Important Functionality: User Authentication

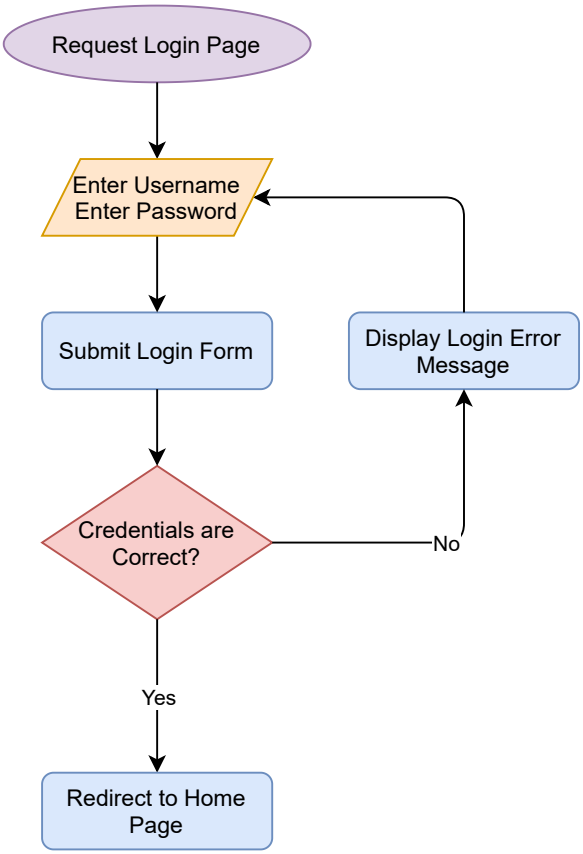
Functional Requirements

- Allow access to webpage with correct credentials
- Allow for different levels of access known as UserRoles
- Prevent unauthorized access to content which is not permitted for certain UserRoles
- Provide logout functionality

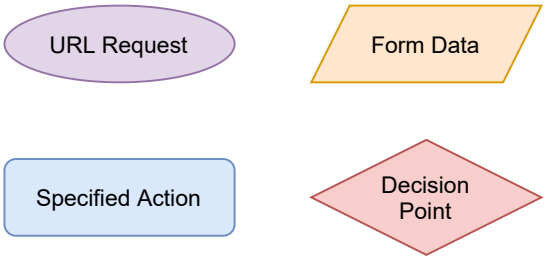
Nonfunctional Requirements

- Must follow web security best practices to ensure that secure information remains secure
- Password must be hashed and salted before checking against salted hash in database
- Logging out must clear cache to prevent data from becoming available to other users

System Model



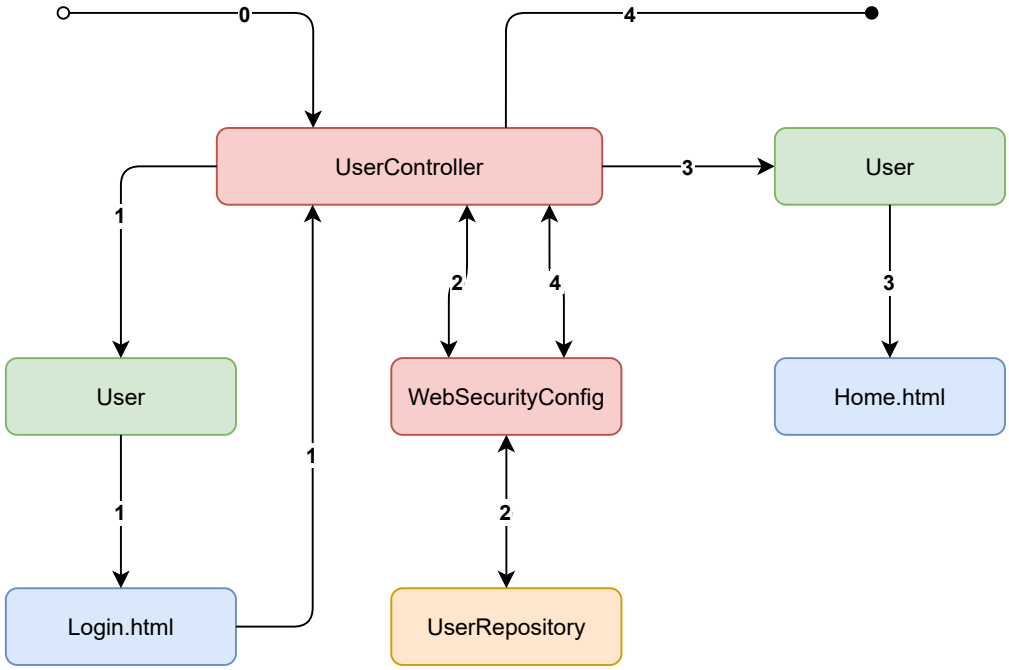
Symbol Key:



Architectural Design

Symbol Key:

Note: File names are listed in box, file types are .java unless otherwise specified.



0. Login Get Request

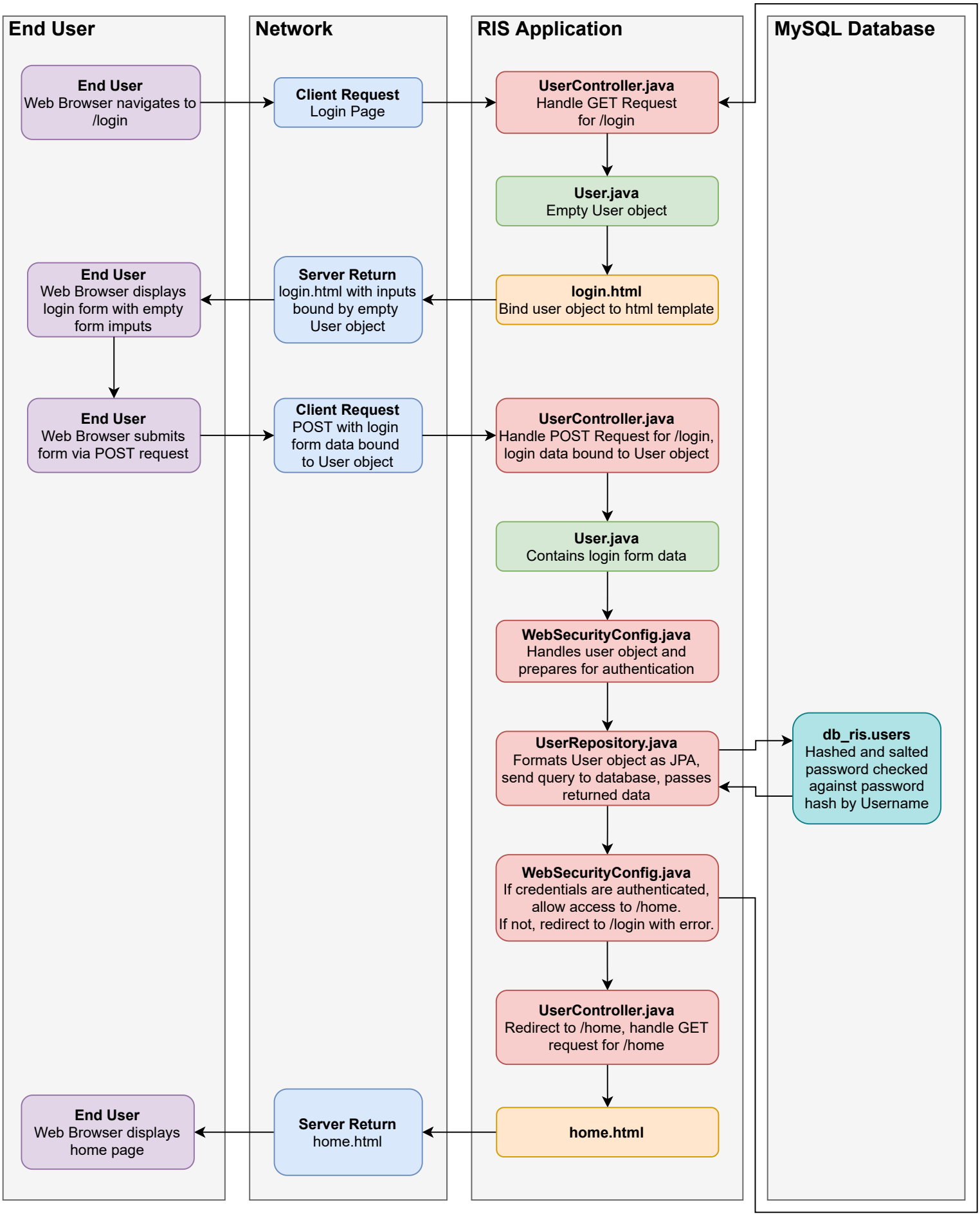
1. Login Process: Send User data object to Login view, on form submission send form-data to controller.

2. User Authentication: Send User data response object from form-data to WebSecurityConfig which passes the User data to the UserRepository model to check against the database. Returns authentication results if successful (User data, UserRole).

3. Successful Authentication: Send User data object returned from the authentication method passed back from WebSecutityConfig to the Home view.

4. Logout: Logout of system and clear logged in user data from WebSecurityConfig.

Detailed Design



Important Functionality: File Uploading / Downloading

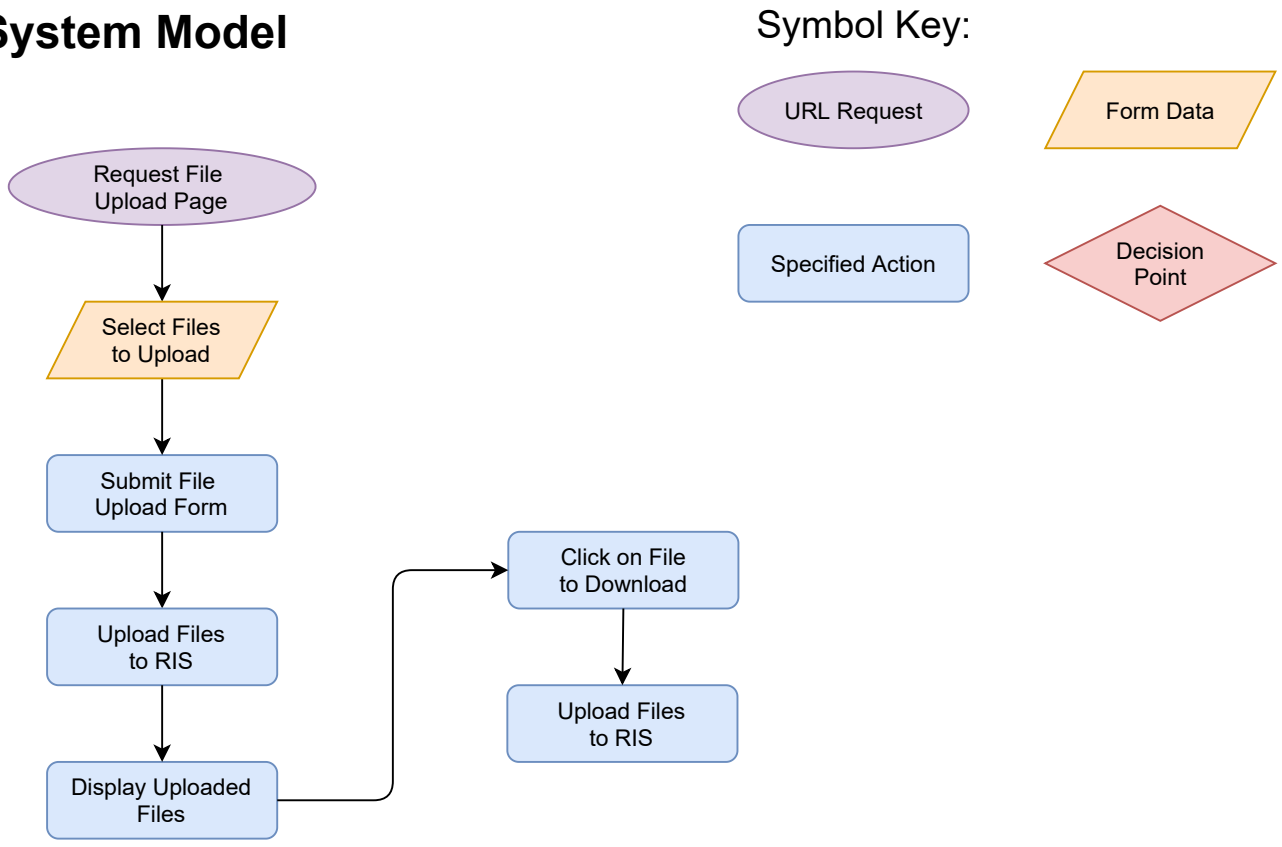
Functional Requirements

- Allow user to upload files from local computer
- Store files in specified RIS upload location
- Track which order is associated with file uploads
- Display file uploads associated with order
- Download files from RIS

Nonfunctional Requirements

- Files must be encrypted on upload and decrypted on download to prevent sensitive information from becoming easily accessible
- A single file upload should only be associated with one order, but multiple file uploads may be associated with a single order.

System Model

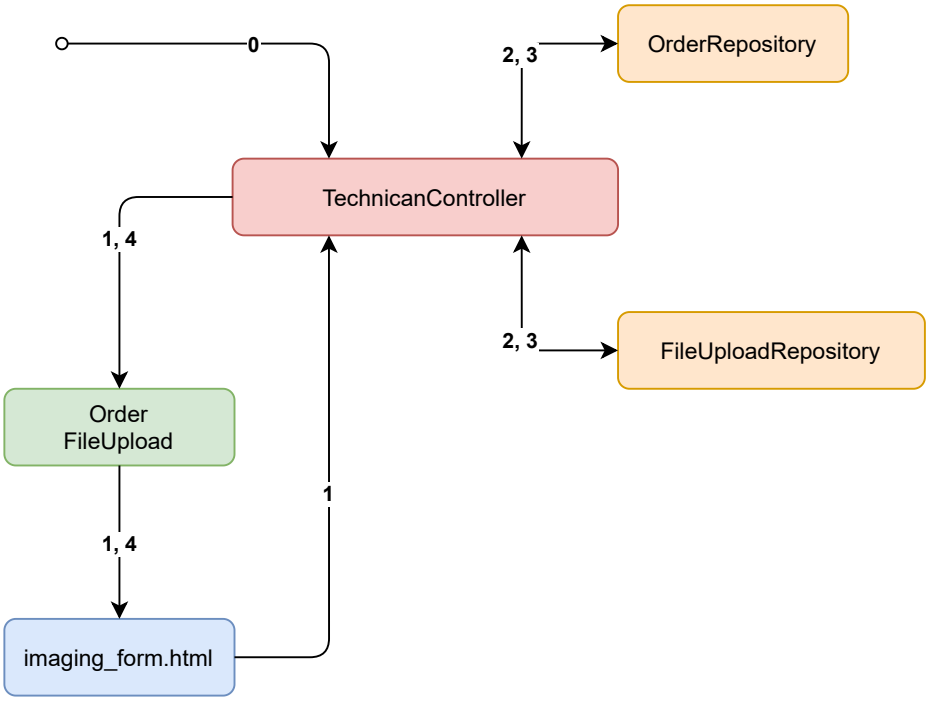


Symbol Key:

Architectural Design

Symbol Key:

Note: File names are listed in box, file types are .java unless otherwise specified.



0. File Upload Page Get Request

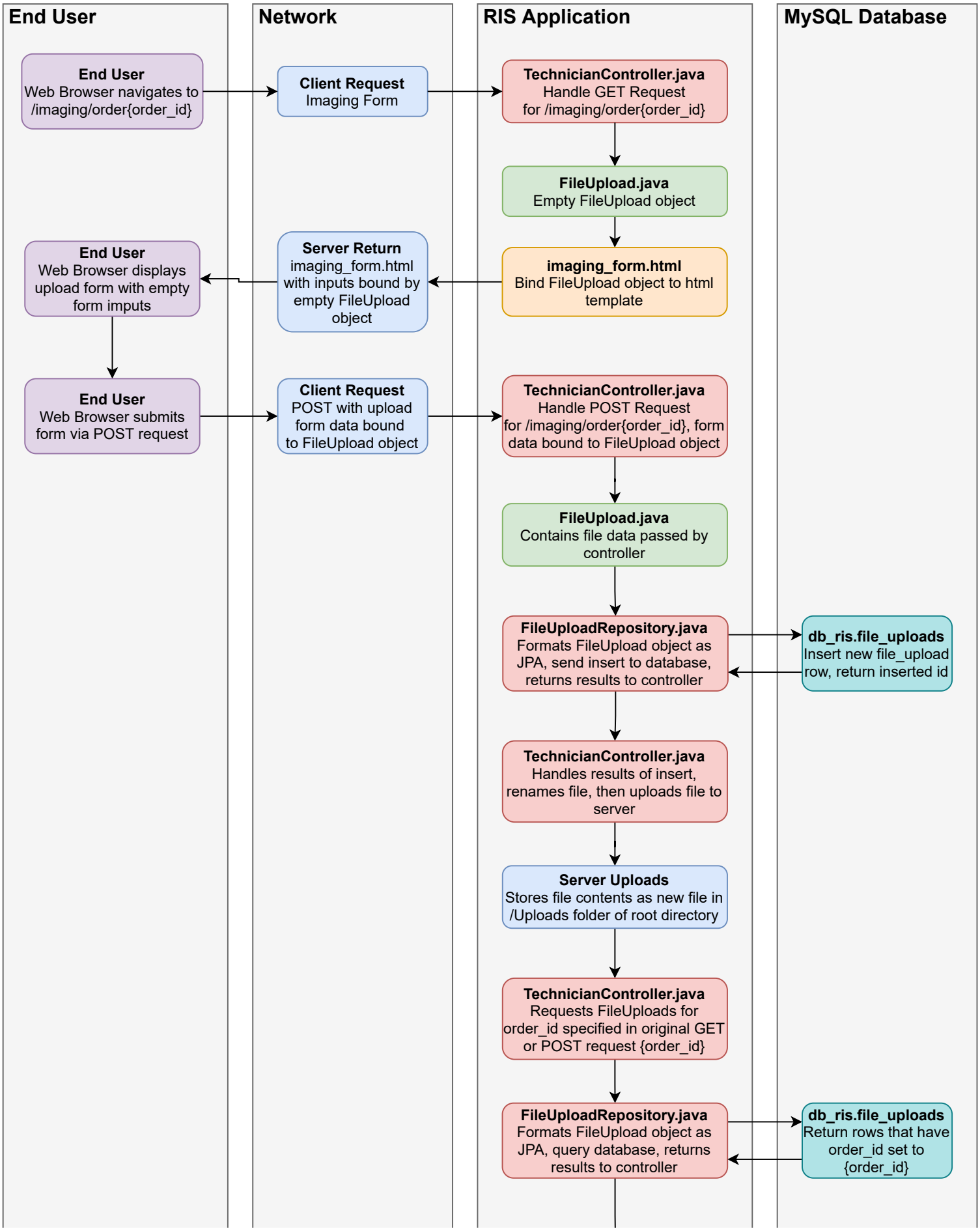
1. Imaging Order Form: Send Order and FileUpload data objects to imaging_form view, return interactions and form-data to controller.

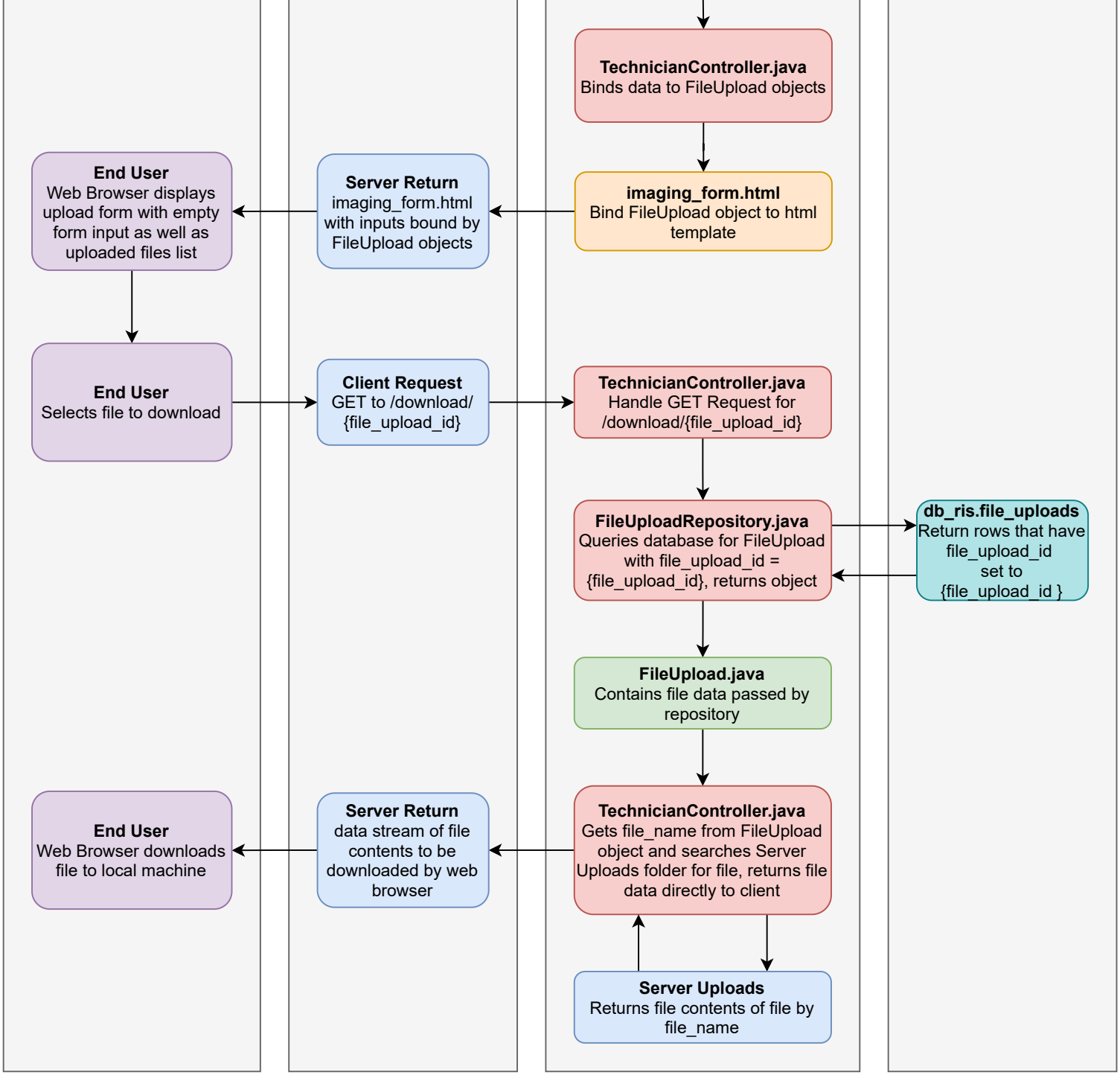
2, 3 Order Model Interactions: Query/Send Order data object to OrderRepository, return results to controller.

2, 3 File Upload Model Interactions: Query/Send Order data object to OrderRepository, return results to controller.

4. Display File Uploads: Send Order and FileUpload data objects to imaging_form view, return interactions and form-data to controller.

Detailed Design




TechnicianController.java
Binds data to FileUpload objects
imaging_form.html
Bind FileUpload object to html template
Server Return
imaging_form.html with inputs bound by FileUpload objects
End User
Web Browser displays upload form with empty form input as well as uploaded files list
End User
Selects file to download
Client Request
GET to /download/ {file_upload_id}
TechnicianController.java
Handle GET Request for /download/{file_upload_id}
FileUploadRepository.java
Queries database for FileUpload with file_upload_id = {file_upload_id}, returns object
db_ris.file_uploads
Return rows that have file_upload_id set to {file_upload_id }
FileUpload.java
Contains file data passed by repository
TechnicianController.java
Gets file_name from FileUpload object and searches Server Uploads folder for file, returns file data directly to client
Server Uploads
Returns file contents of file by file_name
Server Return
data stream of file contents to be downloaded by web browser
End User
Web Browser downloads file to local machine