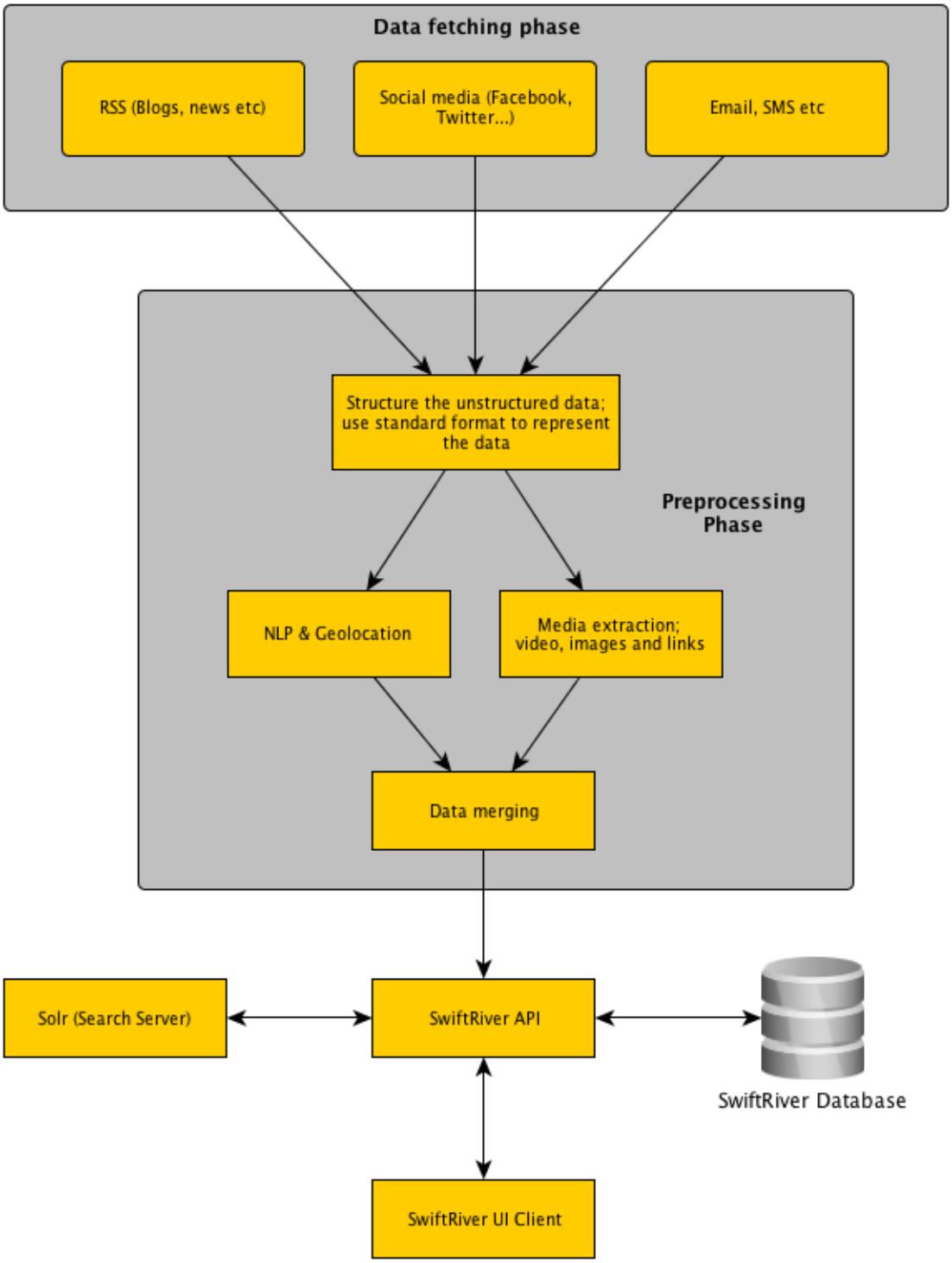
SwiftRiver Architecture

Overview

The application is comprised of the following components:

Component	Description
Content fetchers/crawlers	A set of background applications/daemons that fetch drops from the various sources e.g. RSS, Twitter, Email etc.
Metadata extractors	These perform semantic (named entity extraction & subsequent geocoding of any place names that are encountered) and media extraction (links and images) once the drops have been fetched and structured by the content fetchers
Drop queue processor	Keeps track of drops as they come in from the content fetchers and forwards them to the various pre-processing stages - semantic extraction, media extraction, rules processing. Once a drop has gone through all pre-processing stages, it is reassembled and posted to the API for final storage in the DB NOTE: While drops are undergoing pre-processing, they're maintained in a persistent RabbitMQ queue
API	Posts and retrieves data to/from the database, handles user authentication and authorization, updates the search index
Search Server	Handles all full text and geo search functions and is periodically updated (by default, every 30s) with any new data
UI (web) client	A web application for interacting with the API; fetches data and presents it to the user

The relationship/interaction between these components is shown in the diagram below:



Technical Architecture

The current version of SwiftRiver runs off mixed stack that is primarily powered Java, PHP and Python; Version 1 of the software was based on a LAMP stack.

Component	Underlying Architecture
API	Java - Spring Framework

UI Client	PHP, JavaScript & CSS - Kohana MVC (v3.3) and Backbone JS for the UI
Content Extractors	Python - custom applications and are not based on a particular application development framework
Dropqueue Processor	Java - Spring Framework
Database	MySQL Database Server
Search	Java - Apache Solr Server
Messaging & Queueing	RabbitMQ