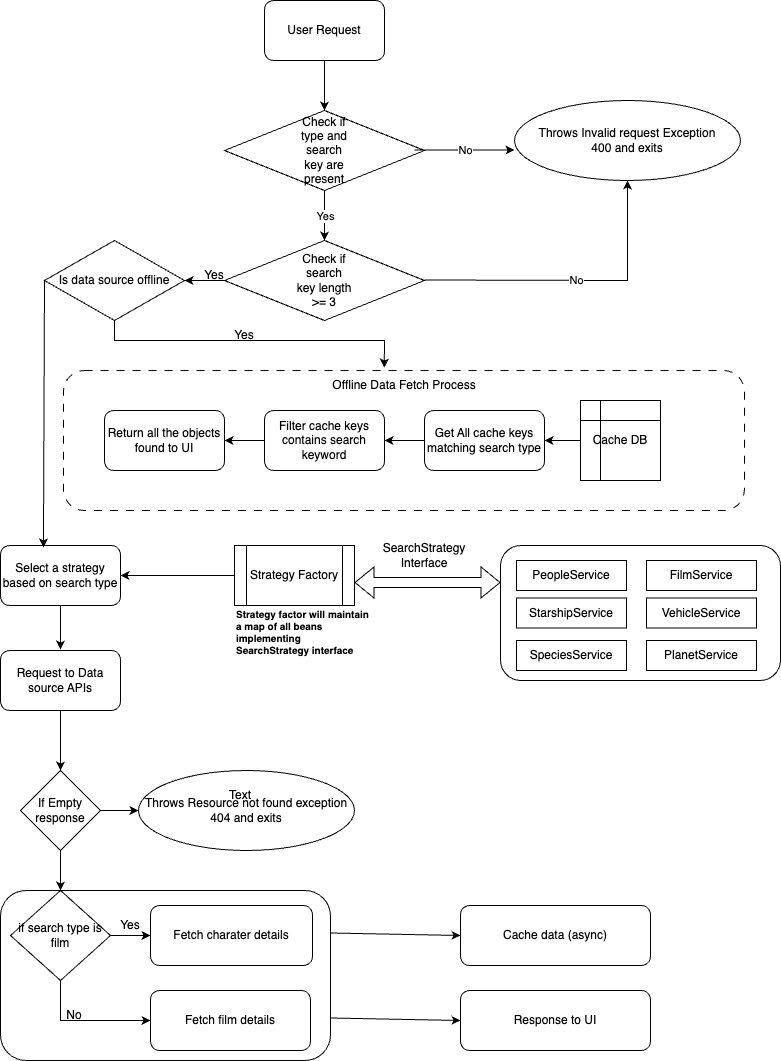
**Design Pattern Used: Strategy Pattern**

By using strategy pattern, we can switch between different implementations dynamically and with ease. In our system, the services implementing strategy interface return same base object. This makes the patter open/closed, which will allow us to add more strategies with affecting existing services.

**System Flow:**



**Modules:**

1. rest-client – Common rest client utility which leverage generic types to be plugged in into any other service.
2. map-cache – This is a simple hash-map based cache system. Can be integrated into any other system. Also, annotation-based caching too available. @MapCache annotation is very basic version of @Cacheable.
3. star-wars-data – Provides data mapping to source API
4. starwars – Main Spring Boot Service

**Implementation:**

The advice class throws 3 exceptions: InvalidSearchRequestException, ResourceNotFoundException and Exception. The errors are wrapped into ErrorResponse object.

For API response the search result data types extend BaseResponse type which gets wrapped into ResponseWrapper. Response leverage generic types for flexibility.

*SearchStrategy:*

The SearchStrategy interface is implemented by PeopleService, FilmService, SpeciesService, PlanetService, StarshipService and VehicleService.

The StrategyFactory maintains a map with beans implementing the SearchStrategy interface. So based on search type, we can fetch which a service for searching data.

The DataSourceService, makes API calls to SWAPI source to get data for the keyword that being searched. The API call implementation is part of rest-client module.

Once the data is fetched from source, we do 2 things.

1. If the search type is film, we fetch character details of each films. If the character is already fetched, it will be in cache, for subsequent times data will come from cache. If the type other than film, we fetch film details for each result item. Caching is done here also for film data.
2. Now we cache each result item for future requests and offline use.

For offline, all the keys starts with type will be fetched and then they will be filtered with only ones that contains the search keyword in it. Now fetched data will be returned to UI.

Swagger:

<http://localhost:8080/swagger-ui.html>

UI:

<http://localhost/>

**Steps to run application:**

Using docker-compose:

1. Update “volumes” value with the local system’s path to the file “service-config.yml” which is placed within service folder. Update highlighted part with your system path.
2. volumes:
3. - ./services/service-config.yml:/app/server-config.yml
4. Run the below command:

docker compose -f star-wars-compose-yaml up –build -d

Note: --build is building the image and -d for detached mode.

Using Jenkins:

In the Jenkins file inside “starwars” spring boot project, update the highlighted key-values with local system values.

1. DOCKER\_REGISTRY = 'localhost:6000'
2. SPRING\_BOOT\_APP = 'starwars-service'
3. SERVICE\_LOCATION = '/Users/ramkbharathi/publicis-sapient/services'
4. CONFIG\_LOCATION = '/service-config.yml'
5. IMAGE\_CONFIG\_LOCATION = '/app/server-config.yml'
6. PATH = "/opt/homebrew/bin:/opt/homebrew/sbin:/Library/Frameworks/Python.framework/Versions/3.11/bin:/usr/local/bin:/System/Cryptexes/App/usr/bin:/usr/bin:/bin:/usr/sbin:/sbin:/var/run/com.apple.security.cryptexd/codex.system/bootstrap/usr/local/bin:/var/run/com.apple.security.cryptexd/codex.system/bootstrap/usr/bin:/var/run/com.apple.security.cryptexd/codex.system/bootstrap/usr/appleinternal/bin:/usr/local/share/dotnet:~/.dotnet/tools:/usr/local/bin/docker:$PATH"

For UI, update the below highlighted fields:

DOCKER\_REGISTRY = 'localhost:6000'

UI\_APP = 'star-wars-wiki'

SRC\_DIR\_PATH = "/Users/ramkbharathi/publicis-sapient/ui/star-wars-wiki"

PATH = "/opt/homebrew/bin:/opt/homebrew/sbin:/Library/Frameworks/Python.framework/Versions/3.11/bin:/usr/local/bin:/System/Cryptexes/App/usr/bin:/usr/bin:/bin:/usr/sbin:/sbin:/var/run/com.apple.security.cryptexd/codex.system/bootstrap/usr/local/bin:/var/run/com.apple.security.cryptexd/codex.system/bootstrap/usr/bin:/var/run/com.apple.security.cryptexd/codex.system/bootstrap/usr/appleinternal/bin:/usr/local/share/dotnet:~/.dotnet/tools:/usr/local/bin/docker:$PATH"

Jenkins pipeline: (Same for both apps)

1. In the Jenkins dashboard click new item in left side bar
2. Now enter a name for pipeline and select “Pipeline” from below options
3. In the next page, navigate to down where you will see a text area with name scripts. In that textarea copy the contents of Jenkinsfile and save it.
4. In following page, click on Build now option from left sidebar.
5. Now do the same for another app also.

Docker repo:

Execute the below commands if need to create local docker repo:

docker pull registry:2

docker run -d -p 6000:5000 –name registry –restart=always registry:2