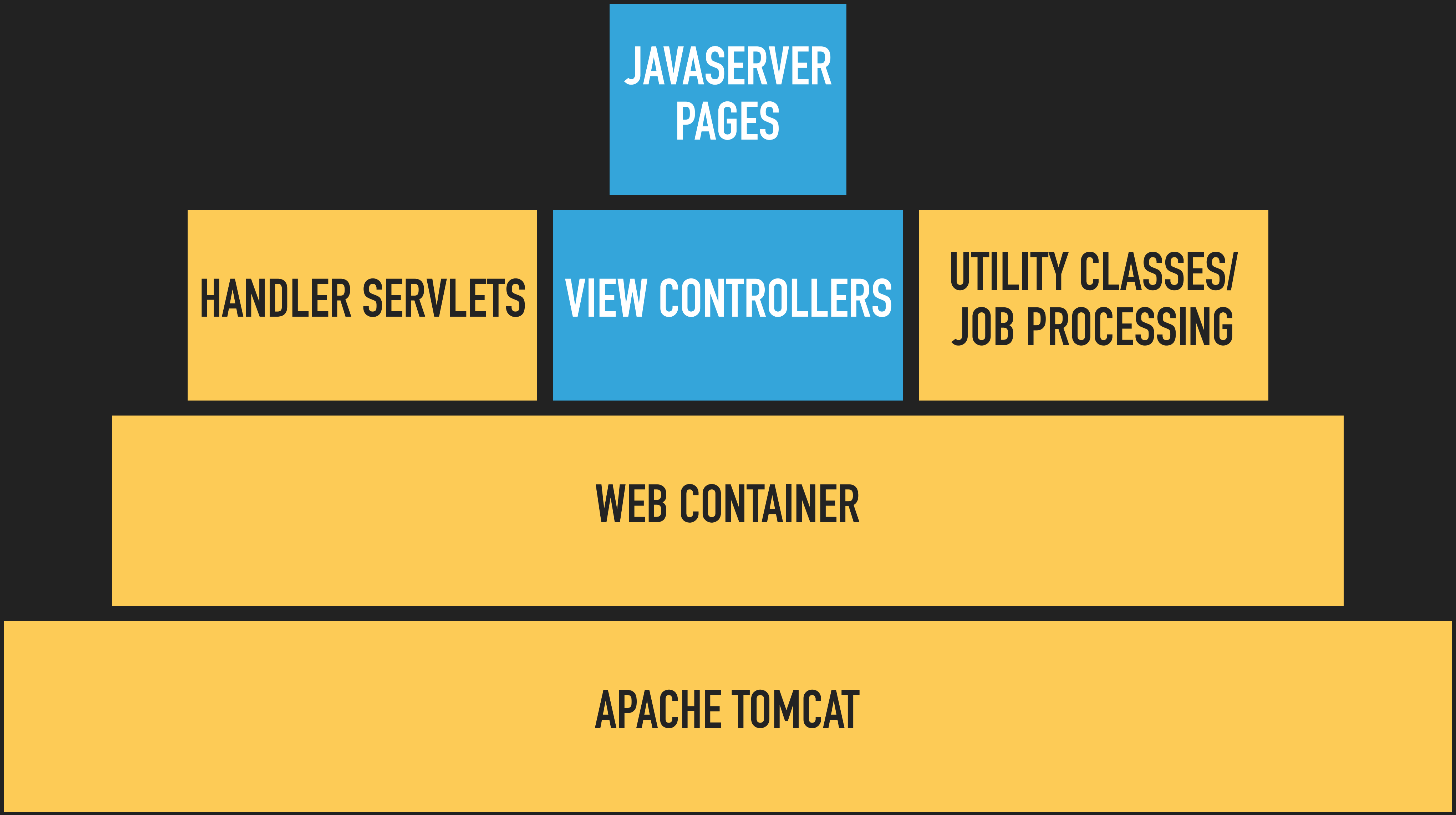
WEB BROWSER/ CLIENT – A MORE DETAILED LOOK

Technology Stack



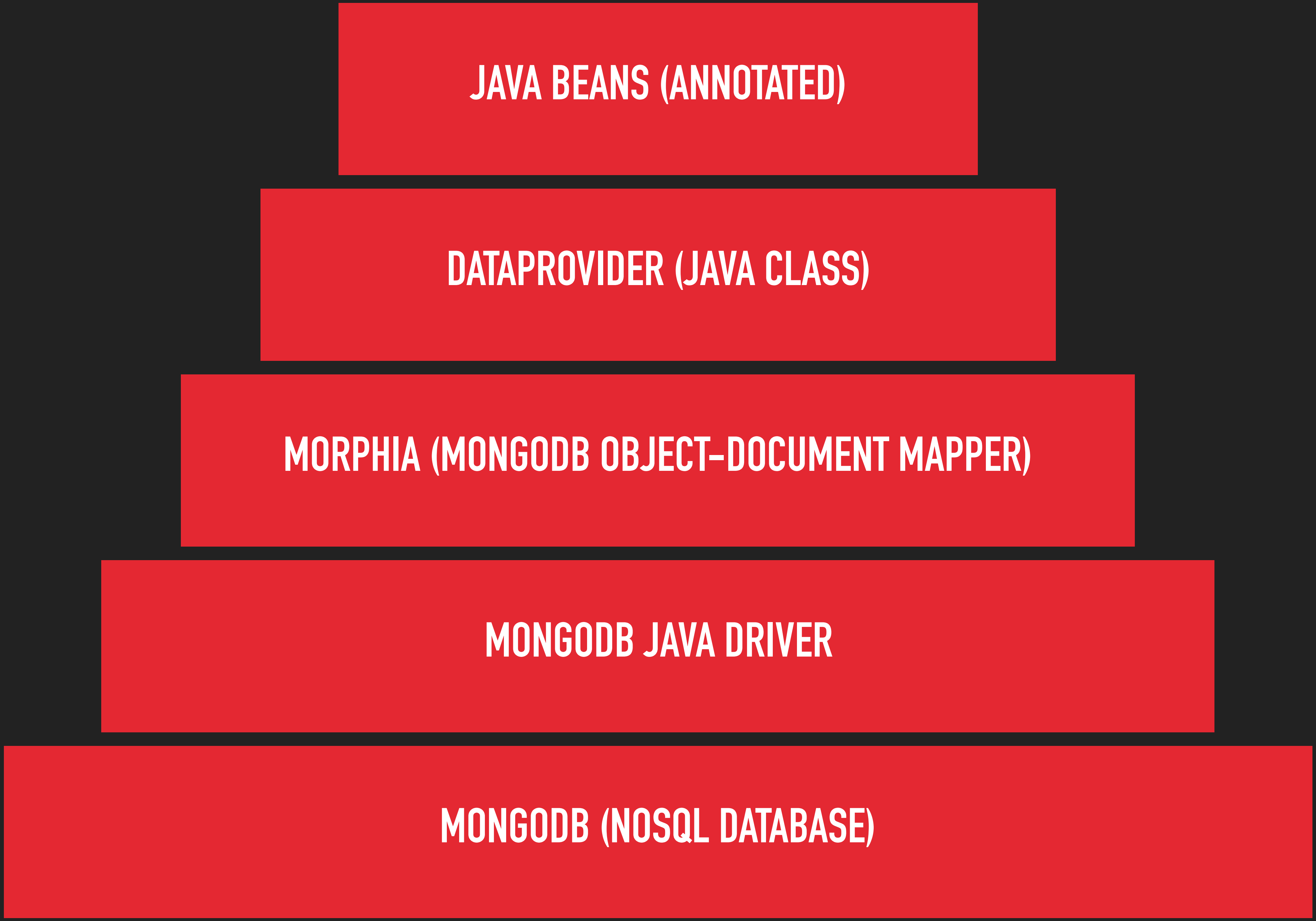
WEB SERVER – A MORE DETAILED LOOK

Technology Stack



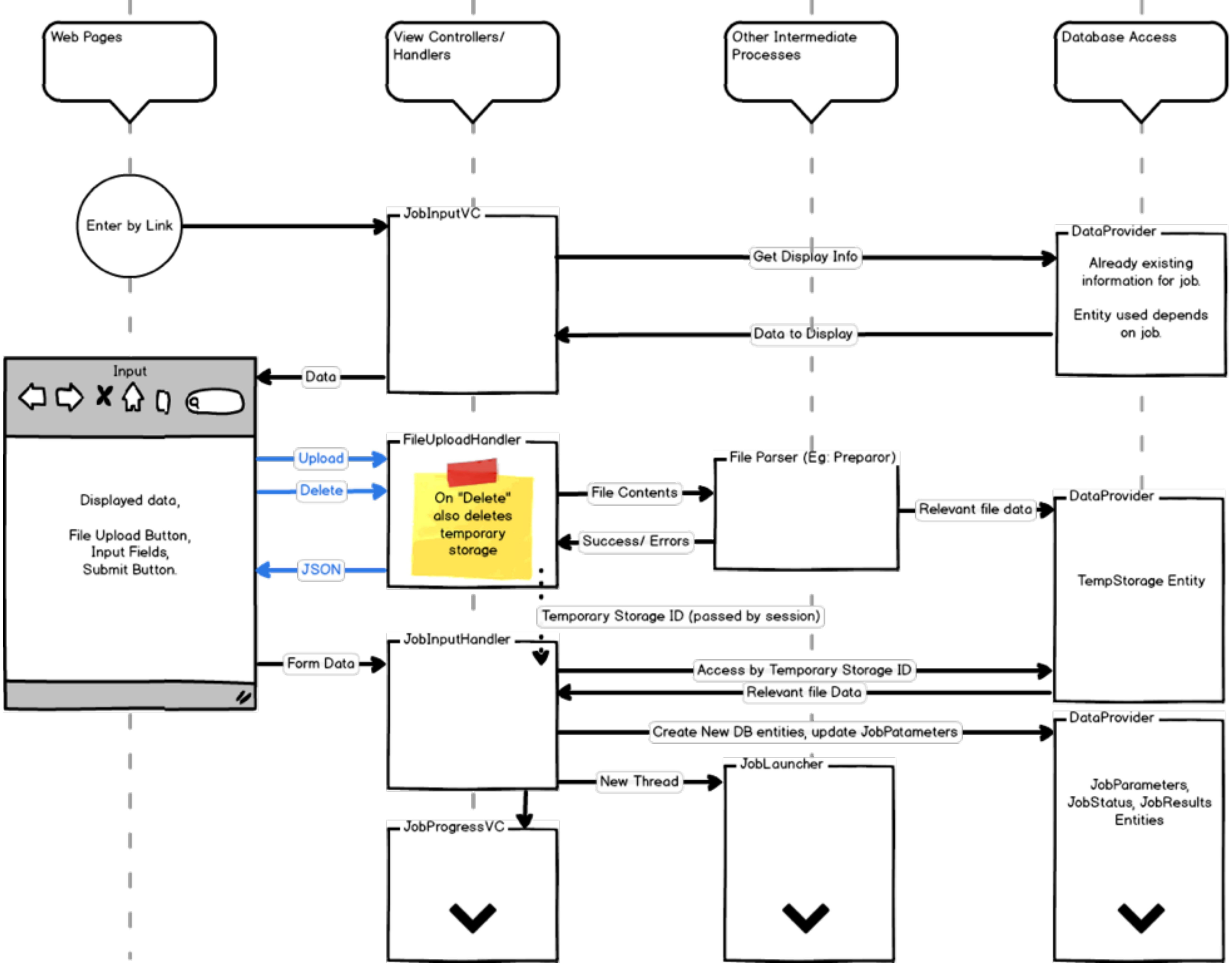
DATABASE – A MORE DETAILED LOOK

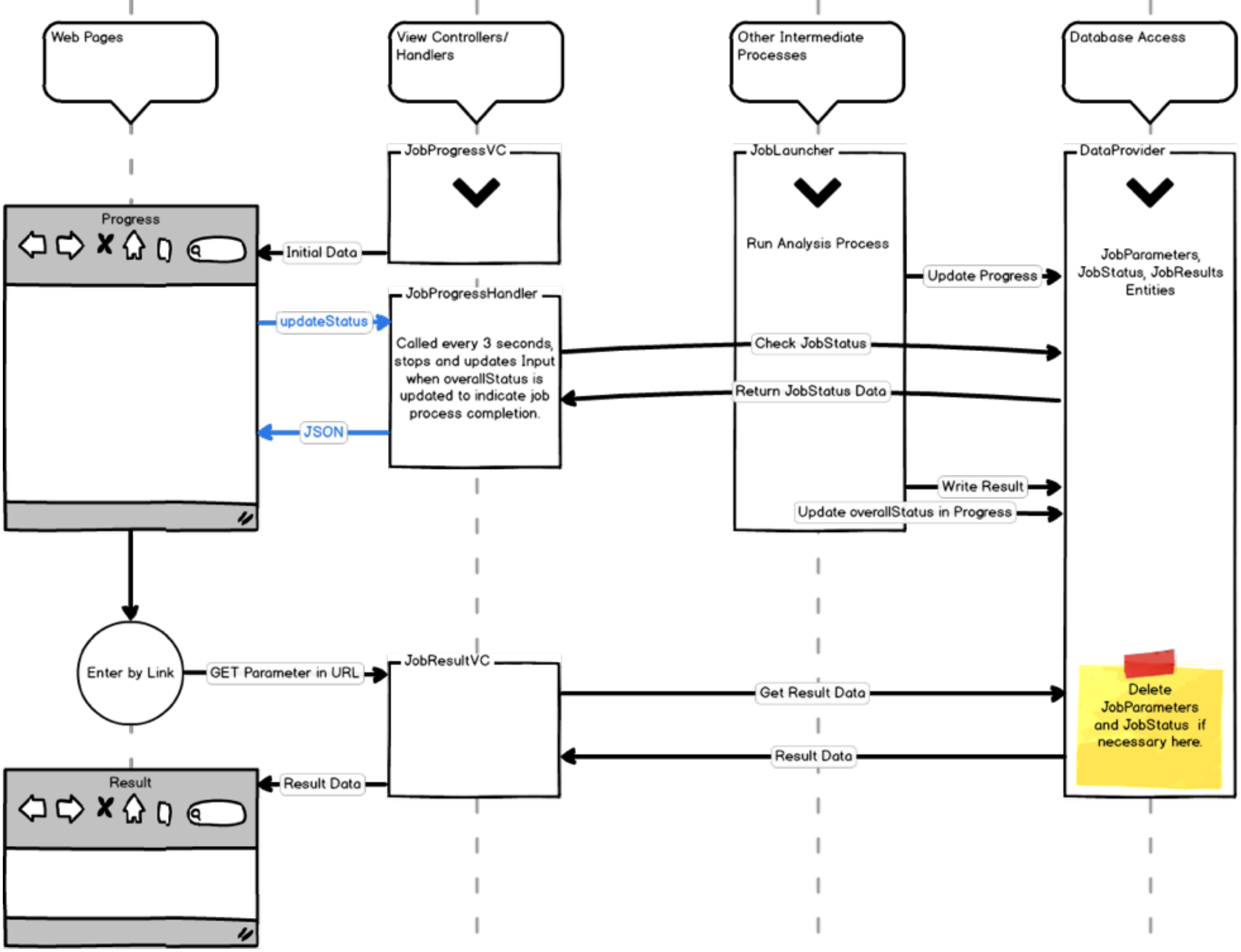
Technology Stack


JAVA BEANS (ANNOTATED)
DATAPROVIDER (JAVA CLASS)
MORPHIA (MONGODB OBJECT-DOCUMENT MAPPER)
MONGODB JAVA DRIVER
MONGODB (NOSQL DATABASE)

DIAGRAMS

INPUT, PROGRESS, AND RESULT


Web Pages
View Controllers/ Handlers
Other Intermediate Processes
Database Access
Enter by Link
JobInputVC
Get Display Info
DataProvider
Already existing information for job.
Entity used depends on job.
Data to Display
Data
Input
Displayed data,
File Upload Button, Input Fields, Submit Button.
FileUploadHandler
Upload
Delete
On "Delete" also deletes temporary storage
JSON
File Parser (Eg: Preparor)
File Contents
Success/ Errors
Relevant file data
DataProvider
TempStorage Entity
Temporary Storage ID (passed by session)
JobInputHandler
Form Data
Access by Temporary Storage ID
Relevant file Data
Create New DB entities, update JobPatameters
DataProvider
JobParameters, JobStatus, JobResults Entities
New Thread
JobLauncher
JobProgressVC


Web Pages
View Controllers/ Handlers
Other Intermediate Processes
Database Access
JobProgressVC
JobLauncher
DataProvider
Progress
Initial Data
Run Analysis Process
Update Progress
JobParameters, JobStatus, JobResults Entities
JobProgressHandler
updateStatus
Called every 3 seconds, stops and updates Input when overallStatus is updated to indicate job process completion.
Check JobStatus
Return JobStatus Data
JSON
Write Result
Update overallStatus in Progress
Enter by Link
GET Parameter in URL
JobResultVC
Get Result Data
Delete JobParameters and JobStatus if necessary here.
Result Data
Result
Result Data

DATABASE OBJECTS

WHAT GOES INTO THE DATABASE?

JOB PARAMETERS

- objectId - Needed for database use.
- jobId - Random UUID representing the job that created the object.
- created - Time the object was created.
- Other variables important to the function of your analysis.

JOB STATUS

- objectId and jobId - Same function as in Job Parameters.
- created - Time when the object was created.
    - lastUpdated - The last time when the object's values were changed.
    - finished - If a record of Job Status is kept, this marks the last time when Status was updated.
- jobName - Name to display on processing page.
- tasks - An arraylist of objects representing each step in the job analysis process. These are embedded in Job Status.

TASK COLLECTION

- taskName - The name of the task.
- taskStatus - Integer representing state of the task. “0” for not completed, “1” for completed, “-1” for failed.
- taskId - A randomly generated ID number to reference this task by.
- subTasks - An arrayList of TaskCollections in this one.
- subTaskLevel - The depth of the current task within the taskCollection tree. Helpful for data visualization.

JOB RESULT

- objectId and jobId - Same function as in Job Parameters.
- Other relevant data for visualizing and displaying the result of your analysis.

TEMPORARY STORAGE

- objectId - Needed for database use.
- created - Time when the object was created.
  - lastAccessed - The last time when the object's values were changed.
- okToDelete - If the storage is to be kept for a longer period of time this can help with DB maintenance.
- Any other fields that are necessary to store file information. Generic objects cannot be stored in the database without a defined class.

CONFIGURATION

INSTALLATION AND RUNTIME SETUP

WEB XML CONFIGURATION

- Context parameter “file-upload” should point to a directory in which to store uploaded files temporarily. Remaining files in this directory should be manually deleted as needed.
- Environmental variable “mongoPath” points to the location of the server where the database is stored.
- Environmental variable “dbName” should be the name of the database used to store data associated with this web project.

LIBRARIES (INCLUDED IN TEMPLATE)

- Morphia 1.2.1
- Mongo Java Driver 3.0.4
- Commons-fileupload 1.3.2
- Commons-io 2.5
- JSTL core
- JQuery UI
- JSON for Java

Morphia: https://oss.sonatype.org/content/repositories/releases/org/mongodb/morphia/morphia/

Mongo Java Driver: https://oss.sonatype.org/content/repositories/releases/org/mongodb/mongodb-driver/ (can probably download a newer version without loss of functionality)

Fileupload: http://commons.apache.org/proper/commons-fileupload/download_fileupload.cgi

Commons: http://commons.apache.org/proper/commons-io/download_io.cgi

JSTL: http://stackoverflow.com/tags/jstl/info taglibs from http://java.sun.com/jsp/jstl/*

Jquery UI: https://jqueryui.com/download/all/, can build custom themes.

Jsonorg jar - needed for working with JSON in ajax response, I don't remember the website I downloaded the initial one from, so here's this: https://github.com/stleary/JSON-java

TOMCAT

- You will need to install Apache Tomcat in order to run the web project in a web. Version 8.0.35 was used in this project.
- $CATALINA_HOME environmental variable needs to be set to point to the root of your Tomcat Installation.

MONGODB

- You will need to install the MongoDB Community Server to create and store information for this project.
- As with Tomcat, you'll need to set environmental variable, $PATH, to the directory of your MongoDB installation.
- If you are not using the default location (/data/db) as your database directory, you can set a new one with dbpath.
- Launch the mongod daemon to start mongodb and launch the mongo process to access the database from the command line.

TEMPLATE ALTERATIONS CHECKLIST

- ❑ Add input fields appropriate for your project to JobInput.jsp.
- ❑ Add javascript to prevent form submission without required fields.
- ❑ Alter the JobParameters.java entity to store relevant field values.
- ❑ If you are using FileUpload, create a class that can:
    - ❑ Extract the relevant information from your file.
    - ❑ Return true in a function if data was successfully parsed.
    - ❑ Return a list of error codes (of value other than -1, 0, or 1) representing any mistake which occurs during parsing.

CONFIGURATION

- ☐ If you are using FileUpload, alter FileUploadHandler.java and TempStorage.java to temporarily hold relevant field values.
- ☐ In JobInputHandler.java, create JobStatus and JobResult entity objects containing values you need to save. Create a TaskCollection object to place into a JobStatus object to show progress on the Progress.jsp visually.
- ☐ In JobLauncher, complete your process step by stem, making sure to updateJobStatus for every relevant stage of the process. Write the results of your process to the JobResult object. Make sure the placeholder Thread.sleep() functions are removed beforehand.
- ☐ Make sure the last thing you do in JobLauncher is update the overallStatus of the job to "1" (meaning complete).
- ☐ Update RobResultVC.java and JobResult.jsp to display result information from your job.
- ☐ Edit the config.properties file to include strings for any error codes caused by incorrect parsing of input.