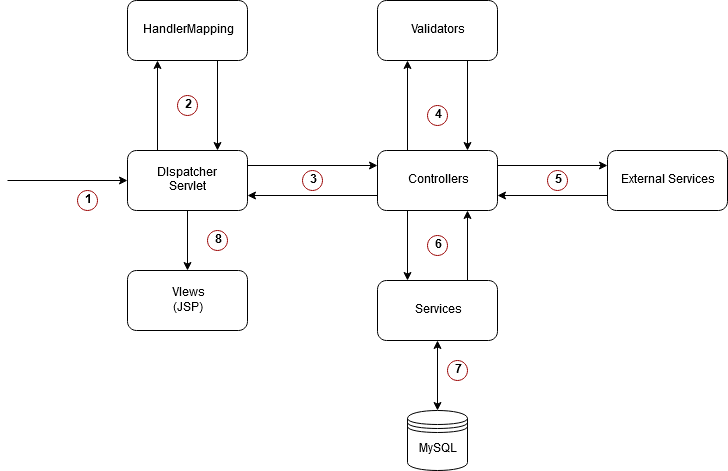
**Task 1: Architectural Design**

General Overview of Architecture



1. The dispatcher receives a HTTP Request from the client(web, mobile).
2. The dispatcher utilises the handler mapping object to pattern match and requests a controller name/method.
3. The dispatcher then calls the controller method with relevant data if it exists or throws an exception otherwise.
4. The controller Utilises the validators to ensure the passed in data transfer objects are acceptable before utilising services.
5. External Services such as Facebook/Google oath are called and utilised if necessary, using HTTP API requests.
6. The passed in DTO after validation is passed to the service layer where business logic is applied.
7. The service layer utilises repositories and relevant connections to persists data by using a JDBC connection.
8. Upon receiving the model data from the controllers, the views are rendered with the help of view resolver.

**Architectural decisions:**

During the first sprint of the application it was concluded that JAVA would be the primary language utilised. This was because every developer was comfortable with working it and the fact that java is platform independent, hence allowing our application to be used on mobile devices as well as desktops. The views were developed as Java Server Pages with HTML being the mark-up language and bootstrap/CSS to create the stylize the pages to be more presentable.

User password are encrypted using BCrypt, which uses a very strong hashing algorithm with randomly generated salts, making our application very robust and secure to any attacks. Furthermore, the use of Spring Bootstrap/Spring Security has in-built parameterization for its CRUD methods, thereby preventing attacks such as SQL injection.

The unique identification of each story is stored as a java long which would allow more stories to be stored. This would be useful as the branch feature can quickly scale out of control with entries.

Access control has been applied on various levels. Only authenticated users can access features such as the dashboard, liking/disliking and creating/contribution to stories. This access control is achieved using spring security, which forces un-authenticated users to register and be authenticated before accessing any views with access control.

Finally, MySQL is used as the Relational Database Management System to store and organise critical application information. This was used over Neo4j for the sprints due to the familiarity with SQL by the developers. However, for better performance and in real world software we would have used Neo4j as it does provide features for querying graphical data efficiently.

**Task 2: Software Design**

**One of the features that our client requested was the ability for users to upload their own cover images for stories they create. There were two solution we decided on:**

* Store the images as blobs (Binary Large Objects) in our database.
* Use an external service to handle hosting of uploaded images.

**Saving the Images as blobs.**

**A picture containing electronics

Description automatically generated**

1. Post request performed by authenticated user, where a story POJO is passed in and an optional multi part file.
2. The multipart file if present is passed to the blob service to process the file.
3. The blob service processes the multipart file and return a blob back to the controller.
4. The blob and the POJO is passed to the persist service to handle save operations.
5. The persist service then assigns the blob to the POJO and saves it to the MySQL database using repositories.
6. The persist service then returns a success or failure as a result of the save operation.
7. The controller uses the response from the persist service and responds with the correct view name to render.

**Now to render these images in the client we would need an additional method in one of our controllers where the blobs are turned back into images and then returned to the image tag.**

Issues with storing the images:

* The database will become very huge very fast, degrading performance.
* Before the image is displayed to the user, the blob must be retrieved , converted back to an image file and then passed to the client to display it. (This back and forth conversions can introduce unwanted bugs/errors).
* Backing up and restoring the database takes longer / more resources.
* Security issues relating to storing .jpg files which could be masked as something else.

**Uploading to Imgur using their API.**

**A picture containing electronics

Description automatically generated**

Explanation of architecture:

1. Post request performed by an authenticated user, where a story POJO is passed in and an optional multi part file.
2. The File is passed to the upload service to upload and return a URL pointing to the image.
3. Upload service performs post request to the Imgur API with the image file.
4. The API then returns a JSON.
5. The upload service extracts the URL of the image from the JSON response if it is present and sends it to the controller method.
6. The POJO object and URL is then sent to the persist service to save to the database.
7. The persist service assigns the URL to the POJO and saves it to the database.
8. Persist service responds with a Success/Failure back to the controller based on the persist operation.
9. The story controller then sends the view to render as a result of the persist service.

Rendering on the client side of uploaded images would then be done like so in a JSP page.

A picture containing object

Description automatically generated where image URL is the URL assigned to the story POJO.

Improvements compared to the previous solution:

1. The database now only saves URL or null object.
2. Less error prone as images are not converted back and forth between blobs.
3. Images are always readily to the client to load.
4. Database size doesn’t increase drastically with each image upload.
5. Images are easily updatable as all that’s needed is to change the URL.
6. Easily expand storage on cloud hosts
7. Less work on the server side required.
8. Easier to maintain and backup.
9. Imgur handles the verification of images and distances the server containing Login Information etc. from the uploading of files to the database.